UTRECHT UNIVERSITY

MASTER THESIS

Efficient and Effective Super Resolution for Single Images using Deep Learning

Author:
Steven van Blijderveen,
5553083

Supervisor:
Prof. dr. ir. A.C. Telea
Second Supervisor:
dr. M. Behrisch
Company Supervisor:
Remco Koopmans

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Faculty of Science
Graduate School of Natural Sciences

June 17, 2022

Abstract

Super resolution is a class of techniques that aims to increase the resolution of an image. Commissioned by the company Stapes IT this thesis aims to create an efficient and effective super resolution method for single images using deep learning. To tackle this issue, a comparison between different deep learning super resolution solutions had to be made to find the best framework to improve upon. It was found Real-ESRGAN is the best starting framework for its fast inference and high quality output. To improve this framework to the needs of Stapes IT the option of continuous floating point upscaling factors was added. This was done by cascading the existing factor 4 and factor 2 models to a factor 8. To quantitatively measure the quality of the models an image data set relevant to the company was created and tested using the Peak Signal to Noise Ratio (PSNR) and Multi-Scale Structural Similarity Index Measure (MS-SSIM) metrics. For both these metrics, a higher score indicates a better result. Comparison to the existing SDMD model on the custom data set shows our cascaded model performs well with PSNR and MS-SSIM scores of **20.794** and **0.836** compared to **18.042** and **0.730** for SDMD. Using a combination of Lanczos4 interpolation and linear image blending the model is able to upscale to any floating point value between 1 and 8 from the existing 2 models. Tested on the same custom data set the models show a slowly declining average MS-SSIM score linearly correlated to the height of the upscaling factor starting with **0.984** for an upscaling factor of 1.5 till **0.836** for factor 8. We conclude that our method proves that creating super resolution images with continuous upscaling factors is practical, delivering decent high resolution images and can be implemented in a digital environment for commercial and practical use.

Contents

Chapter 1

Introduction

Single image super resolution (SISR) is one of the many challenges in the world of imagery today that deep learning hopes to tackle. SISR is a class of techniques that is used to enhance the resolution of an image. The algorithm or model creates new pixels for an image without any prior knowledge of that image. By enhancing the resolution of an image (also called upscaling), you get the opportunity to either enlarge the image while maintaining a certain quality, or zoom in on specific parts of the image without these parts becoming too pixelated to see any details. Since the knowledge about deep learning keeps growing, the impact on the field of super resolution is also increasing as deep learning algorithms seem to be able to handle this challenge well [30].

To achieve super resolution, two major deep learning techniques can be used. The first technique is the use of convolutional neural networks (CNN's). Since 2014 this technique made its entrance in the world of SISR with the introduction of SRCNN [4]. After this many variations on CNN's have been constructed to solve the super resolution challenge. Topics of research and improvement in this area could be designing new architectures, building new loss functions or use different types of learning strategies.

The second technique is a certain variation of a CNN: a generative adversarial network (GAN). Since GAN's are relatively new deep learning techniques being introduced only in 2014 [5], a lot of research can still be done in this area. With the introduction of the SRGAN in 2017 [12] it was shown that this specific deep learning technique could also provide solid solutions for the super resolution challenge. GAN's perform particularly well on images that have lots of texture, but also tend to create more artifacts. The area's of improvement for this technique are equal to those in normal CNN's: finding new architectures, loss functions or learning strategies.

1.1 Stapes IT & Tinify

The company Stapes IT[1], who are the technical owners of the company Tinify[2], also have an interest in owning a good solution for the super resolution challenge. Tinify is specialized in compressing images for large websites, independent hobbyists and anything in between. Besides their compression tool Tinify also offers the possibility to automatically resize images. Up until now this resizing tool only contains a way to downsize images[3]. This is because enlarging images poses the problem which

[1]http://www.stapesit.com/
[2]https://www.tinypng.com/
[3]See https://tinypng.com/developers/reference/python, the 'Resizing Images' section

super resolution tries to solve: the quality of an image will drastically decrease the more you try to enlarge it. Since customers have the need to enlarge images while also maintaining the image quality and to have a new image adjustment feature on the website, Stapes IT decided to invest in a single image super resolution algorithm.

1.2 Research Question

In order to find this model for Stapes IT, the following research question has been formulated:
"What is an efficient and effective deep learning approach for generating super resolution images that can be deployed in a practical online solution for nonspecialist end users?"

1.3 Requirements

There are many different ways to tackle the SISR problem, for this thesis it has been decided that only deep learning techniques will be taken into consideration. This choice was made to make the research area more narrow and because the most possibilities to improve and investigate lie in this area. Besides the existence of many different techniques to encounter super resolution, there are also many different solutions for this problem which can all be sufficient based on the requirements. SISR-algorithms made with deep learning techniques can for example be trained to perform well in a specific domain of images, or build to be extremely lightweight or fast. As the customers of Tinify and the developers of Stapes IT have their own particular needs and wishes, a list of functional and non-functional requirements will guide us in finding the algorithm suitable for Tinify. Each requirement will get a certain priority rating, based on how important it is for both Tinify and Stapes IT. Requirements can be labeled *critical*, meaning it's of the highest importance this requirement is met. If requirements aren't critical but still much appreciated if achieved they'll be labeled *important*. Finally requirements will be labeled *optional* if they are nice to have but no deal breaker if they are missing.

1.3.1 Functional Requirements

The functional requirements for this project are defined as the requirements which the algorithm needs to fulfill to cater to the needs of the customers of Tinify. For each of the following requirements the needs of the customer are considered, as well as technical feasibility and feasibility in regard to time constraints.

Domain - Since the customers of Tinify are very diverse, not one domain can be named that contains all possible images up for enlargement. As this is the case, the algorithm should not be domain-specific and should be able to upscale any image irregardless of the contents of that image. This requirement is critical, if the model can't upscale an image of any given domain it's not valuable to Tinify.

Quality - This is an important requirement, meaning the aim is to get the best image quality there is, but only if all critical requirements are met. Some specific areas of image quality are more important than overall quality though, in particular: images should ideally not have any artifacts nor blurred or noisy regions on them. The quality of super resolution algorithms is usually measured by PSNR and/or SSIM metrics. These metrics are fast and easy but have the disadvantage that they can give results which are not corresponding to human perception. For this project human perception is the only quality assessor that really matters but since this is a subjective metric it's hard to use it to compare different models. For this reason human perception will be used to provide us with a first insight on quality only, but objective metrics such as PSNR and SSIM will be used to determine which models perform best. The exact values which determine whether a model is good enough will be examined during the comparison of the models.

Upscaling factor - The upscaling factor determines how many pixels the algorithm will add to the input image. One of the implicit constraints is that the input image is between 25x25 and 1000x1000 pixels. With this in mind if the input image is 100x100 pixels and we have an upscaling factor of x3 the output image will be 300x300 pixels. Since we want to be able to perform super resolution on any image, as opposed to having one specific class of images, it will be harder to keep the quality of the output images high for a large upscaling factor. On the other hand the upscaling factor also should not be too low, since customers needs possibly will not be fulfilled then. To find a good balance on this matter the model should be able to perform upscaling up to a factor 8, but the main focus in terms of quality control will be on the smaller factors (up to factor 4). Customers should be able to choose their own upscaling factor as a parameter, preferably a floating point value between 1 and 8 with up to 2 decimals precision. This might pose problems however, which need to be investigated. If upscaling with this much precision is not possible while using neural networks, the upscaling factor can also be, though less preferred, an integer between 1 and 8. These final obstacles restrict this requirement to the important category, since there is no way of telling if floating point valued upscaling is a realistic target at this stage of the research.

Speed - The upscaling process should be fast. Usually there are more operations on an image which have to be executed synchronously and users want their files as soon as possible. This does not mean the upscaling process should be real-time, but it also should not take longer than a few seconds on the Google Cloud Platform for an output image of roughly 2 megapixel. The matter of speed is only important to the actual interference time of the algorithm, not to the learning part, this may take as long as needed. To have a fast model is very nice, but it would not be a problem if upscaling an image would take up to a minute, which is why this requirement is optional.

Image extension handling - The compression and resizing algorithms of Tinify can handle `.jpeg`, `.png` and `.webp`, which means the super resolution algorithm should be able to handle those extensions too. Though this requirement is important to Tinify, it's also one that can be accomplished at all times by simply transcoding the images. Because of this possibility this requirement doesn't have much priority and is deemed optional.

Proof of quality - The paper describing the model should provide enough proof of the quality of the model. Proof can be provided as examples of output images, results from super resolution contests/challenges or results of quality metrics for certain image databases. Included in showing proof is also showing what goes wrong, e.g. artifacts created by the model or specific textures the model can't handle. Having this proof matters very much, since it would be a waste of time if the chosen model performs a lot worse than expected, therefore this is a critical requirement.

1.3.2 Non-functional Requirements

The non-functional requirements are defined as those requirements that describe how a system performs while doing the work described by the functional requirements. This mostly entails how the model will fit the needs of the developers and how it will fit in the current digital infrastructure. All these requirements have been carefully discussed with the CTO and company supervisor for this thesis: Remco Koopmans.

Programming interface - One of the strengths of Tinify is that the company offers relatively simple API's. The preferable outcome of this project is that the user can simply add the preferred upscaling factor in the API and receive an upscaled image. As previously mentioned, this is already possible for downscaling images, but not for upscaling images yet. This is a critical requirement as it's the only way Tinify offers these type of products to their clients.

Digital infrastructure - The model should be able to run on the Google Cloud Platform. All of Tinify's techniques are cloud-based, meaning this solution should be able to run in the cloud too and satisfy the constraints given by this environment. This is another critical requirement since the company won't change their programming environment for this model.

Cost of ownership - This is hard to measure concretely, but an important aspect to Tinify. The model should be easy to maintain and be lightweight so it doesn't cost much to have it running non-stop. In practice this means that a trained instance of the model is preferably not larger than 50MB and should fit in the Rust framework Tinify uses. The developers at Stapes IT do have experience converting Python-based trained neural networks to Rust so they fit in the framework, so this should not be a problem. To make the model easy to maintain unit testing should be possible. This can be done by having some standard images and their desired quality metrics after upscaling, which we can have automatically tested whenever there's a change in the code. Since it's so hard to measure, this is an optional requirement. The limit of 50 MB is not set in stone and there should always be a possibility of converting from any language to Rust.

Scalability - Tinify runs in a containerized elastic cloud environment. This means the solution should be able to scale horizontally using the Tinify cloud orchestration. To be able to scale horizontally the model should be stateless, which means we need to be able to call a trained instance of the model at any given time. For models made with popular packages like Tensorflow or PyTorch it's always possible to save an instance of the trained model, which means this requirement should not pose any problems. That being said, this is a critical requirement. It would not be practical if the model needs to be trained or initialized in another way any time it has to provide new upscaled images.

Replicability - In order to replicate models and be able to adjust them, a few things are needed. Preferably the source code is online available, but in some cases a detailed description of the model could also suffice. Furthermore the image database(s) for the training phase should also be publicly available to make sure the results are equal. Lastly the exact training process should also be described, as this can significantly influence the performance of the model. Since it's of critical importance the model can be reproduced, this is a critical requirement.

1.4 Structure of the Thesis

The rest of the thesis will be outlined as follows: In chapter 2 the related work will be discussed by going over various deep learning super resolution models and see if they satisfy our requirements. After that the model that satisfies most requirements or satisfies them in the best way will be chosen as model to improve upon. Chapter 2 will conclude with the possible improvements that will make the chosen Real-ESRGAN model suitable for Tinify. Chapter 3 then continues by explaining how the improvements were made on the Real-ESRGAN model. This will include a section about the original upscaling factors of Real-ESRGAN, the attempt to add more upscaling factors, an explanation of how cascading the models works and an explanation how continuous upscaling factors were achieved. Chapter 4 will handle the results of the experiments that were introduced in chapter 3 to see if the proposed improvements work and improve the model. This chapter will start by stating some general information about the way inference is performed and on what images this is done. Thereafter the results of experiments performed on the cascading model will be shown and finally the results of the experiments performed testing the continuous upscaling factors will be shown. In chapter 5 the performance of the final solution will be discussed using the original requirements as mentioned in section 1.3. Finally the research question will be answered and the possibilities for future work will be discussed.

Chapter 2

Related work

2.1 The models

Since there are already lots of super resolution models available created by other researchers, the question arises why Tinify has the need for a new one. Most importantly, Tinify has a very specific set of requirements that are likely not yet met by the models that are already out there. Each already existing model probably has its own pro's and con's in regards to these requirements. For comparing these pro's and con's some of the requirements mentioned in section 1.3 can be disregarded. The image extension handling requirement and the scalability requirement can easily be met for any model, which is why they won't play a role in the comparison of models. Furthermore it's not likely any model already has the programming interface Tinify uses, which is why this won't be taken into account as well.

In the following sections a few models have been picked to compare. The models will appear in chronological order and have been picked based on a few criteria:

- All models have to be deep learning models, as explained in section 1.3.
- The model is currently a state of the art model, or the model is a building block for a current model.
- The model is present in recent benchmarks.
- At first impression, the model seems to fulfill at least some of the requirements mentioned in section 1.3. E.g. a model that focuses solely on faces won't be picked since it doesn't fit the domain constraint.

For each of the models there will be a brief introduction explaining how the model works and what makes it special. After that the pro's and con's will be described to be able to compare it with the other models and conclude which of the models is most suitable for Tinify to adjust.

2.1.1 SRCNN

SRCNN was, as mentioned in chapter 1, the first deep learning approach to tackle the SR challenge. To perform super resolution, Dong et al. [4] heavily based the modeling of their CNN on traditional sparse-coding-based SR methods [32] that were state of the art at that time. In figure 2.1 a comparison between SRCNN and a sparse-coding-based method can be seen. Being based on sparse-coding-based methods, the CNN was a rather simple model which performed three steps in order to get from a low resolution to a high resolution image. Before any of these three steps were taken the low resolution image was first upscaled by a bicubic algorithm.

Since this is a preprocessing step it won't change during the learning phase of the CNN and thus is not included in the three steps. The first of the three steps is patch extraction and representation. This is done by a convolutional layer which maps patches of the image to a vector with the size of the feature maps. Secondly they perform non-linear mapping, which uses a convolutional layer to map each of the previously found feature map vectors to vectors which represent the high resolution patches of the image. Finally they reconstruct the image by taking these high resolution patches and averaging them, since the patches will overlap. For the learning phase of the CNN the Mean Squared Error (MSE) was used as a loss function, which means the model is optimized for PSNR scores.

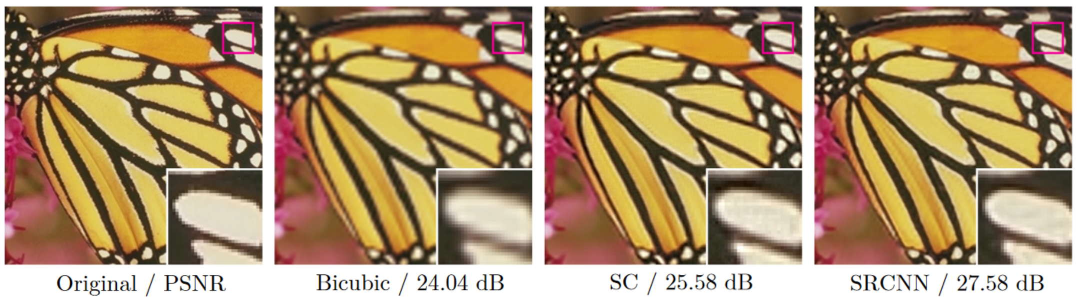

FIGURE 2.1: A comparison of the SRCNN model[4] with a sparce-coding-based method (SC) and the standard Bicubic algorithm

Domain - The paper states they used the same data sets for training and testing as Timofte et al. [24]. This is however a bit unclear, as that paper mentions all images are available on their project page[1], though the images available there are only in grayscale. Besides that Dong et al. mention they have a training set of 91 images next to their test sets Set5 [2] and Set14 [33], but the only other set on the project page of Timofte et al. is the B100-set which has 100 images. Nonetheless the images in the test sets are very diverse, which supports the idea that this algorithm can perform super resolution on images of different domains.

Quality - The paper only mentions PSNR as a quality measure, which doesn't provide us with a lot of information in terms of quality since we are looking for good human perception scores. SRCNN averaged the best PSNR scores at that time for upscaling factors of 2, 3 and 4 on Set5 (36.34, 32.39 and 30.09 respectively) and an upscaling factor of 3 on Set14 (29.00). However, judging from the images in the paper, the resolution isn't high enough to satisfy the needs of this project, which makes sense since this is the first super resolution deep learning model.

Upscaling factor - In the paper it is mentioned that SRCNN can handle upscaling factors of 2, 3 and 4. For each factor a separate network is trained with the same architecture but different variables.

Speed - In the paper the inference time per image is mentioned in a large table. All inference times are well below one second as measured on their machine (Intel CPU 3.10 GHz and 16 GB memory), which satisfies this requirement.

Cost of ownership - The implementation of SRCNN is available on their project page[2], which shows that the full Matlab code is a little over 7MB. Though this would

[1]https://people.ee.ethz.ch/~timofter/ACCV2014_ID820_SUPPLEMENTARY/index.html
[2]http://mmlab.ie.cuhk.edu.hk/projects/SRCNN.html

need to be transcoded to Rust or Python, the network architecture is so small that it would definitely fulfill this requirement.

