SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA

Faculty of Informatics and Information Technologies

FIIT-5212-102879

**Samuel Bubán**

# Image processing using methods of artificial intelligence

Bachelor thesis

May 2022

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA

Faculty of Informatics and Information Technologies

FIIT-5212-102879

**Samuel Bubán**

# Image processing using methods of artificial intelligence

Bachelor thesis

Study programme: Informatics

Study field: Computer Science

Training workplace: Institute of Computer Engineering and Applied Informatics

supervisor: Ing. Marek Jakab, PhD.

May 2022

Slovak University of Technology in Bratislava
Ústav počítačového inžinierstva a aplikovanej informatiky

Faculty of Informatics and Information Technologies
Academic year: 2021/2022
Reg. No.: FIIT-5212-102879

# BACHELOR THESIS TOPIC

| | |
|---|---|
| Student: | **Samuel Bubán** |
| Student's ID: | 102879 |
| Study programme: | Informatics |
| Study field: | Computer Science |
| Thesis supervisor: | Ing. Marek Jakab, PhD. |
| Head of department: | Ing. Katarína Jelemenská, PhD. |

Topic: **Spracovanie obrazu metódami umelej inteligencie**

Language of thesis: English

Specification of Assignment:

Rozpoznávanie, opis a zmena vizuálnych vlastností obrazu je v súčasnosti intenzívne skúmaná oblasť počítačového videnia a hlbokého učenia s významným uplatnením v robotike, obohatenej realite, medicíne či priemyselných aplikáciách. Napriek svojmu potenciálu a doterajšiemu výskumu má hlboké učenie stále mnoho problémov v oblasti spracovania obrazu pre potreby úprav, transformácií a vylepšovania obrazových dát, ktoré ho vzďaľujú od nasadenia v praxi. Analyzujte aktuálne metódy spracovania obrazu a zlepšovania kvality obrazu s využitím konvolučných neurónových sietí. Zamerajte na najnovšie aktuálne metódy v danej oblasti publikované za posledné roky. Experimentuje s vybranými existujúcimi riešeniami zameranými na túto problematiku. Navrhnite a implementujte vlastnú metódu s využitím dostupných nástrojov. Vytvorte softvérový prototyp, overte a porovnajte navrhnuté riešenie s existujúcimi riešeniami na dostupných verejných datasetoch.