Replicability - In terms of replicability there is only one thing missing: the original training set of 91 images. However, later in the paper they test whether using a larger data set[3] improves the performance and this set is available upon registering. Besides this the code is available as mentioned and could also be reconstructed from the paper.

Proof of quality - In the paper sufficient examples are shown of the performance of SRCNN, which show the model was performing well at that time, but is now no longer relevant in terms of quality.

2.1.2 SRGAN

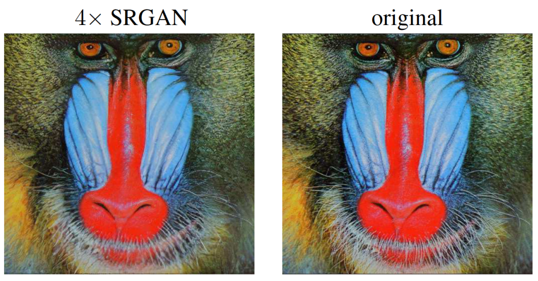

FIGURE 2.2: A comparison between an image upscaled 4 times by SRGAN[12] and the original image

Another breakthrough in the usage of deep learning techniques for SISR was the use of GAN's. As mentioned in chapter 1 the first appearance of GAN's in the world of super resolution was SRGAN. A GAN typically consists of two parts: a generator network and a discriminator network. The discriminator tries to discriminate between super resolution (generated) images and high resolution (non-generated) images. The goal of the generator is to generate images that fool the discriminator, making it believe they are high resolution images. Being the first super resolution GAN, Ledig et al. [12] created both architectures based on different GAN models from other image-related domains. Most importantly, the generator network exists of a number of residual blocks with a specific layout as proposed by Gross and Wilber [6]. This creates a very deep network, capable of producing much detail. This shows well in figure 2.2 where the texture of the baboon fur is replicated very well, the difference with the original image is almost only visible when zooming in. The discriminator network is a pretty straightforward feed-forward CNN model, with a Sigmoid function as its final layer to output a value between 0 (SR-image) and 1 (HR-image). A big change Ledig et al. made by which they diverted from previous work was no longer using MSE as a loss function, since it favours high PSNR scores and not necessarily high perceptual scores. Trying to get higher perceptual scores, they constructed a loss function which consists of two parts: content loss and adversarial loss. The content loss, which they also call VGG-loss since it's based on the VGG network [22], is defined as the euclidean distance between the image feature representations of the reconstructed image and the reference image. The adversarial loss is created to favour images that are more natural looking, based on the probabilities the discriminator returned during training.

Domain - For training the model, 350 thousand images were randomly picked from the ImageNet dataset [19], which ensures the model will work for images from any

[3]ImageNet ILSVRC 2013: `https://image-net.org/challenges/LSVRC/2013/` [19]

domain. This is also confirmed by using the test-sets which were also used by SR-CNN (Set5, Set14 and B100). These sets are all very diverse yet SRGAN performed well on them.

Quality - As mentioned Ledig et al. didn't want to focus on PSNR scores solely. This is why they compared their model based on MOS: mean of opinion score. They let 26 people rate images from various models including their own on a scale from 1 (bad quality) to 5 (excellent quality). With MOS-scores of 3.58, 3.72 and 3.56 on Set5, Set14 and B100 respectively the SRGAN model scored significantly higher than any other model in the comparison. This while scoring only average or slightly below average on PSNR (29.40, 26.02, 25.15) and SSIM (0.8472, 0.7397, 0.6688) metrics. This shows that for the requirement of quality we shouldn't focus too much on PSNR and SSIM. It also shows the GAN-model is, in terms of quality, a suitable model for Tinify.

Upscaling factor - SRGAN only works with an upscaling factor of x4. This means some work would need to be done in order to make it suitable for Tinify.

Speed - Although nothing about inference time is mentioned, Ledig et al. do mention in their 'Future work' section that the model could perform at real-time if the network architecture would be made a bit more shallow. This would however reduce the qualitative performance a little. This gives the impression that the inference time of the model is within our set limits.

Cost of ownership - Though no official source code of SRGAN is published online, there are some reconstructed models available online[4,5] which stay well under the 50 MB limit.

Replicability - The previously mentioned reconstructed models show building the actual model should not pose any problems. With all data sets publicly available it should be possible to reproduce the images shown in the paper. The only small remark is the exact training set can't be used again, as it were 350 thousand images randomly sampled from the ImageNet data set.

Proof of quality - Some examples are shown in the paper and in particular the MOS results show the quality of the images is good. There is however mention of bad quality when upscaling text in images or structured scenes. These images are not shown, nor are other images on which possible artifacts are present.

2.1.3 EDSR/MDSR

EDSR (Enhanced Deep Super Resolution Network) [13] is a prize-winning modification on SRResNet [12]. SRResNet is a deep residual network proposed in the same article as SRGAN, heavily based on the deep residual network by He et al. [7]. Since SRResNet didn't modify the original residual network by He et al. much, Lim et al. [13] decided to take a closer look at this network and make some changes which make it more suitable for super resolution. By doing so the EDSR architecture won 1st prize in the NTIRE 2017 super resolution challenge [25]. The most important difference between EDSR and SRResNet is that Lim et al. removed the batch normalization layers. This doesn't only improve performance, but also saves approximately 40% of memory usage during training. With this change and by using residual scaling [23] with factor 0.1, a deeper network could be created for EDSR

[4] `https://github.com/eriklindernoren/Keras-GAN/tree/master/srgan`
[5] `https://github.com/eriklindernoren/PyTorch-GAN/tree/master/implementations/srgan`

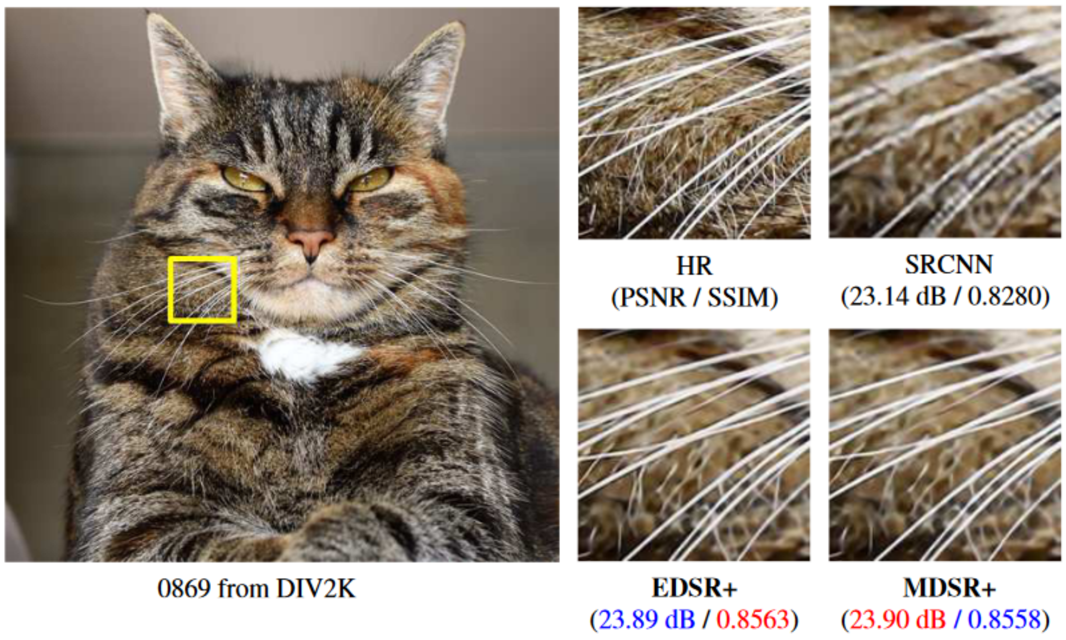

FIGURE 2.3: A comparison between EDSR[13], MDSR and SRCNN for x4 upscaling

which performs much better than SRResNet. Besides the EDSR model Lim et al. also introduced MDSR (Multi-scale Deep Super Resolution Network) in the paper. This is a single network which can upscale images for x2, x3 and x4 upscaling factors simultaneously. The EDSR network has 1.5 million parameters for each of these upscaling factors, which counts up to 4.5 million in total, but the MDSR network only has 3.2 million parameters making it a lot smaller than all EDSR networks combined. MDSR is capable of performing almost equal to EDSR, scoring only a few hundredths worse on both PSNR and SSIM metrics for the DIV2K validation set. A comparison between EDSR, MDSR and SRCNN can be seen in figure 2.3, which shows both models are a big improvement on SRCNN and are almost indistinguishable from each other.

Domain - For training the EDSR network, Lim et al. used the DIV2K data set [1]. This is a very diverse data set which ensures the model should be able to perform super resolution on any given image. The provided images in the paper confirm this.

Quality - Winning the NTIRE 2017 super resolution challenge, this model proved to be of high quality. Compared to various other models, EDSR scored the highest PSNR and SSIM scores for upscaling factors of x2, x3 and x4 on various test sets (Set5, Set14, B100, Urban100 [8] and the validation set of DIV2K).

Upscaling factor - There are three different versions of the EDSR model, constructed and trained for x2, x3 and x4 upscaling factors. More interesting is MDSR which can perform these upscaling factors simultaneously, thus is the first one to somewhat satisfy the multiple upscale factor requirement.

Speed - Though the training speed (8 and 4 days for EDSR and MDSR respectively) is mentioned in the paper, nothing is said about the inference time.

Cost of ownership - The code for EDSR and MDSR can be found on Github[6] as mentioned in the paper. The code however is written in Lua, which would pose some problems in terms of transcoding to Python or Rust. Upon further examination, the authors of the paper also published their code in Python on Github[7], which removes that problem. That page also provides us with the full models, which shows the MDSR model is 30MB, while all three EDSR models are around 160MB. This means MDSR would fulfill the requirement, but interestingly enough even a single EDSR model would not.

Replicability - Since the code is available on Github, the data sets are all publicly available and the training process is well described in the paper, it should be no problem to replicate the results of this paper.

Proof of quality - Winning the NTIRE 2017 super resolution challenge is a big proof of quality, besides that also various examples are shown in the paper. Without specifically mentioning them, some small errors are also shown in the example images which show there's room for improvement.

2.1.4 ESRGAN

ESRGAN [28] is the price winning improvement on the pioneer SRGAN model, hence the name Enhanced-SRGAN. The enhancement is made by improving the SRGAN model on three different features, while focusing on human perception scores rather than PSNR scores. These three features are the network architecture, the adversarial loss and the perceptual loss. They improved upon the network architecture by introducing the Residual-in-Residual Dense Block (RRDB) without batch normalization as the basic network building unit. The RRDB is basically a multi-level residual network using dense connections which makes the network deeper and more complex, enabling the possibility of better performances. Besides the introduction of the RRDB they also removed the Batch Normalization layers, which were responsible for creating some artifacts. After these improvements on the network architecture, they also improved the adversarial loss function. Instead of having the discriminator determining whether an image is real or fake by outputting a score between 0 (fake) and 1 (real), the new relativistic discriminator outputs a score between 0 and 1 where 0 means the image is more fake than the real data and 1 means the image is more real than the fake data. This minor change ensures more detailed textures and sharper edges in the generated images. Finally the use of perceptual loss was changed by constraining on features before activation rather than after activation. By changing this the loss function can supervise more activated neurons which leads to a better performance and on top of that this fixes some brightness issues. In figure 2.4 a comparison between ESRGAN and the already discussed models can be seen, which shows ESRGAN performs much better on the detailed textures.

Domain - For training the model, the creators of ESRGAN used three large high resolution data sets (DIV2K, Flickr2K[8] and OutdoorSceneTraining [31]) which are all openly available. The images in these data sets are very diverse, which leads to the model being able to enhance very diverse images, thus fulfilling our first requirement.

[6] `https://github.com/limbee/NTIRE2017`
[7] `https://github.com/sanghyun-son/EDSR-PyTorch`
[8] `https://cv.snu.ac.kr/research/EDSR/Flickr2K.tar`

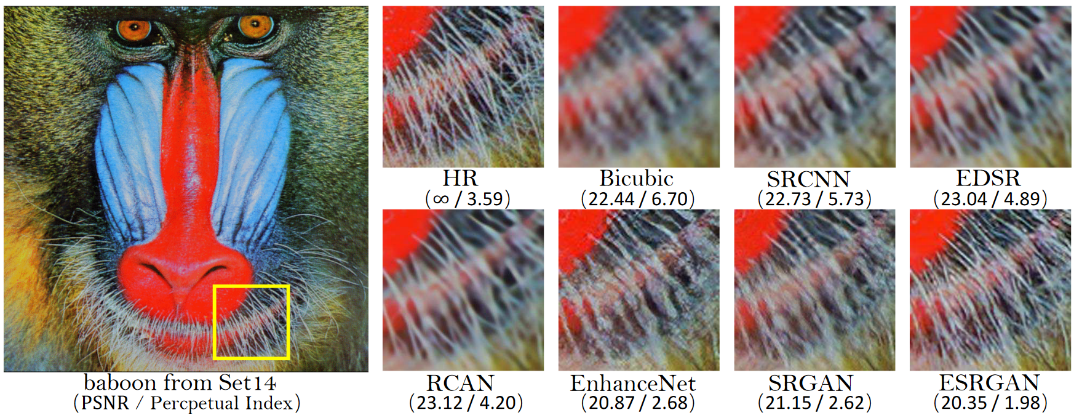

FIGURE 2.4: A comparison between ESRGAN[28] and various other models for x4 upscaling

Quality - This model won the PIRM-SR challenge [3] based on a perception index metric (PI), which is a combination of Ma's score [15] and NIQE [16]. All competing models were grouped based on their RMSE score, which is practically the same as the PSNR score. After grouping the models, the model with the lowest PI score on the PIRM validation set[9] would win the challenge. With a PI score of 2.040 for an RMSE score of 15.15 they won the challenge.

Upscaling factor - All examples in the paper use an upscaling factor of 4, which leaves us with some questions for other factors. In the paper there is no mention of using different upscaling factors, so the model probably would need some work to be able to handle different integer factors and even more for handling floating point factors.

Speed - The speed of both the training of the model and the inference is not mentioned in the paper.

Cost of ownership - Upon examining the ESRGAN github page[10], it was found that the stripped down model is 33,4 MB. This is well within the limits of what Tinify prefers, so it shouldn't cause any problems.

Replicability - With both the code and all the data sets the model was trained on available online, we should be able to reproduce the exact values ESRGAN gets in the paper. The only thing missing is the validation set on which they won the PIRM challenge, which means we can't reproduce the scores they show for the challenge.

Proof of quality - There are some examples of images in the paper which show the quality of the model. Besides that winning the PIRM challenge shows the quality they claim the model has is valid. Missing in the paper are examples of mistakes/artifacts the model creates and general points for improvement.

2.1.5 Impressionism

The developers of the Impressionism model [10] found that the creation of the low resolution images using Bicubic kernels was not realistic and was responsible for blur and noise in the constructed high resolution image. This is why Ji et al. [10]

[9]Should be available on `https://pirm.github.io/`, but unfortunately their link doesn't work anymore

[10]`https://github.com/xinntao/ESRGAN`

focused on developing a model which is better suited for constructing the low resolution images. Their Real-SR (Realistic Degradation for Super-Resolution) model creates a low resolution image from a high resolution image in three steps. The first step is to downsample the image using a bicubic kernel, which is where other models both begin and end their downsampling. In addition to that the Real-SR model has a pool of degradation kernels from which they randomly take one and apply it on the downsampled image. Finally they inject noise in the downsampled image, because downsampling using the bicubic kernel removed all the high-frequency information. By collecting noise patches from the original high resolution image and applying them to the downsampled image the noise is both realistic and diverse enough for training purposes. In figure 2.5 it's clearly visible this approach to noise helps the model detect and remove noise when upscaling an image. Ji et al. used the ESRGAN model for performing their actual super resolution and discovered that the VGG-style discriminator didn't work well with these new low resolution images. They decided to use a patch-level-discriminator [36] in order to focus more on local features instead of global ones. Using this new way of generating a training set and a slightly altered ESRGAN model, Impressionism won the NTIRE 2020 challenge on real world super resolution [14].

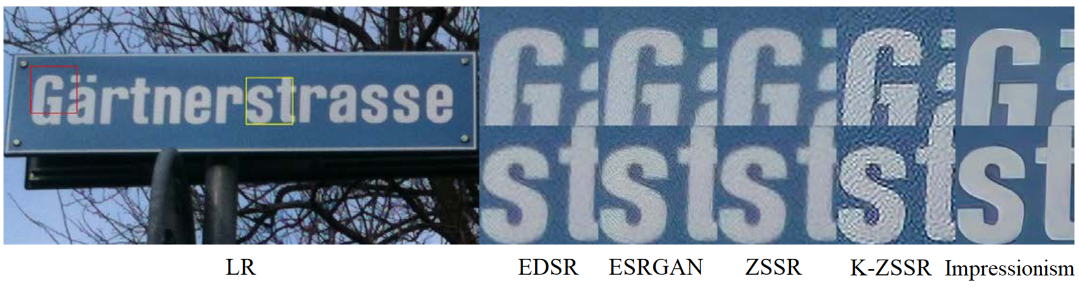

FIGURE 2.5: A comparison of x4 upscaling of the Impressionism model[10] with other models which shows Impressionism is very capable in removing noise.

Domain - Like most models, Impressionism uses the DIV2K and Flikr2k data sets to train. These data sets provide enough diversity to make sure the model can perform super resolution in any given domain.

Quality - Impressionism was compared on some evaluation metrics for the NTIRE challenge. In addition to standard PSNR and SSIM metrics, LPIPS [34] was also used to compare the models. LPIPS compares image features between the generated and the ground truth image, which seems to correlate more with human perception. Though Impressionism wasn't the best model based on the PSNR metric (24.67, 16th place) or the SSIM metric (0.683, 13th place), they did score the best on the LPIPS metric (0.232) and on human perception which was evaluated as a MOS score (2.195). Note that the MOS scale here is inverted as opposed to in the SRGAN paper: the lower the score, the better. These metrics and the visuals provided in the paper show that Impressionism is a high quality model.

Upscaling factor - The upscaling factor is never mentioned in the paper, nor if the model can handle more than 1 factor. The images in the paper seem to be upscaled with a factor 4, and given that other models also mainly use this upscaling factor it's safe to assume that's the only factor the model can handle.

Speed - The paper has no mention of inference speed nor training time.

Cost of ownership - Even though some code for the model can be found on their Github page[11] there is no trained model available there. But since it's a slight alteration on the ESRGAN model there's no reason to believe this model is too large to handle.

Replicability - Considering all the code is available on the Github page and all databases are available too, it should be possible to replicate this research by Ji et al. Besides this they also included pseudo-code for the Real-SR algorithm in their paper and have the code for that algorithm available online too[12], which should make it even easier.

Proof of quality - In addition to the training data sets Ji et al. also used the DPED [9] data set which consists of images taken by an Iphone3 camera. Since these images are are low quality and a real-world example, they are the perfect for comparisons with other models. In these comparisons it can be seen that Impressionism deals much better with real life images than other models. Besides these visuals, winning the NTIRE challenge is of course valid proof of the model having great quality.

2.1.6 Real-ESRGAN

The creators of the ESRGAN recently created an improvement upon their own model, called Real-ESRGAN [29]. The improvement focuses mainly on creating the low resolution image in a different way, similar to the developers of Impressionism. Using these more advanced low resolution images, they also found out the discriminator network is no longer suitable, as did Ji et al. Wang et al. [29] decided to change the VGG-style discriminator not to a patch-level-discriminator but to a U-Net design [20]. This discriminator focuses more on detailed per-pixel feedback, instead of patches of pixels, which it provides to the generator. These changes to the training images and the discriminator network increase the training instability, which is why Wang et al. also implemented spectral normalization regularization [17] to stabilize the training dynamics. In the paper, Wang et al. show some applications of Real-ESRGAN on real life images, such as in figure 2.6 which show some promising results and great handling of different textures.

Domain - As ESRGAN, this model is trained on the DIV2K, Flickr2K and OutdoorSceneTraining data sets. Though these diverse data sets already make sure the model is suitable for any image domain, the new way of fabricating the low resolution images should make the model even more suitable for real life low resolution images.

Quality - Wang et al. claim there are no existing metrics that reflect actual human perceptual preferences on this level of detail, which is why they don't mention any metrics at all. Instead of that they show various examples of the Real-ESRGAN output and compare those to current state of the art models. Real-ESRGAN performs really well in the removal of complicated artifacts present in the low resolution images, but unfortunately GAN training also introduces some new artifacts on a few images. In addition to that the model may amplify complicated artifacts in the test data that are not present in the generated low resolution training data.

Upscaling factor - In addition to the x4 upscaling that ESRGAN could perform, Wang et al. also introduced x1 upscaling (sharpening an image) and x2 upscaling.

[11]https://github.com/jixiaozhong/RealSR
[12]https://github.com/Tencent/Real-SR

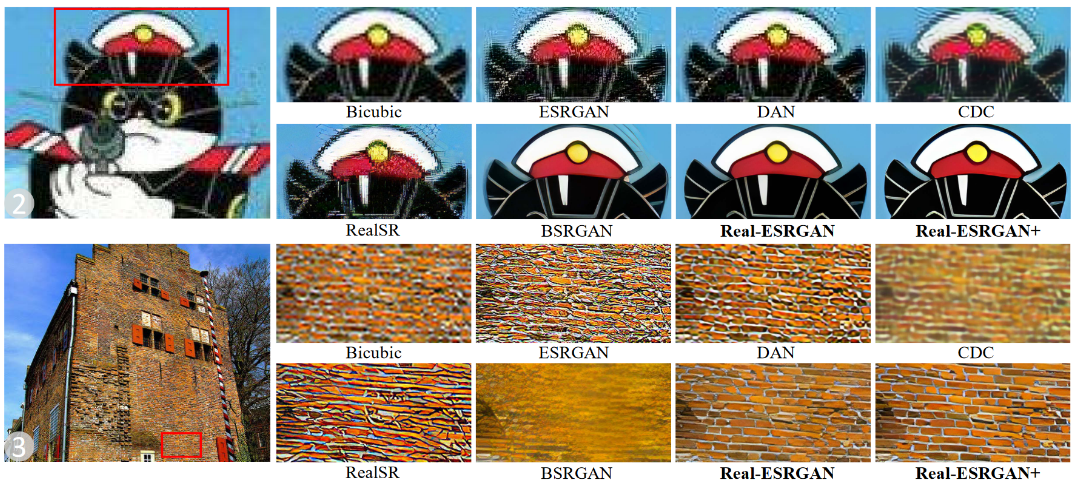

FIGURE 2.6: A comparison of the Real-ESRGAN model[29] with other models not mentioned in this paper which show its application to both a cartoon and a real world image. Note that Real-SR is not the same model as mentioned in section 2.1.5

There is however no mention of how this is done, whether they added different models or the Real-ESRGAN model has some parameters that can be changed.

Speed - The paper has no mention of either inference speed or training duration.

Cost of ownership - As can be found on their Github page[13], with 33,4 MB the Real-ESRGAN stripped down model is exactly as big as the ESRGAN stripped down model, which is within the limits set by Tinify.

Replicability - As mentioned the code is fully available on Github, the data sets are available online as well. This way it should be completely possible to reproduce the results of Wang et al.

Proof of quality - In contrast to the ESRGAN paper, not only images with good results are shown, but also images in which the limitations of Real-ESRGAN are shown. This provides a much better estimation of were the challenges of this model lie and how well it will perform.

2.2 Choosing the model

The six previously discussed models all have its pro's and con's regarding the use for Tinify. Having discussed the pro's and con's for each requirement mentioned in section 1.3 per model, we can summarize them. This summary can be found in table 2.1. Each column of this table will be briefly explained and discussed, followed by a conclusion: the picking of the model to improve. Finally the possible future improvements will be discussed.

2.2.1 Comparing the models

The domain column shows little variance for the different models. As mentioned at the start of this chapter the models have been picked with the requirements in mind, which means all models can handle multiple domains. The slight difference between

[13]https://github.com/xinntao/Real-ESRGAN

	Domain	Quality	Upscaling Factor	Speed	Cost of Ownership	Replicability	Proof of Quality
SRCNN	+	--	-	++	++	++	+
SRGAN	+	+/-	-	+	++	++	+
MDSR	+	+/-	+/-	?	++	++	+
ESRGAN	+	+	-	?	++	++	+
Impressionism	++	++	-	?	++	++	+
Real-ESRGAN	++	++	-	?	++	++	++

TABLE 2.1: A comparison of all the models based on the requirements mentioned in section 1.3. All requirements are judged on a scale which ranges from 'does not fulfill the requirement at all' to 'perfectly fulfills the requirement' (--, -, +/-, +, ++). All results are based on the information mentioned earlier in this chapter.

the two models which scored maximum points, Impressionism and Real-ESRGAN, and the rest of the models is that Impressionism and Real-ESRGAN can handle real life images much better due to their innovative way of creating the low resolution images in the training set.

For the quality column it's clear to see that the quality of super resolution images improved a lot over time. This seems to be the main goal of the super resolution field: getting the best quality there is for upscaled images. With that goal in mind, it is no surprise the two most recent models scored highest in this area. Important note for this column is that the image quality was estimated by human perception based on the images shown in the papers. This was done since quality metrics have shown to sometimes disagree with evaluation based on human perception, while a good score based on human perception is the main goal for this project as mentioned in section 1.3.

The upscaling factor requirement seems to be the hardest requirement to fulfill. None of the models come close to the original requirement, which was being able to upscale an image for any given floating point value in the range of 1 to 8. Most models can perform super resolution for only one upscaling factor and need a differently trained model for another upscaling factor. The only exception to this is MDSR, which is capable of upscaling an image using three different upscaling factors at the same time. This is a slight improvement over the other models, but still doesn't come close to our original requirement. Since non of the models can really fulfill this requirement, the challenge of making upscaling possible for floating point values will be one of the improvements that will be made upon the picked model, as will be discussed in section 2.3.

The column for speed has a lot of question marks, as none of the more recent papers mention inference time anywhere. This should not pose a problem though, as it's known from experience that inference time for CNN's and GAN's is usually within a few seconds which fulfills the requirement mentioned in section 1.3.

The cost of ownership for all models seems to be equal, though this requirement is a little hard to estimate. It is certain that all trained models stay within the 50MB size requirement, but how easy or hard it is to maintain the models is not something that can be estimated up front.

In terms of replicability all models score full marks. For each model the complete code has is available online and descriptions of the training phase are available in the paper. Besides this all data sets used are available as well, which should make replicating the models possible.

At last the proof of quality requirement. There is not much difference between the models here. For each model at least some examples of output images were shown in their paper. Most of the discussed models also participated in super resolution challenges, which perform more unbiased checks of quality. Finally there's one model that stands out in the proof of quality column: Real-ESRGAN. The reason this model scores full marks is because the authors also extensively discuss the flaws of the model. This can help in assessing the limits of what the model is capable of.

2.2.2 Conclusion

After this comparison it can be concluded that three models stand out in one way or another and might be interesting for the purpose of this thesis.

The first model that stands out is the MDSR model. This model is the only model that can handle multiple upscaling factors at once, albeit those factors are only integer values. Since the quality of images of this model is significantly lower than the quality of the newer models, this model is not a real contender however. The small advantage this model has by being able to handle multiple upscaling factors is not sufficient, especially since different solutions for the upscaling factor problem seem to be needed anyway.

The other two models are the newest two: Impressionism and Real-ESRGAN. In table 2.1 there is not much difference between the two, except for a slightly better proof of quality for Real-ESRGAN. The point these models excel in is quality, and in terms of quality it's hard to estimate the difference between the two. This can only be estimated by observation of the example images, since the Real-ESRGAN model does not have any metrics mentioned in its paper. Upon observation of the images it seems Real-ESRGAN might be able to handle different textures better, but it's hard to compare as both papers use different example images. Despite this difference being hard to assess, a choice has to be made. With the possible advantage in terms of quality for Real-ESRGAN in mind and the fact that the authors more extensively discussed the flaws of the model, Real-ESRGAN will be the model used for improvement.

2.3 Possible improvements

Now that it is clear the Real-ESRGAN model will be the model to improve, the possible improvements can be discussed. Besides the already mentioned problem of the upscaling factor, which will be discussed in the following subsection, there are some other areas in which the model can improve. In the following sections these other improvements are shortly discussed, where some of the improvements will be in the scope of this thesis and some will be mentioned for possible future research.

2.3.1 Upscaling factor

The first and most obvious improvement that has to be implemented to make sure the model fulfills the requirements mention in section 1.3 is the upscaling factor. To fulfill the initial requirement, the model has to be able to upscale an image to any floating point value between 1 and 8. This requirement can be achieved in various ways, where it's important to keep in mind that the improvement doesn't necessarily have to be in the model itself. Since the goal of this thesis is to have the model work in a way that is good for Tinify, some adjustments to the output image can also be made outside of the model but before the image gets returned to the customer. A way this can be realized is having multiple instances of the trained model online, each for a different integer factor between 1 and 8. When the customer asks for an upscaling factor of 2.7, the models with factor 2 and 3 output an image, and an interpolation of some sort is made between these two images resulting in the image with an upscaling factor of 2.7. Note that this can also be combined with an MDSR-like solution. If a way is found to have the Real-ESRGAN model output multiple images at once, each with a different upscaling factor, only one trained instance of the model has to be kept online.

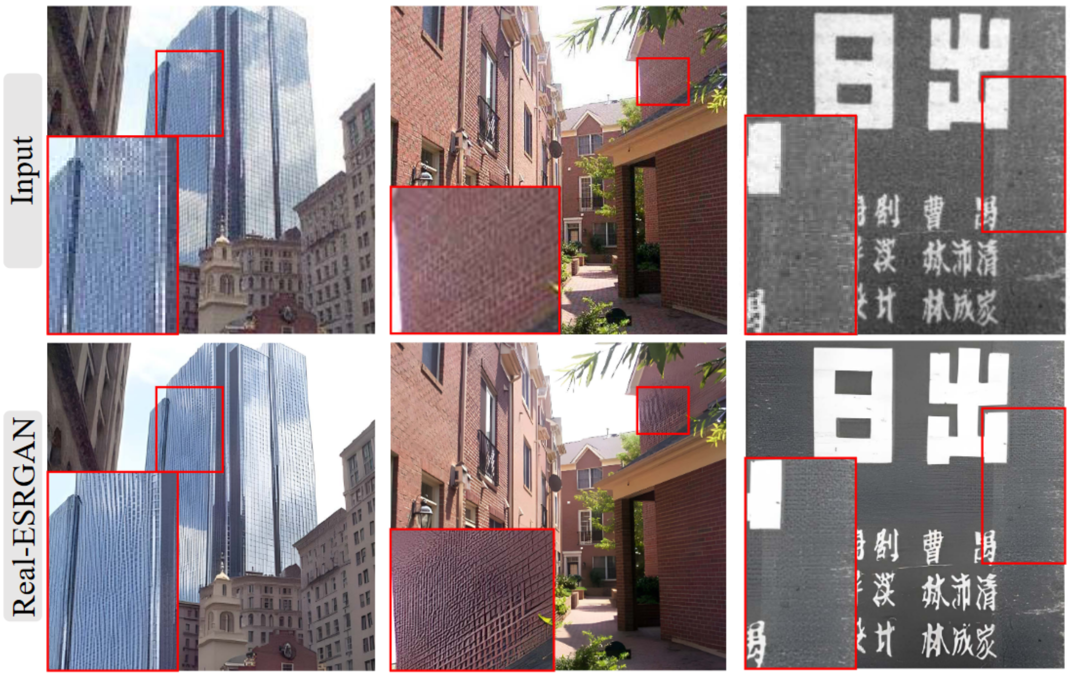

FIGURE 2.7: Examples of limitations of the Real-ESRGAN model[29]. 1. Twisted lines 2. Artifacts caused by GAN training 3. Unknown and out-of-distribution degradations

2.3.2 Quality and Domain

Another point that can be improved is the quality of the output images. As mentioned before, Real-ESRGAN still has some limitations. These limitations are fairly specific and pose no big problem for the intended use of Tinify. Three different examples of these limitations can be seen in figure 2.7. As improving upon these limitations directly is a complicated task and seems to be no major improvement to what Tinify wants to achieve, it's deemed out of scope for this project.

An important question to ask here is: What is a major improvement in quality for Tinify? The answer to that is unknown, since there is no monitoring of what types of images users upload and only a vague idea of images that users want to upscale. For improving the image quality it seems to be crucial to have better estimations of the image domain.

A possible solution could be user feedback training. This means that once the model is online, the users could give feedback on their upscaled image. This can be done by something simple as a Likert scale. With the received feedback it would be possible to get a better idea of the image domain that needs more training. The downside to this is that Tinify wants to use the upscaling model in an API which means the upscaling should be done automatically and possibly in batches. To let the user provide feedback for each individual image would require a whole new user interface and possibly a lot of effort from the user, which is why this idea is undesirable for Tinify.

An idea that is closely related to user feedback training is continuous training. This could be done either supervised or unsupervised. In the supervised variant it would also rely on users (or developers) giving feedback about output images, which was already deemed undesirable. Unsupervised learning for super resolution would mean that the model itself would estimate the quality of the output image. An option could be to use the LPIPS metric which was used in the Impressionism paper, that metric seemed to correlate more with human perception and thus is favourable for the causes of Tinify. Unsupervised learning is a big challenge however and is a high risk technique for a product that has to deliver good results to clients every day. Therefore it is also deemed out of scope for this thesis.

The final idea for improving image quality is to estimate the quality per domain. This would also involve using a quality metric such as LPIPS to estimate the quality of the images. After estimating the quality the images would still need to be categorized to certain domains and as mentioned earlier it's undesirable to have this categorization be done by either the developers or the users. Categorizing images to certain domains could also be done automatically, using multidimensional projection techniques such as LAMP [11]. An idea would be to have a technique such as LAMP automatically categorize all the input images obtained from users and see what domains are most frequently used. If one or two domains stand out we could use the multidimensional projection technique on the training data set as well, find the same domain of images in the training data, and perform training more focused on that domain. This idea seems to be in scope of the thesis project and will be further examined upon.

Chapter 3

Improving Real-ESRGAN

In this chapter some of the possible improvements named in section 2.3 will be implemented to the Real-ESRGAN model and discussed. All mentioned improvements were implemented in a cloud-based test environment, using the Real-ESRGAN framework as provided on Github[1]. Although multiple areas were discussed as possible areas of improvement, this chapter will focus solely on the upscaling factor. This decision was taken due to time constraints and difficulties with the provided framework.

In section 3.1 the process of creating the original upscaling factors as mentioned in the Real-ESRGAN paper will be discussed. Following that in section 3.2 a failed attempt at adding more integer upscaling factors will be analyzed. In section 3.3 a solution to this failed attempt will be suggested with the proposal of cascading models. Finally in section 3.4 the solution for a continuous upscaling factor will be explained.

3.1 Real-ESRGAN original upscaling factors

In section 2.1.6 it was mentioned that there was no information on how Wang et al. [29] achieved the multiple upscaling factors. Upon closer investigation this has to be redacted. The paper mentions very briefly they inverted the pixelShuffle operation as proposed by Shi et al [21]. The original pixelShuffle operation is able to create a high resolution image out of a multilayered image by shuffling pixels from those multiple layers into one image. Wang et al. decided to use the inverted version of this operation, pixel-unshuffle, on input images that are going to be upscaled with a x1 or x2 factor. This way they shrink the image to half (for factor 2) or a quarter (for factor 1) of the input size, so the x4 model will upscale it to the desired scale. This operation is pictured in figure 3.1. This basically means the x2 and x1 model architecture only differ very slightly from the x4 model architecture, just in the first layers. Of course there will also be differences in the weights of the models, but it's interesting to know the basic architecture is similar for all three.

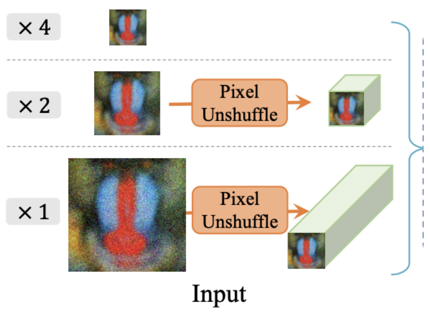

FIGURE 3.1: The first layer of the Real-ESRGAN model, showing a visualisation of the Pixel Unshuffle operation

[1]https://github.com/xinntao/Real-ESRGAN

3.2 Adding more integer upscaling factors

One of the requirements mentioned in section 1.3 was to have upscaling factors available for each floating point value between 1 and 8. A logical first step is to add all integer points between 1 and 8. Once we have that we can either choose tho create more models depending on how well it went, or find some other smart way to be able to upscale for any given floating point number between 1 and 8.

A slight head start was given to us by Wang et al., as they made the x2 and x4 model and their corresponding Discriminator models available in the 'Model Zoo'[2]. Training and fine-tuning those models can be done easily using the framework they provided us with. In the detailed YAML configuration files which can be found in the project, there are several options to alter training in any way we would like. The elaborate way this project is set up is both a blessing and a curse for our own project. The Real-ESRGAN repository also makes a lot of use of the BasicSR [27] repository which was also mainly constructed by Xintao Wang. Due to the strict framework and the complicatedness of the BasicSR repository it was deemed too difficult to add more upscaling factors using the framework Wang et al. provided us with. On the other hand creating a new instance of Real-ESRGAN from scratch would be out of scope for this project due to the limited amount of time, which made it necessary to think about other options and possibilities.

3.3 Cascading models

One solution to create a model with another factor was to take the two models that were already given to us (factor x2 and factor x4) and use them in a cascading way to create a x8 model. What that means is an input image is taken, upscaled by a certain amount and subsequently the output image is upscaled again (and possibly again) to reach a factor 8 output image. There are three ways to do this:

- Use the x2 model three times in a row
- Use the x2 model first and then the x4 model
- Use the x4 model first and then the x2 model

These three takes at a cascading model were thoroughly investigated to find the best performing one, detailed results for those experiments can be found in section 4.2. Finally it was found the cascading model that used the x4 model first and the x2 model second yielded the best results.

3.4 Towards continuous upscaling factors

One of the most important requirements mentioned in section 1.3 was to have a continuous upscaling factor. This means the client should be able to upscale an image for any floating point factor between 1 and 8. Since we have three working models (x2, x4, x8) the image interpolation idea mentioned in section 2.3.1 can be implemented.

How this will work is that two images I_s and I_l get generated from the original input image, where I_s is the smaller image and I_l the bigger one. Since we have

[2]https://github.com/xinntao/Real-ESRGAN/blob/master/docs/model_zoo.md

three working deep learning models, different models will be used depending on the upscaling factor. Evidently for the exact factors of 2, 4 and 8, the corresponding models will directly be used without any interpolation. Factors within the interval $[1, 2]$ are an exception that will be discussed later in this subsection. For factors in the interval $[2, 4]$, I_s will be the original image upscaled using the x2 deep learning model and I_l will be the original image upscaled using the x4 deep learning model. Similarly for factors in the interval $[4, 8]$, I_s is the image upscaled by a factor 4 and I_l is the image upscaled by a factor 8. Then these two images get either up- or down-scaled to the desired dimensions. This can be done by the following formula:

$$f_n = f_o / f_u$$

Where f_n is the new upscaling factor which is used to scale to the correct output, f_o is the original upscaling factor as requested by the client and f_u is the upscaling factor already used (2, 4 or 8). f_n is calculated for both I_s and I_l individually, since f_u will be different for both images. The up- or down-scaling is done using Lanczos4 interpolation, which provides some of the best image quality for a simple filter based interpolation technique [18]. Now we have found two images $I_{s'}$ and $I_{l'}$ which respectively are the up- and down-scaled versions of I_s and I_l using their uniquely calculated factor f_n. What is left is blending those two images. The blending of the images gets done in a linear way, meaning the following formula is used for blending.

$$I' = \alpha * I_{s'} + \beta * I_{l'}$$

Where I' is the output image, $I_{s'}$ is the image generated by the smaller model and upscaled to factor f_o, and $I_{l'}$ is the image generated by the larger model and downscaled to factor f_o. α and β can be defined as follows:

$$\alpha = 1 - (f_o - f_{us}) / (f_{ul} - f_{us})$$

$$\beta = 1 - \alpha$$

Where f_o still is the original upscaling factor, f_{us} the upscaling factor of the small deep learning model and f_{ul} the upscaling factor of the large deep learning model. This way the closer a model is to the actual requested upscaling factor, the more influence it will have on the image.

With these formulas in mind there is one small problem that needs some extra attention: what happens with upscaling factors in the interval $[1, 2]$? As we don't have the original x1 model that is mention in the Real-ESRGAN paper, there are two options:

- Use the original input image as factor x1 and use Lanczos4 interpolation to upscale it to the desired factor to get $I_{s'}$. Thereafter use the Real-ESRGAN model to upscale the input image to a factor 2 and use Lanczos4 to downscale it to the desired factor in order to get $I_{l'}$. Subsequently blend these two images to get I'. In this case we would use all formulas as mentioned before but replace I_s with the input image instead of an image generated by a deep learning model and f_{us} would simply be 1.
- Only use the x2 image and use Lanczos4 interpolation as output. This would mean we only get an I_l which would be the input image upscaled by a factor 2 by the deep learning model. This image would then get downscaled using Lanczos4 interpolation and the calculated factor f_n to $I_{l'}$ and that would be the output image.

Both options will briefly be tested and reviewed in section 4.3.

Chapter 4

Results

In this chapter the results of the experiments as mentioned in 3 will be shown and explained. The overview will start off with section 4.1 which states some general information on the way inference takes place and what kind of image data set was used for testing. Following that, the results of the experiments on cascading models will be shown in section 4.2. This includes the choosing of the best way to cascade the models by testing the different variations on football logo's first and on the Tinify data set later. Thereafter the results of comparing the best found cascading model with the SDMD model are also shown. In section 4.3 the results for the experiments concerning the continuous upscaling factor are shown. This includes the small experiment for upscaling factors between 1 and 2 and the results of a larger scaled experiment concerning the full range of upscaling factors 1 to 8.

4.1 General inference information

For all experiments performed a Google Cloud Virtual Machine with 1 NVIDIA Tesla A100 GPU was used. Most experiments were done using the custom made Tinify data set, which will be further clarified in section 4.1.1. Whenever images needed resizing (for all testing purposes), the Tinify API was used to downscale the images. Since the Tinify API not only resizes images but compresses them simultaneously with their own compression technique, the details on this operation cannot be shared. However, since the same downscaling procedure has been used for all images, it can safely be said that this has no influence on the outcome of the experiments. The only influence this might have is when we would compare the MS-SSIM or PSNR scores with the scores of other models, but since a unique data set is used such a comparison would be illogical regardless.

4.1.1 The Tinify data set

The Tinify data set is a small image data set with 100 images that were handpicked from the websites of Tinify's biggest clients. The data set was created to accurately represent the clients of Tinify, since we want to upscale the images in the best way possible for those clients. The images can be divided into 8 categories, where each category represents one client. Due to privacy rules the data set cannot be made publicly available. To still give some insights on the data, table 4.1 will show some specifics per category. For each category the number of images of that category is shown, adding up to 100 total images. Then some general information about the contents of the images is shared. Lastly the size of the smallest and biggest image in the category is displayed, which can give an insight in the distribution in terms of size within that category. Important to keep in mind is that these sizes are the sizes of the ground truth images, during testing they evidently get downscaled first. In figure 4.1 an example of all image categories is shown.

Category name	# of imgs	General information	Min. size	Max. size
Stickers	15	Multi-colored computer generated illustrations	600 x 336	600 x 600
Clothing	15	Photographs of folded clothing with bland backgrounds	1496 x 1496	1760 x 2640
Car parts	15	Detailed photographs of specific car parts with white backgrounds	400 x 400	6160 x 5008
Tennis	10	Photographs of various tennis equipment with white backgrounds	800 x 800	800 x 800
Interior	15	Various detailed embroidery designs	800 x 800	800 x 800
Football cards	5	Playing cards with images of famous football players	360 x 548	360 x 548
Random objects	15	From toy cars to chainsaws, some with detailed backgrounds	712 x 712	712 x 712
Beers	10	Various images of beer kegs or bottles with transparent background	696 x 696	696 x 696

TABLE 4.1: An overview of the Tinify data set

4.2 Cascading models

As explained in section 3.3 the first step to create more upscaling factors was to use a model multiple times or to use multiple models. To make a cascading model that can upscale with a factor 8 there were three options.

- Use the x2 model three times
- Use the x2 model first and then the x4 model
- Use the x4 model first and then the x2 model

All these models were created in the most sober way possible, meaning no image modifications between passing the image from one model to the next. First the models were judged based purely on perception. Results and specifics on this can be read in section 4.2.1. Though there seemed to be a clear winner, extra and more mathematical proof was required for absolute certainty, which lead to two more comparisons of the cascading models. The specifics and results of these comparisons can be read in section 4.2.2.

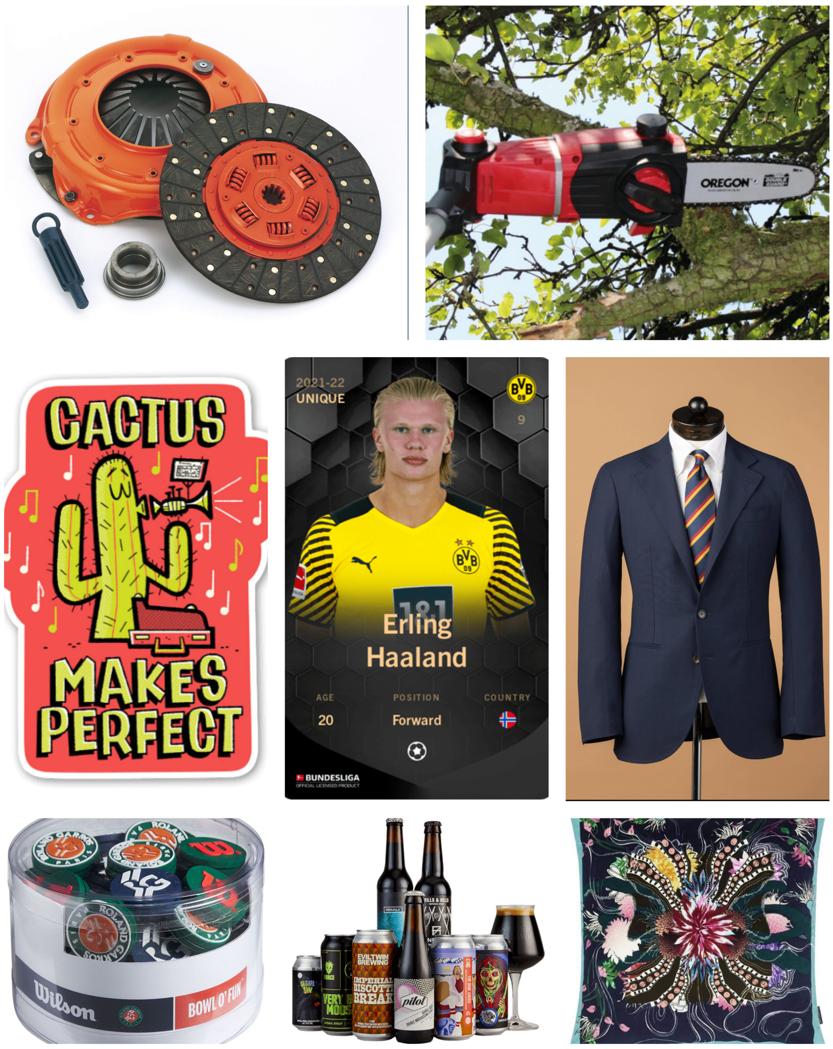

FIGURE 4.1: An example image of each category. From left to right and top to bottom: Car parts, Random objects, Stickers, Football cards, Clothing, Tennis, Beers, Interior.

4.2.1 Comparing on football logo's

To find the best of the three options mentioned, the models were first tested on some random images, including football logo's, to see if based on just perception a superior model could be picked. In figure 4.2 a comparison between the three cascading models is shown by upscaling the logo of Belgian football club Anderlecht. What can be seen from both zoomed in examples is that the x4x2 model shows most detail. In the top image you can see the crown upscaled by the x4x2 model has more (but not all) gems than the other two models. Besides that the pattern on the bottom of the crown looks much more like the pattern in the original picture. Furthermore it's noteworthy that the x2x2x2 and the x2x4 model show minimal differences between each other. In the bottom image it can be seen that the lines are much cleaner in the x4x2 model. The face is less messy and the logo/image on the chest is much more distinct.

The results for this Anderlecht logo were similar to results found in other images. The differences between the x2x2x2 model and the x2x4 model seem to be minor, and the level of detail seems the best in the x4x2 model. However, to be absolutely certain and have more concrete evidence pointing to one model, it was decided to do some more tests and compare the results using similarity metrics.

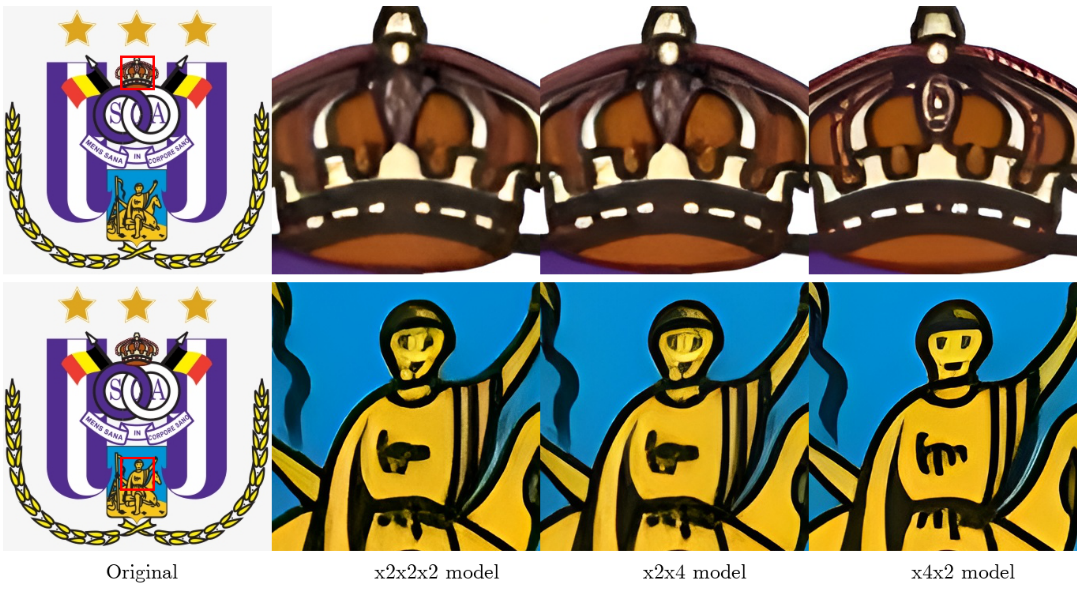

FIGURE 4.2: A comparison of the Anderlecht football club logo on the different cascading models. The x4x2 model is showing the much more detail and strong lines compared to the other two. Zoom in for better view.

4.2.2 Comparing on the Tinify data set

To find more mathematically valid results, it was decided to compare the three cascading models on the Tinify data set. To be able to compare the output images with the ground truth, the first step was to downscale the ground truth images with a factor eight. However, because some of the images had dimensions which were not divisible by eight, some preprocessing had to be done. 39 out of the 100 images were found to have undesired dimensions, and needed preprocessing.

First it was decided to add a bar of black pixels to the images, which would make the dimensions suitable for x8 downscaling and upscaling again. After the experiment was finished the thought came up that a bar of black pixels might interfere with the quality of the images. Therefore the experiment was repeated with the same data set, but now with the 39 images cropped to a desirable amount of pixels instead of extended.

The complete results for both experiments can be found in appendix A, as a reference the resulting graph for the x4x2 model with cropped images is shown in figure 4.3. For all images in the Tinify data set the MS-SSIM score and the PSNR score where calculated and then plotted in the graph. Then the average per image category was calculated and plotted with a slightly bigger dot. By doing this and then drawing lines from the individual images to the category average, it can be easily seen how much variation there is within a certain category. From the graphs a few interesting conclusions can be drawn.

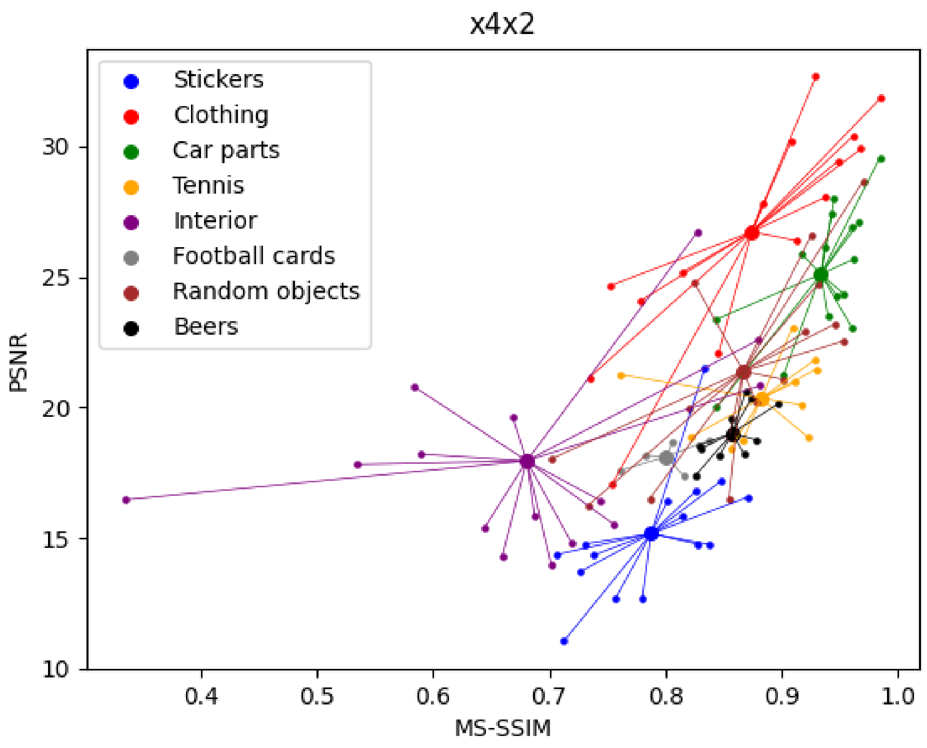

FIGURE 4.3: PSNR and MS-SSIM results on the Tinify data set for the x4x2 cascading model with cropping

The first thing that is interesting is that the general distribution of the image classes stays very similar for all cascading models. This means one model is not specifically better for a certain class of images. This makes sense since all cascading models are trained on the same data sets, meaning it would be curious if one of the cascading models suddenly became more proficient in upscaling a certain image category.
Secondly the differences in scores between the results with cropping and the results with adding the black pixel bar are almost indistinguishable from each other. This confirms that adding a solid bar of pixels to an image does not interfere with the general quality of the upscaled image, which is a plus for the models in general.
Thirdly and most importantly it's still visible that the x4x2 model scores better than the x2x2x2 model and the x2x4 model. On a quick glance there might not seem to be big differences between all the graphs, but when we take a closer look it can be seen that the scales on the axes are slightly shifted, especially for the x4x2 models. This makes the graphs look more similar even though the x4x2 models are scoring significantly higher both on PSNR and on MS-SSIM metrics. These results are a nice confirmation of what was already perceptually established: the x4x2 model yields the best results.

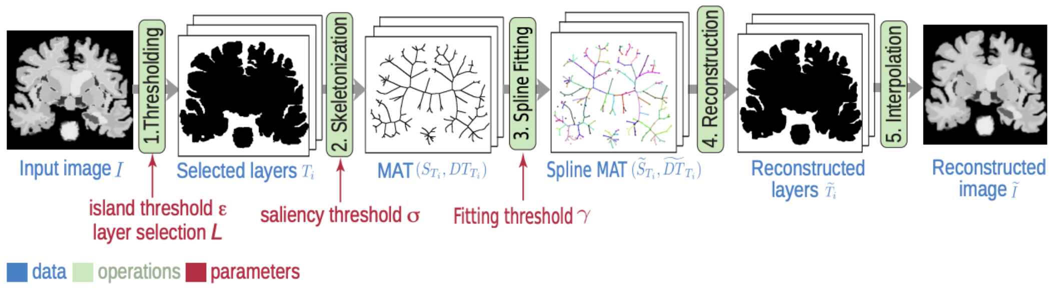

FIGURE 4.4: A visual representation of the 5 SDMD steps

4.2.3 Comparing with SDMD

To make sure the new cascading model can compete with already existing models, we wanted to compare the SSIM and PSNR scores with those of an already existing model. The problem is that none of the models mentioned in chapter 2 can upscale to a factor 8. In search of a model that could handle upscaling factors this big the Spline-Based Dense Medial Descriptor (SDMD) [26] was found. This is a model mainly built for performing image compression, but also able to upscale or downscale images. SDMD can be explained by clarifying the 5 steps that are performed in the process. These 5 steps are visually represented in figure 4.4. The first step is thresholding. In this step the input image is processed to various binary layers by performing thresholding on it with different threshold values. Thereafter a number T of these layers gets selected and skeletonized. What this means is that for these layers the objects in the image get a vector representation of the skeleton of that object. A skeleton of T is also called medial axis of T, or MAT. Then each of the branches in the MAT's get represented as B-splines to give us Spline MAT's. These B-splines can be rasterized to provide us with a pixel representation of the MAT. When this rasterization happens the size of the final image can be determined, which means the upscaling of the image can be done here. Using a disc-union method, a layer $\tilde{T}$ can be reconstructed from this pixel representation. Finally an image can be constructed by combining all these constructed layers $\tilde{T}$. The SDMD paper can provide more detailed information on each of these steps.

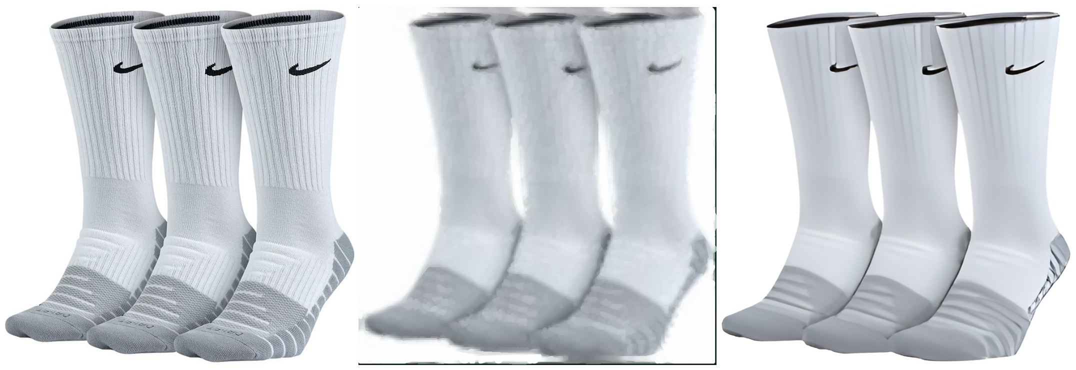

FIGURE 4.5: A comparison on an image of the Tennis category. From left to right: The ground truth, x8 upscaling by SDMD (MS-SSIM **0.683**, PSNR **17.809**) and the x4x2 upscaled image (**0.762**, **21.261**). As can be seen by both perception and the metric scores, though both images aren't good representations of the ground truth (especially visible when zooming in), the x4x2 solution comes a lot closer than SDMD.

The SDMD model was tested on the cropped downscaled Tinify data set. For testing super resolution, all parameters were set in such a way to get the highest image quality. This means for example that parameter T was set to the maximum amount in order to use all the obtained layers after thresholding. This was done to use as much information possible, to get the highest quality of super resolution. Similar measures were taken for other parameters. The resulting graph can be found in figure 4.6 or in appendix A accompanied by the earlier mentioned results.

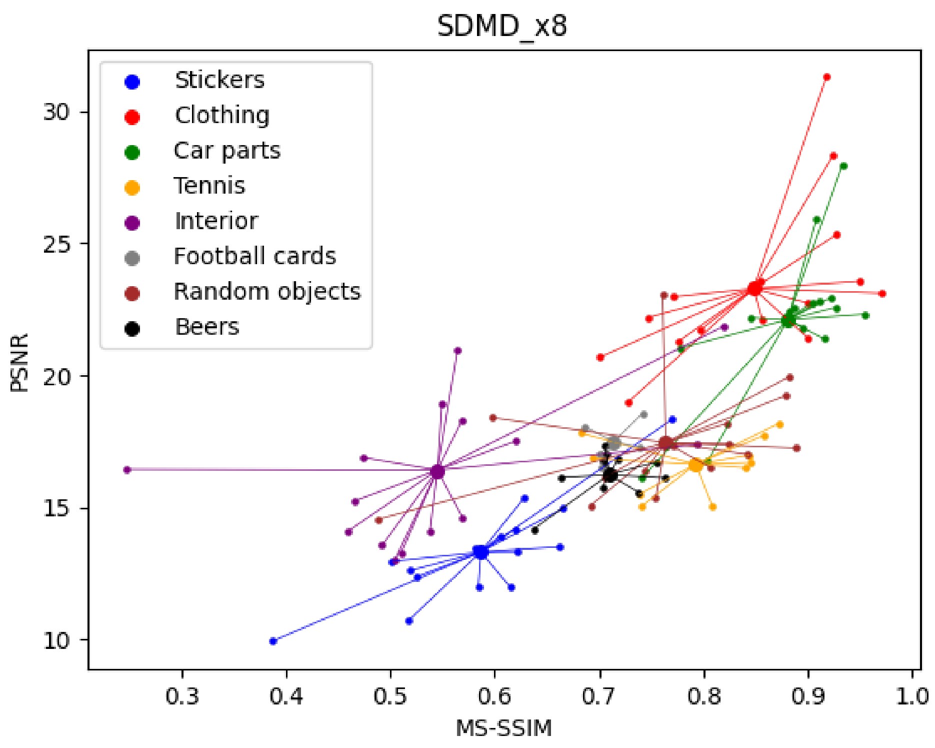

FIGURE 4.6: PSNR and MS-SSIM results on the Tinify data set for the x8 SDMD model with cropping

As expected, since the SDMD model was not created with specifically super resolution in mind, it did not get results as good as the x4x2 cascading model. An example is shown in figure 4.5. What can be seen here is that the SDMD model yields very blurry results, compared to a more clear image constructed by the x4x2 model. When zooming in the loss of texture is also very clear in the image constructed by the x4x2 model and the creation of artifacts on the right heel. Lastly the SDMD model also had some issues with the edges of the images, which can also be seen in the example image. This will most certainly also affect the PSNR and MS-SSIM scores.

On average the SDMD model scores an MS-SSIM score of **0.730** and a PSNR score of **18.042** where the cascading model scores an MS-SSIM score of **0.836** and a PSNR score of **20.794**. Interestingly enough the distribution of the image classes for the cascading model is similar to the one for the SDMD model, which means both models perform better on some specific classes and worse on some others. Why both models struggle with the same categories can be explained in a few possible ways.

- The number of details in that image category is much higher than in others.

- The images in that category are smaller in size, which is why the models have less to work with when they are upscaling
- The image category has images of certain objects that are just hard to upscale

The categories with the worst scores are Stickers and Interior. In table 4.1 it can be seen that the Stickers category is indeed one of the categories with the smallest images, which could correspond to it having bad results. However, the Interior category has images of exactly the same size as the Tennis category but the MS-SSIM scores differ a lot. This shows that image size is not the only factor influencing super resolution quality. The Interior images have much more detail, which partially already gets lost during downscaling, explaining the lower scores. An example of an image from the Interior category upscaled by the SDMD model and the x4x2 model can be seen in figure 4.7. Here it can clearly be seen a lot of the detail from the left pillow gets lost, especially in the image created by SDMD which just gets very blurry. Also again the issue with the image edges is visible. With both the number of details and the size of the images having such an obvious impact on the output image detail, it's hard to determine if the third factor, certain objects being hard to upscale, is also a determining factor. However the big variation within image categories gives an indication that this probably isn't a factor.

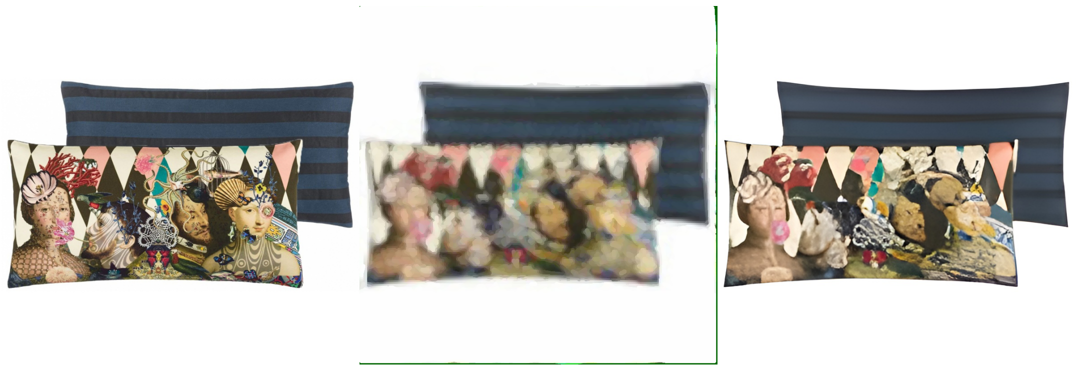

FIGURE 4.7: A comparison on an image of the Interior category. From left to right: The ground truth, x8 upscaling by SDMD (MS-SSIM **0.794**, PSNR **17.372**) and the x4x2 upscaled image (**0.882**, **20.828**). When zooming in it can be seen both models struggle with the amount of detail in the ground truth image, most probably due to the small size the ground truth image has when downscaled.

4.3 Continuous upscaling factors

In this section the results of the experiments on the continuous upscaling factor as explained in section 3.4 will be shown. As mentioned before, these experiments were conducted using the Tinify data set. All images in the data set were first cropped in such a way that both the height and the width of the image would be divisible by the upscaling factor that was tested. This way it was made sure no rounding errors would make it impossible to check the PSNR and MS-SSIM metrics, as these metrics need the images to be exactly the same size. After cropping the images they were downscaled using the desired factor with the Tinify API, after which the images were upscaled using the models and interpolation techniques as explained in section 3.4.

4.3.1 Upscaling factors between 1 and 2

As mentioned in section 3.4 the first small choice that had to be made was what would be done with upscaling factors in the interval $[1,2]$. The two choices are:

- Use the original image and the x2 upscaled image and blend the two after using Lanczos4 interpolation. Recall as mentioned in section 3.4 Lanczos4 interpolation is done with a unique factor f_n for both of the images which can be calculated as follows:

  $$f_n = f_o / f_u$$

  Where in this case f_o would be the original upscaling factor within the interval $[1,2]$ and f_u would be 1 for I_s and 2 for I_l. The blending of the images I_s and I_l would be done using:

  $$I' = \alpha * I_s + \beta * I_l$$

  Where α and β remain as mentioned in section 3.4:

  $$\alpha = 1 - (f_o - f_{us})/(f_{ul} - f_{us})$$
  $$\beta = 1 - \alpha$$

- Only use the x2 upscaled image and use Lanczos4 interpolation to downscale to the preferred size. Recall from section 3.4 this solely entails generating an image I_l from the x2 deep learning model, calculating a factor f_n for the x2 model and using Lanczos4 interpolation with that factor f_n to find the output image.

To test both these methods they were performed on the Tinify data set with an upscaling factor of 1.5. Results of these tests can be found in figure 4.8 and figure 4.9. Significant differences can be spotted in these figures and these differences become even more clear in the average scores. The method with interpolation scores an average MS-SSIM score of **0.984** and a PSNR score of **31.102** compared to **0.975** and **28.573** for the method without interpolation. With these scores and the graphs in mind it seems clear that interpolating with the original image is the way to go. However, to actually see what the difference between the images is like, the image with the highest variance in metric scores between the two methods is shown in figure 4.10. When you zoom in on these images the difference in texture is clear to see. In the ground truth image the texture of the fabric is clearly visible. On the upscaled image that's interpolated with the ground truth image, it can be seen that some of the texture turned more smooth, which is why it doesn't score perfectly on the metrics. On the other hand the text is still looking sharp, as well as other little details. On the rightmost image, without interpolation, it can be seen that large parts of the shirt lost their texture and became completely smooth. Still the details like the text and the buttons look good, but the fabric lost most of its texture. This is well represented in the MS-SSIM and PSNR scores and confirms that interpolation with the ground truth image is the correct choice.

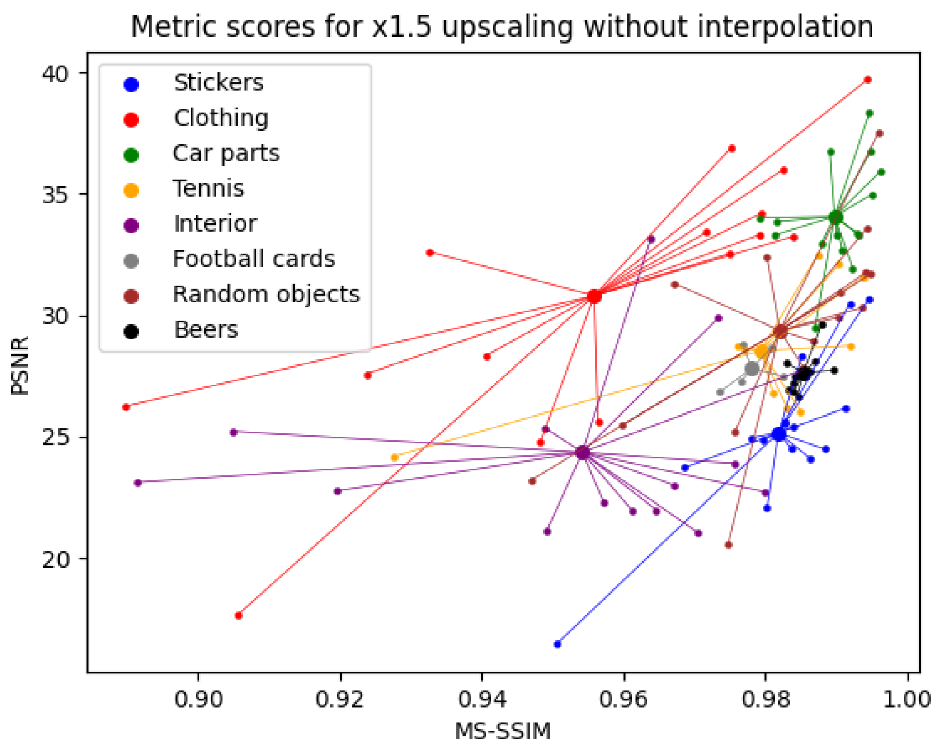

FIGURE 4.8: PSNR and MS-SSIM results on the Tinify data set for an upscaling factor of x1.5 without interpolating with the original image

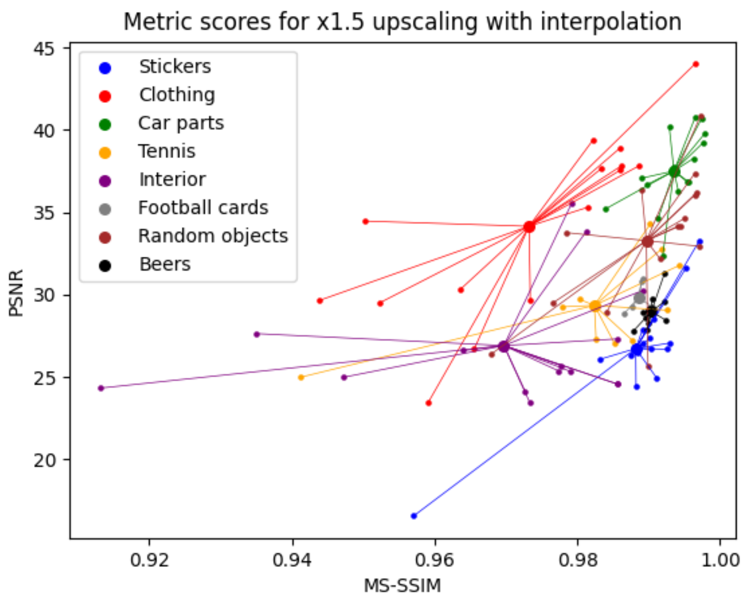

FIGURE 4.9: PSNR and MS-SSIM results on the Tinify data set for an upscaling factor of x1.5 interpolated with the original image

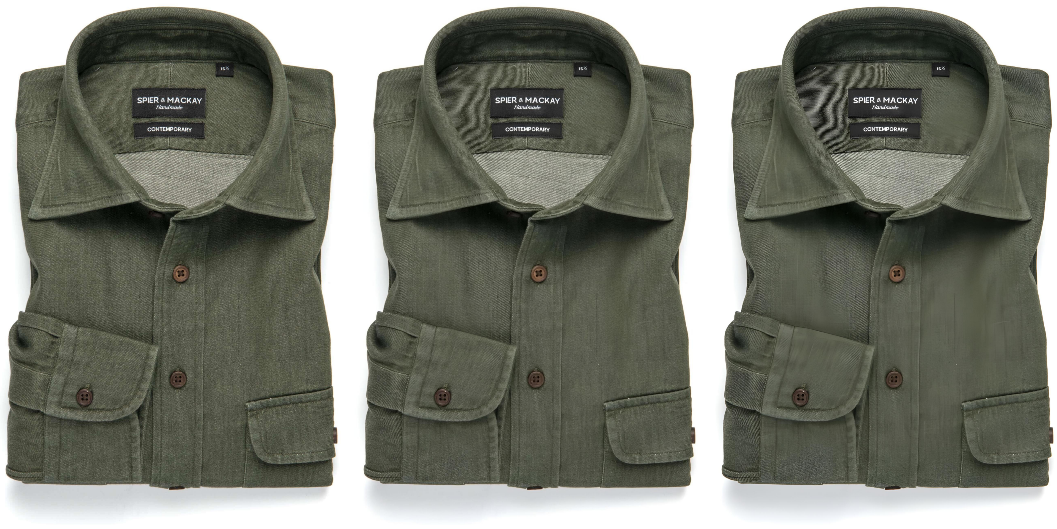

FIGURE 4.10: A comparison between (from left to right) the ground truth, x1.5 upscaling with interpolation (MS-SSIM **0.944**, PSNR **29.650**), x1.5 upscaling without interpolation (**0.890**, **26.240**). The interpolated (middle) images shows a lot more texture than the the image without interpolation (right). Zoom in for better view.

4.3.2 Upscaling factors between 2 and 8

Even though the interpolation technique has been proven to be the best for the image quality for upscaling factors between 1 and 2, one might argue this is slightly biased since it uses the ground truth image. To test if the interpolation technique also works for upscaling factors where we don't interpolate with the ground truth image, a small experiment was conducted for images with an upscaling factor of 3. This upscaling factor was chosen for two reasons. Firstly this factor is exactly between 2 and 4, which means it uses exactly 50% of both deep learning models, so it tests the most pure form of image blending. Secondly in section 1.3.1 it was mentioned the quality for images with an upscaling factor up to 4 is more important than images with a higher upscaling factor. With this in mind it seemed logical to perform an extra experiment on images with an upscaling factor lower than 4, to perform an extra quality check on these images.

The experiment was conducted as follows: all 100 images of the Tinify data set were cropped if needed and downscaled with a factor 3. These downscaled images were then upscaled in three different ways:

- The x4 deep learning model was used to upscale the images with a factor 4. Then the images got downscaled to get to factor 3 by using Lanczos4 interpolation.
- The x2 deep learning model was used to upscale the images with a factor 2. Then the images got upscaled further to get to factor 3 by using Lanczos4 interpolation.
- The interpolation technique as explained in section 3.4: both deep learning models get used, the images get up or downscaled to factor 3 using Lanczos4 interpolation and the two resulting images get blended linearly (so using both images 50%) to get the output image.

Model used	Only x4	Only x2	Both x4 & x2
PSNR	25.699	25.932	26.569
MS-SSIM	0.948	0.944	0.953

TABLE 4.2: PSNR and MS-SSIM results on the Tinify data set for upscaling with factor 3 using only the x4 model, only the x2 model or both models. It can be concluded that using both models is slightly better, but not with a significant difference.

In figure 4.11 an example comparison image can be seen. When comparing the different approaches to factor 3 upscaling in this image it's not easy to spot the differences. However, when zooming in especially the contrast between the image that only used the x2 model and the others is significant. This image (the third from the left) shows the least detail in the text, where differences are easiest to be spotted. This can be explained by the fact that the x2 model wasn't made with a dedicated architecture, but is just the architecture of the x4 model with a pixel-unshuffle layer at the start as explained in section 3.1 and visualised in figure 3.1. There are some differences that can be spotted between the image that only used the x4 model (second image from the left) and the interpolated image (fourth image from the left). One is that the text, especially the 'R' from Ford, looks a little better in the image that only used the x4 model compared to the interpolated image. On the other hand the tail of the horse has a slightly stronger outline in the interpolated image compared to the image that only used the x4 model. However, the differences between these two images are minor, which is also reflected in the small differences between the PSNR and MS-SSIM scores for both images. In table 4.2 the average metric scores for the full Tinify data set are shown. It can be seen that using the image blending technique for an upscaling factor 3 yields slightly better results than using either the x4 or x2 deep learning model individually. Even though the differences between the PSNR and the MS-SSIM scores for the different methods are not very big, from this experiment it can be concluded that using the blending technique yields either equal or better results than using just one model.

Now that a choice is made for the use of image blending not only for upscaling factors between 1 and 2, but for the whole range of upscaling factors, the whole

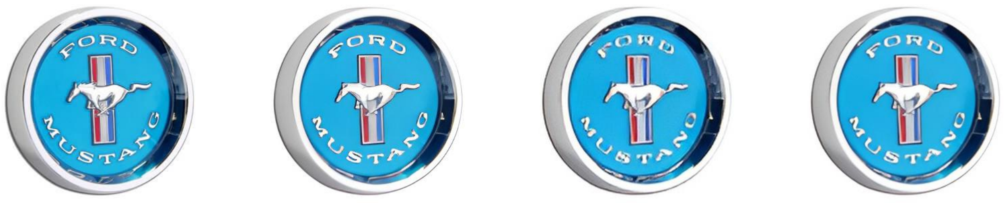

FIGURE 4.11: An example of a comparison between (from left to right) the ground truth, x3 upscaling using only the x4 model (MS-SSIM **0.970**, PSNR **26.479**), x3 upscaling using only the x2 model (**0.959**, **25.449**), x3 upscaling blending both models (**0.971**, **26.886**). The solution only using the x2 model (third image from the left) shows the least amount of detail, especially visible in the 'Ford' text. The other two options do look different from each other, but arguably look equally close to the ground truth image. Zoom in for better view.

Factor	x1.5	x2	x2.5	x3	x3.5	x4	x4.5
PSNR	31.102	27.748	27.179	26.569	25.391	24.385	23.799
MS-SSIM	0.984	0.967	0.960	0.953	0.940	0.925	0.912

Factor	x5	x5.5	x6	x6.5	x7	x7.5	x8
PSNR	23.322	22.729	22.365	21.952	21.492	21.218	20.794
MS-SSIM	0.901	0.887	0.877	0.864	0.854	0.842	0.836

TABLE 4.3: An overview of the PSNR and MS-SSIM scores on the Tinify data set with continuous upscaling factors

range can be tested. Important to notice is that upscaling can be done like explained in section 3.4 for any floating point value. Some examples of an image upscaled with random factors can be seen in figure 4.12. For testing however, it was decided to use each upscaling factor between 1 and 8 with steps of 0.5. This decision was made for two reasons. First of all to keep the downscaling and cropping of the ground truth images relatively easy, as for testing you need to be able to downscale and then upscale again without getting rounding errors. For this reason, as explained before, the images need to be cropped first to have both dimensions divisible by the desired factor. Cropping to a number divisible by 0.5 is much easier than cropping to a number divisible by a random floating point value. Secondly to keep the amount of upscaling that needed to be done to a reasonable amount. The average PSNR and MS-SSIM scores can be found in table 4.3. In figure 4.13 a boxplot is displayed for each of these upscaling factors showing the corresponding MS-SSIM score. In figure 4.14 a similar figure is displayed showing the PSNR scores. What can be seen from these plots is that the image quality gradually declines as the upscaling factor grows bigger, as we would expect. This decline can be explained in two ways:

- The higher the upscaling factor is, the more pixels have to be generated. The more pixels that have to be generated, the more chance that the upscaled image diverts from the ground truth.
- For testing we have to downscale the images first. The smaller an image is, the less detail it has. So if we have to downscale with a factor 8 there's a lot less detail that the model can use to recover the original image.

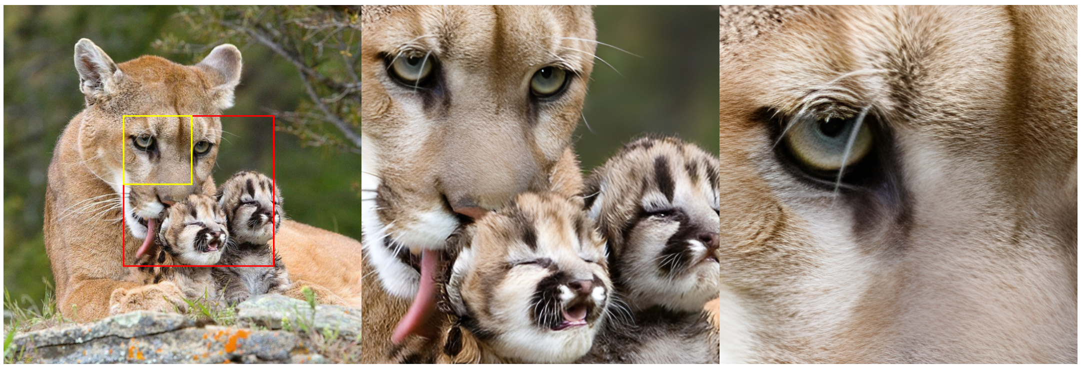

FIGURE 4.12: An example of various floating point upscaling factors. From left to right: the ground truth, a x2.374 upscaled patch, a x5.196 patch. Image taken from the OutdoorSceneTraining data test set [31]

.

This second statement can be checked for our data set, since we have images with varying sizes. If the decline for a large image is less steep than the decline for a small image, it can be concluded that the second statement has at least some validity. That would mean the model performs worse for images that are too small, which is the case for a part of the test set, and will probably perform better when it's in production when the images provided should be large enough. To check this statement a red and a blue line were added to the plots, where the red line represents the largest image in the data set and the blue line the smallest. Coincidentally both images were from the Car Parts category. For the MS-SSIM score, the blue line shows a steeper decline than the red line, which was expected and confirms the second statement. For the PSNR score the angle of the lines is more similar, which could either mean that the second statement is not valid, or that PSNR is a less reliable metric when it comes to checking this.

Even though the decline in image quality was expected as the upscaling factor increased, the almost linear decline is noteworthy. It was expected that the quality of the output images would be significantly better for the ones that didn't need interpolation, so for upscaling factors of 2 and 4. However, in the results in table 4.3 and figures 4.13 and 4.14 no such significant spike in quality is visible.

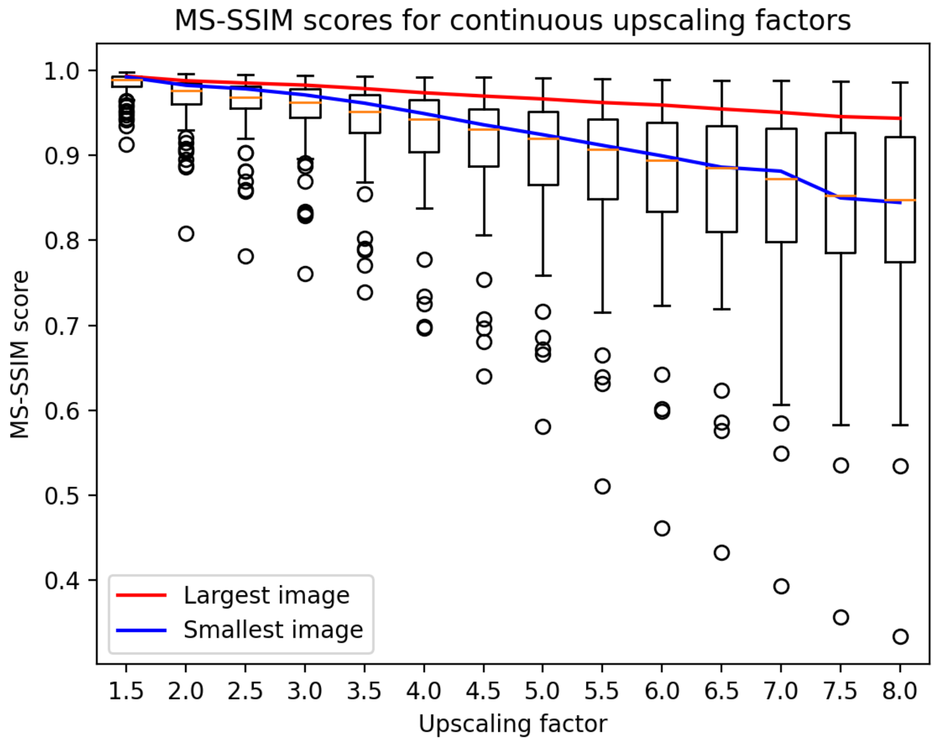

FIGURE 4.13: MS-SSIM results on the Tinify data set for upscaling factors 1.5 till 8.0. The images with the largest and smallest amount of pixels are represented with the red and blue lines.

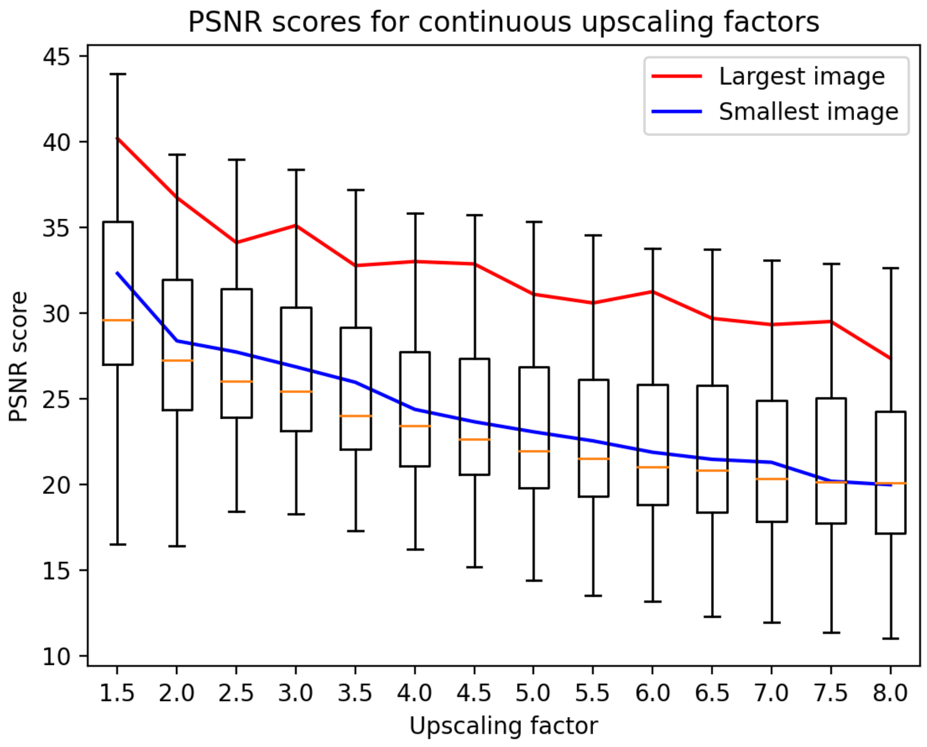

FIGURE 4.14: PSNR results on the Tinify data set for upscaling factors 1.5 till 8.0. The images with the largest and smallest amount of pixels are represented with the red and blue lines.

Chapter 5

Discussion and Conclusion

In the previous chapter the results were shown and explained. In this chapter these results will be discussed in section 5.1 according to the original requirements as defined in section 1.3. After this discussion a conclusion can be drawn regarding the final super resolution model. Finally the possible improvements and future research opportunities for this model will be discussed in section 5.2.

5.1 Performance Analysis

In this section the performance of the final model will be analyzed according to the functional and non-functional requirements as defined in section 1.3. After analyzing and discussing these requirements, the research question as defined in section 1.2 can be answered and conclusions can be drawn.

5.1.1 Functional Requirements

Here the functional requirements as defined in section 1.3.1 will one by one be revisited and discussed considering the final model. The functional requirements were defined as requirements which the algorithm needs to fulfill to cater to the needs of the customers of Tinify.

Domain - As already mentioned in section 2.1.6, the Real-ESRGAN model was trained on various data sets which should make the model versatile enough to cater to the needs of the clients of Tinify. When we look at figure 4.3 it can be seen that not all domains score equally but as explained in section 4.2.3 it is more likely that these differences can be explained by the size of the images and the amount of detail in the images. With these explanations in mind we can assume that it's almost certain the model is good enough for the different domains of images that Tinify clients use.

Quality - The next functional requirement as defined in section 1.3.1 was quality. As mentioned there is a lot of variation in terms of quality metric scores when we look at the Tinify data set. This can mostly be attributed to the size of the images and the amount of detail in the images. The average PSNR and MS-SSIM scores can't be compared to other super resolution models since they were not tested on the same data sets. However it can be stated that for a random group of images (the Tinify data set) the quality metrics certainly look as good or even better than we would expect, when we compare them to average scores of other models on other data sets. The only valid comparison was between the cascaded model and the SDMD model, where as expected the cascaded model scored better. From the images in figure 4.12

it can be seen that upscaling, even for larger factors works really well on real life images and even creates textured details such as hair and eyes almost perfectly.

Upscaling factor - In section 1.3.1 the ideal solution for the upscaling factor was mentioned as having a floating point value between 1 and 8 with up to 2 decimals precision. The current solution offers even more decimals precision as it can upscale for any floating point value between 1 and 8. In the functional requirements it was also mentioned the main focus in terms of quality control would be up to factor 4. After constructing the x8 cascading model it was expected that the quality would decline faster for upscaling factors higher than 4, as the cascading model shouldn't perform as well as a dedicated model trained for a certain upscaling factor. However, when looking at figure 4.13 and figure 4.14 the decline in quality looks rather linear, there is no bigger drop in quality for upscaling factors higher than 4 which is a very positive result. When looking at figure 4.13 there is an increase in variance for upscaling factors higher than 4, which also might have to do with the difference of image sizes as mentioned in section 4.3.2.

Speed - In section 1.3.1 it is mentioned the upscaling process should be fast, preferably no longer than a few seconds for an output image of roughly 2 megapixel. As this was an optional requirement no time has been spent on making the algorithm fast. There also is a lot of difference in speed depending on whether interpolation is being used or not. As mentioned in section 4.1 a Google Cloud Virtual Machine with 1 NVIDIA Tesla A100 GPU was used. Upscaling to an image of roughly 2 megapixel takes around 12 seconds for a factor 2 or 4, but if we want an upscaling factor other than 2 or 4 it will take more than twice of that number: around 30 seconds. This is due to the fact of Lanczos4 interpolation being a quite slow method and for interpolation it has to be used twice. Keep in mind that for an output picture of roughly 2 megapixel and an upscaling factor just over 4, the image also gets upscaled with factor 8 meaning an image of roughly 4 megapixel is also being generated which makes the whole process a lot slower. Since the focus for this thesis was more on quality than speed, choices have been made to do it this way. However it's needless to say different interpolation methods besides Lanczos4 could be used in practice to possibly speed up the process.

Image extension handling - The model works for all image extensions mentioned in section 1.3.1, which are `.jpeg`, `.png` and `.webp`.

Proof of quality - As already mentioned in section 2.1.6 the Real-ESRGAN paper showed both positive and negative proof of quality of the model. In the end this was deemed enough to be picked as preferred framework for this project. Although no official experiments have been performed to see if the quality shown in the paper is equal to the quality we see ourselves in the model from Github, the visuals show enough evidence that this is the case. As for this paper, the proof of quality was shown both by publishing quality metrics and by showing example output images.

5.1.2 Non-functional Requirements

In this section the non-functional requirements will be discussed in a similar way as the previous section did for the functional requirements. The non-functional requirements were defined in section 1.3.2 as those requirements that describe how a system performs while doing the work described by the functional requirements.

Programming interface - In section 1.3.2 it was mentioned that the preferable outcome of this project would be that the user can simply add the preferred upscaling factor as a parameter in the Tinify API and receive the upscaled image. Due to time constraints the final model hasn't been implemented in the Tinify API yet. However, the current Python API of the existing model already works in a way that is similar to the simple way the Tinify API works. To upscale an image all that has to be done is run the the program from the command line, with parameters for the input image, upscaling factor and the desired destination of the output image. The Python program will then perform all necessary steps automatically and output the image in the desired destination.

Digital infrastructure - The requirement for digital infrastructure was defined as follows: The model should be able to run on the Google Cloud Platform. As the current model is running in the Google Cloud Platform already without any problems, this will likely not cause any problems in the future as well.

Cost of ownership - In the non functional requirements in chapter 1 it was mentioned that the model should be easy to maintain and be lightweight, so it doesn't cost much to have it running non-stop. For the current solution we need to have two trained models in the cloud environment: the x2 and x4 model as obtained from the Real-ESRGAN Github page. As mentioned in section 2.1.6 these models are 33,4 MB apiece, meaning they slightly go above the limit of 50 MB set in section 1.3.2. However, as mentioned in that section, this limit was not set in stone and shouldn't pose any problems. Other than that the cost of ownership is hard to measure since the solution isn't live in the Tinify API yet.

Scalability - The scalability requirement was described as having a model that is stateless, which would make horizontal scaling possible. As the model is written in Pytorch it's possible to obtain trained instances of the x2 and x4 model, which means the scalability requirement is fulfilled.

Replicability - The replicability requirement was mainly set for the model we were going to use as framework. As described in section 2.1.6 and throughout the rest of the thesis, the chosen Real-ESRGAN model was easy to replicate, mainly due to the availability of the full code repository on Github. As for our own solution, due to the commercial nature of the end product the exact code repository and image test set are not meant to be publicly available. However, with the detailed descriptions in this thesis the experiments could be repeated by other employees of Stapes IT who do have access to the code and the image database.

5.1.3 Conclusion

Now that all results have been shown and discussed, the research question can be answered. The research question was defined in section 1.2 as follows:

"What is an efficient and effective deep learning approach for generating super resolution images that can be deployed in a practical online solution for nonspecialist end users?"

In section 2.1 different deep learning models were compared in terms of efficiency and effectiveness for the prospected end users of Tinify and Real-ESRGAN was found to perform best. By adding the x8 upscaling factor and finding a way to be able to upscale for any floating point value between 1 and 8 the effectiveness of the model was then highly improved. However, this improvement came at the cost of efficiency, as the model also turned significantly slower after improvement. Nonetheless the speed was deemed fast enough for the end users. The comparison with the SDMD model in section 4.2.3 also showed the model is more than competitive in terms of quality for the high upscaling factor of 8. From these findings we can conclude this model is both efficient and effective enough for the end users of Tinify and although it's out of scope for this thesis, from the prototype it is clear that the solution can be implemented in a practical way for nonspecialist end users.

5.2 Future Work

Even though the found solution is deemed sufficient for the end users of Tinify, there is still a lot of improvement possible which will be discussed next.

5.2.1 Improving on Deep Learning

One of the questions that still stands after this thesis is whether adding more dedicated deep learning models to the solution would improve the quality of the output images. At the start of this project it was planned to either have a trained model capable of handling all upscaling factors or to have a model for all integer values between 1 and 8 and interpolate between those. In the end due to time constraints and the complexity of the framework it was decided to only use a x2 and a x4 model (and a combination of those) and interpolate between them. The interesting conclusion was that the output image quality decreased rather linearly as the upscaling factor increased, even though we would've expected the quality to be much lower for the interpolated images. It would be interesting to see if adding more deep learning models dedicated to other upscaling factors would improve the quality of the images, or if they would actually be superfluous.

In addition to this it would also be interesting to see if a dedicated architecture for the x2 model would improve that model much. As explained in section 3.1 and visualised in figure 3.1 the architecture for the x2 model is the exact same architecture as the x4 model, with a pixel-unshuffle layer at the start to get to the right dimensions. In figure 4.13 it can't really be seen that the x2 model performs worse than what should be expected, however in figure 4.14 there is a big drop between the x1.5 and x2.0 model. This raises the question if a more dedicated architecture for the x2 model would improve the quality of that model.

5.2.2 Interpolation Techniques

For this thesis it was chosen to use Lanczos4 interpolation as image interpolation technique to get two images to the same size to be able to blend them. This technique was chosen as it was proven as one of the better image interpolation techniques and as it was easy to implement. There are newer and arguably better interpolation techniques however, such as AKNR [35], which might perform better.

Besides the fact that other interpolation techniques could be used, the current procedure of interpolating between two images isn't proven to be the most optimal one. By first interpolating the images to the correct size and then blending them, a lot of new information in the smaller image has to be created and a lot of information of the larger image gets thrown away. It might be better to find a way to use the information from the larger image to create the extra information in the smaller image, or find a way to blend a larger and a smaller image without interpolating them to the desired size first.

5.2.3 Continuous training

Finally there's the improvements on both the quality and the image domains as mentioned in section 2.3.2. These improvements are specifically important for the needs of Tinify and its clients, not necessarily for improving the solution for general use. One of the proposed improvements is unsupervised continuous training. As already mentioned this is technique is probably too risky and experimental for implementing it in a live product.

The other option closely related to this that also was mentioned in section 2.3.2 was estimating the quality per domain offline. This could be done by using a technique such as LAMP [11] to automatically categorize the images and then use either a quality metric or human perception to see which category could be improved. Then the original deep learning models could be trained more using images from this category to get better results.

Appendix A

Cascading model comparisons

A.1 Results with black pixel bar

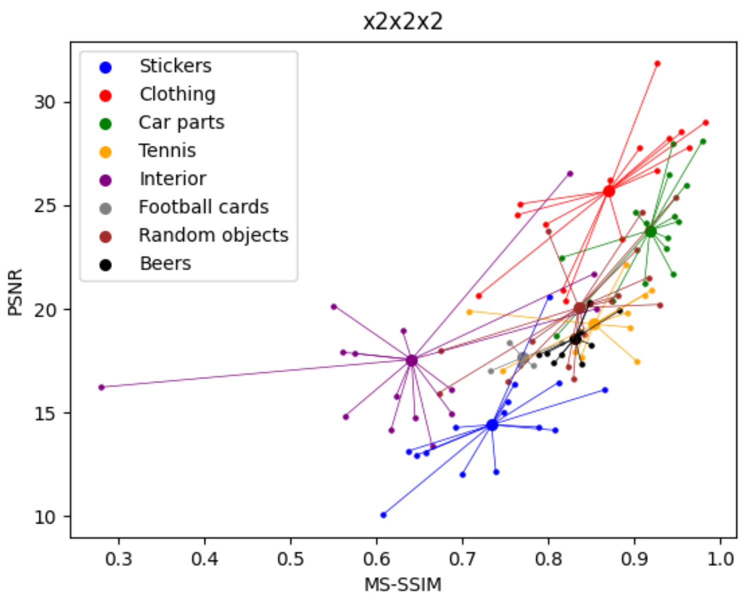

FIGURE A.1: PSNR and MS-SSIM results on the Tinify data set for the x2x2x2 cascading model with black pixels added

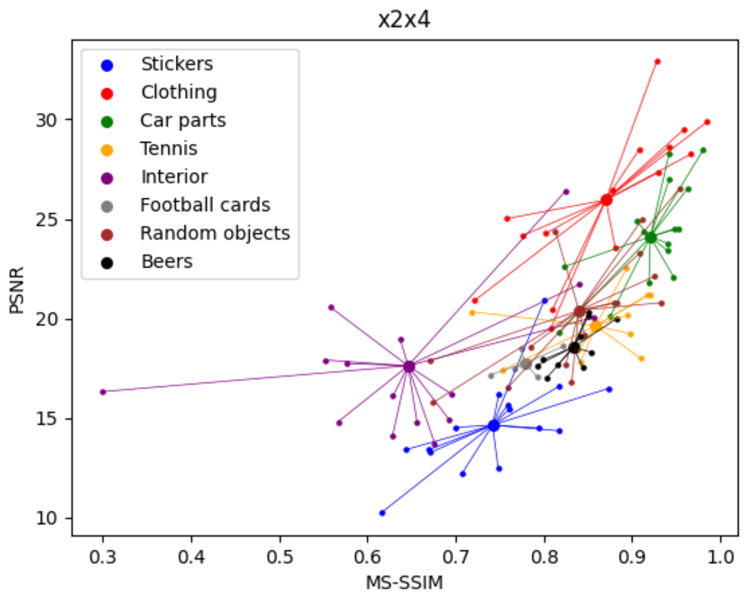

FIGURE A.2: PSNR and MS-SSIM results on the Tinify data set for the x2x4 cascading model with black pixels added

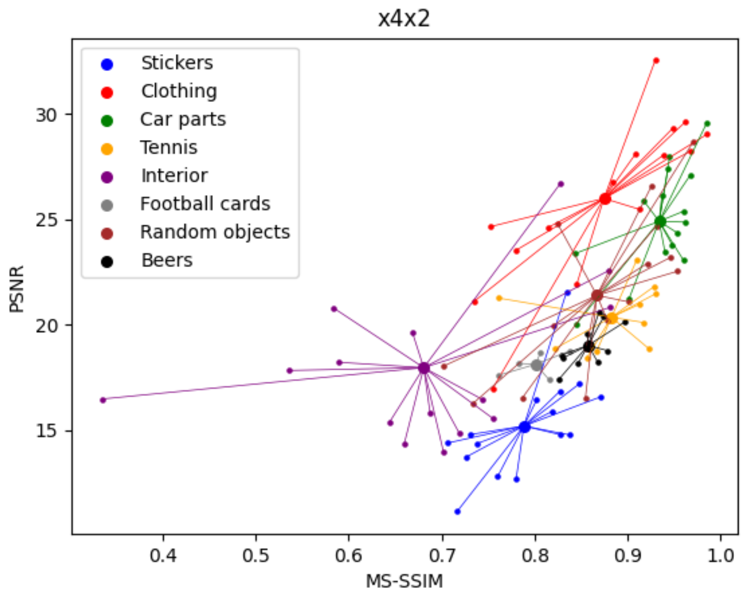

FIGURE A.3: PSNR and MS-SSIM results on the Tinify data set for the x4x2 cascading model with black pixels added

A.2 Results with cropping

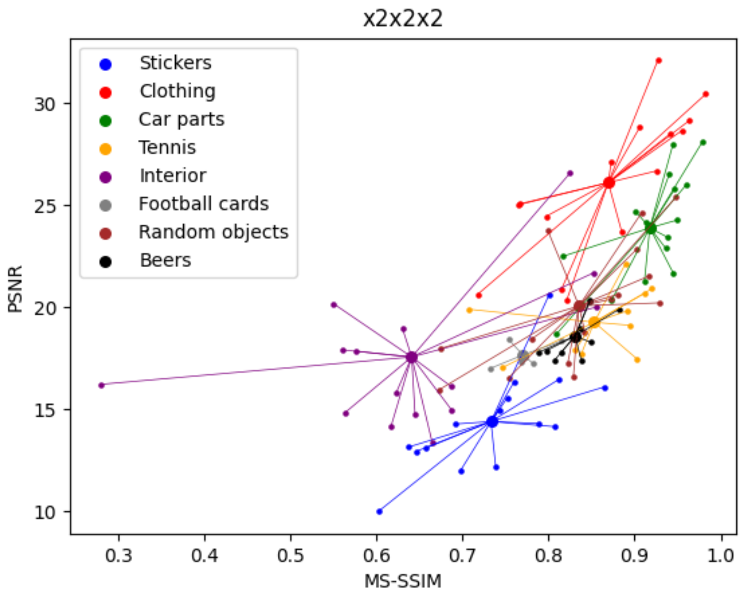

FIGURE A.4: PSNR and MS-SSIM results on the Tinify data set for the x2x2x2 cascading model with cropping

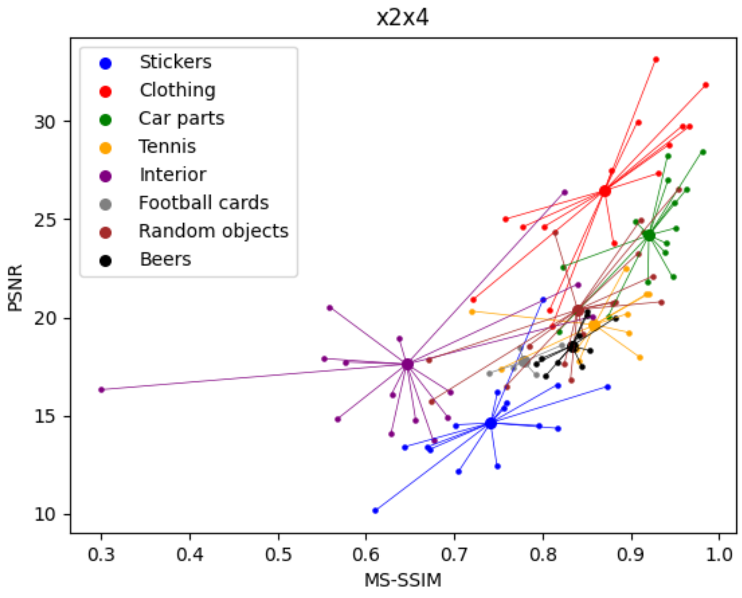

FIGURE A.5: PSNR and MS-SSIM results on the Tinify data set for the x2x4 cascading model with cropping

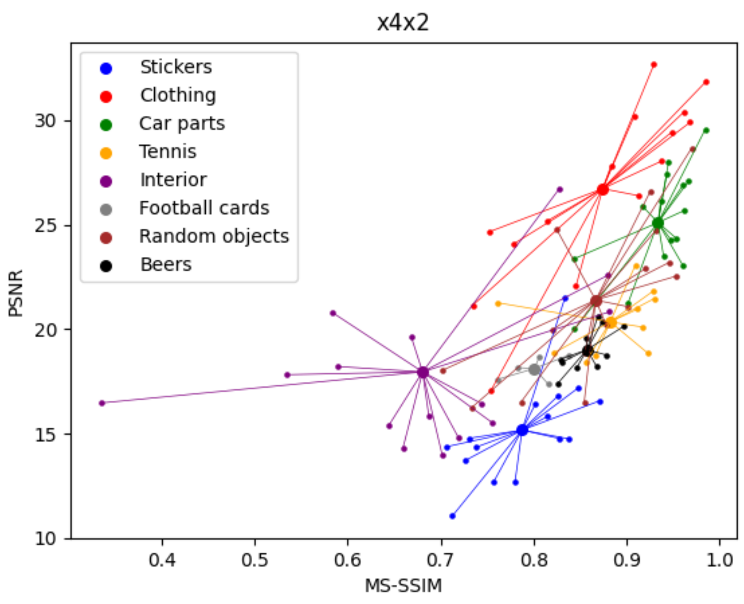

FIGURE A.6: PSNR and MS-SSIM results on the Tinify data set for the x4x2 cascading model with cropping

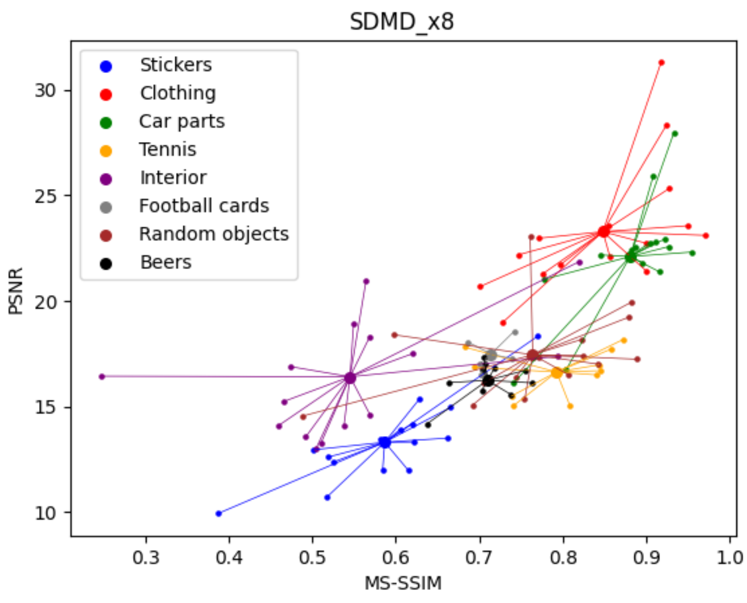

FIGURE A.7: PSNR and MS-SSIM results on the Tinify data set for the x8 SDMD model with cropping

Appendix B

Planning

Phase two of this thesis project has 23 weeks left, starting from the 13th of December till the 22nd of May. A schematic overview of the planning for these weeks can be found in figure B.1. A more detailed explanation of the tasks will follow.

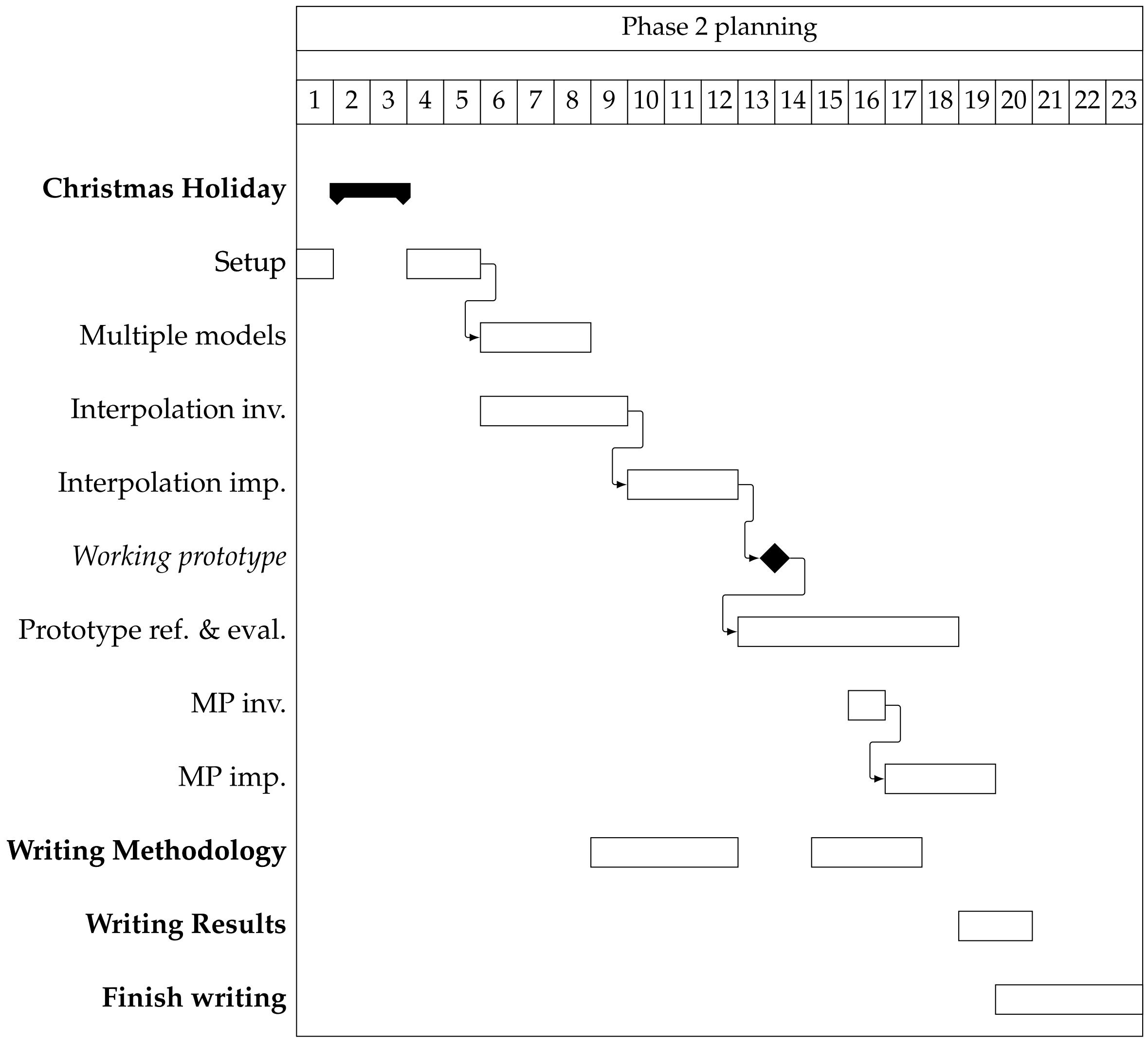

FIGURE B.1: A Gantt-chart overview of the planning for phase 2.

1. **Setup (1, 4-5)**: The project is fully setup once the following is done: Real-ESRGAN is implemented in a separate cloud environment (not in the Tinify production environment). We are able to upscale an image with a factor of 4 (the standard for Real-ESRGAN) using this model from the cloud from either an API (preferred) or a user interface of some sort. Note that this task overlaps with the Christmas holidays, which is why it will take 5 weeks instead of 3.

2. **Multiple models (6-8)**: Train and test all the models we want to have running in the final product (upscaling factor 1 till 8). This task is done once we have the multiple models running in the cloud environment and are able to choose which one to use from the API or GUI which was constructed in the setup phase.

3. **Interpolation investigation (6-9)**: The first part of the interpolation task. This task is done once enough investigation has been done on how to interpolate between two super resolution images. This investigation can be done by finding several interpolation techniques and testing them on the models.

4. **Interpolation implementation (10-12)**: The implementation of the chosen interpolation algorithm. This task is done when the interpolation algorithm is successfully implemented in the environment created in the setup. Users should now be able to upscale their image to any desired floating point value between 1 and 8, thus we have a working prototype.

5. **Prototype refinement & evaluation (13-18)**: This mainly entails refining the prototype and evaluating if the results are as good as promised. Besides that implementing the prototype in the Tinify environment will also be part of this. This might not directly be in the large image manipulation API[1], since this is a very complicated environment and we are unable to estimate whether it's possible to implement this within the time frame of the thesis. If it's not possible to implement the model in the API, the model will be available for users on a separate location.

6. **MP investigation (16)**: Investigating the multidimensional projection technique. This task is done once a fitting technique is found to project the images on a 2D plane in which various image domains are represented. Besides this it also has to be investigated whether LPIPS performs as well as needed or other metrics have to be used.

7. **MP implementation (17-19)**: Implementing the multidimensional projection technique. This task is done once a program is written that can automatically place images on the 2D plane mentioned in *MP inv*. Furthermore these images also need to be automatically rated based on LPIPS or another metric that has been chosen.

8. **Writing Methodology (9-12), (15-17)**: This task is divided in two parts. In the first part the basic structure of the methodology chapter should be created and the sections about image interpolation should be written. In the second part the sections about the multidimensional projection techniques will be written.

9. **Writing Results (19-20)**: Writing the results chapter.

10. **Finish writing (20-23)**: Writing all remaining chapters: Discussion & Conclusion, Future work and finishing any unfinished chapters.

[1] https://tinypng.com/developers/reference

Bibliography

[1] Eirikur Agustsson and Radu Timofte. "Ntire 2017 challenge on single image super-resolution: Dataset and study". In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2017, pp. 126–135.

[2] Marco Bevilacqua et al. "Low-complexity single-image super-resolution based on nonnegative neighbor embedding". In: (2012).

[3] Yochai Blau et al. *The 2018 PIRM Challenge on Perceptual Image Super-resolution*. 2019. arXiv: 1809.07517 [cs.CV].

[4] Chao Dong et al. "Learning a deep convolutional network for image super-resolution". In: *European conference on computer vision*. Springer. 2014, pp. 184–199.

[5] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems* 27 (2014).

[6] Sam Gross and Michael Wilber. "Training and investigating residual nets". In: *Facebook AI Research* 6 (2016), p. 3.

[7] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[8] Jia-Bin Huang, Abhishek Singh, and Narendra Ahuja. "Single image super-resolution from transformed self-exemplars". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 5197–5206.

[9] Andrey Ignatov et al. "Dslr-quality photos on mobile devices with deep convolutional networks". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 3277–3285.

[10] Xiaozhong Ji et al. "Real-world super-resolution via kernel estimation and noise injection". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 2020, pp. 466–467.

[11] Paulo Joia et al. "Local affine multidimensional projection". In: *IEEE Transactions on Visualization and Computer Graphics* 17.12 (2011), pp. 2563–2571.

[12] Christian Ledig et al. "Photo-realistic single image super-resolution using a generative adversarial network". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4681–4690.

[13] Bee Lim et al. "Enhanced deep residual networks for single image super-resolution". In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2017, pp. 136–144.

[14] Andreas Lugmayr, Martin Danelljan, and Radu Timofte. "Ntire 2020 challenge on real-world image super-resolution: Methods and results". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 2020, pp. 494–495.

[15] Chao Ma et al. "Learning a no-reference quality metric for single-image super-resolution". In: *Computer Vision and Image Understanding* 158 (2017), pp. 1–16.

[16] Anish Mittal, Rajiv Soundararajan, and Alan C Bovik. “Making a “completely blind” image quality analyzer”. In: *IEEE Signal processing letters* 20.3 (2012), pp. 209–212.

[17] Takeru Miyato et al. “Spectral normalization for generative adversarial networks”. In: *arXiv preprint arXiv:1802.05957* (2018).

[18] Pankaj S Parsania and Paresh V Virparia. “A comparative analysis of image interpolation algorithms”. In: *International Journal of Advanced Research in Computer and Communication Engineering* 5.1 (2016), pp. 29–34.

[19] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[20] Edgar Schonfeld, Bernt Schiele, and Anna Khoreva. “A u-net based discriminator for generative adversarial networks”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 8207–8216.

[21] Wenzhe Shi et al. “Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 1874–1883.

[22] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).

[23] Christian Szegedy et al. “Inception-v4, inception-resnet and the impact of residual connections on learning”. In: *Thirty-first AAAI conference on artificial intelligence*. 2017.

[24] Radu Timofte, Vincent De Smet, and Luc Van Gool. “Anchored neighborhood regression for fast example-based super-resolution”. In: *Proceedings of the IEEE international conference on computer vision*. 2013, pp. 1920–1927.

[25] Radu Timofte et al. “Ntire 2017 challenge on single image super-resolution: Methods and results”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2017, pp. 114–125.

[26] Jieying Wang, Jiri Kosinka, and Alexandru Telea. “Spline-Based Dense Medial Descriptors for Lossy Image Compression”. In: *Journal of Imaging* 7.8 (2021), p. 153.

[27] Xintao Wang et al. *BasicSR: Open Source Image and Video Restoration Toolbox*. https://github.com/xinntao/BasicSR. 2018.

[28] Xintao Wang et al. “Esrgan: Enhanced super-resolution generative adversarial networks”. In: *Proceedings of the European conference on computer vision (ECCV) workshops*. 2018.

[29] Xintao Wang et al. “Real-esrgan: Training real-world blind super-resolution with pure synthetic data”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 1905–1914.

[30] Zhihao Wang, Jian Chen, and Steven CH Hoi. “Deep learning for image super-resolution: A survey”. In: *IEEE transactions on pattern analysis and machine intelligence* (2020).

[31] Chao Dong Xintao Wang Ke Yu and Chen Change Loy. “Recovering Realistic Texture in Image Super-resolution by Deep Spatial Feature Transform”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.

[32] Jianchao Yang et al. “Image super-resolution via sparse representation”. In: *IEEE transactions on image processing* 19.11 (2010), pp. 2861–2873.

[33] Roman Zeyde, Michael Elad, and Matan Protter. “On single image scale-up using sparse-representations”. In: *International conference on curves and surfaces*. Springer. 2010, pp. 711–730.

[34] Richard Zhang et al. “The unreasonable effectiveness of deep features as a perceptual metric”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 586–595.

[35] Jieying Zheng et al. “Image interpolation with adaptive k-nearest neighbours search and random non-linear regression”. In: *IET Image Processing* 14.8 (2020), pp. 1539–1548.

[36] Jun-Yan Zhu et al. “Unpaired image-to-image translation using cycle-consistent adversarial networks”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2223–2232.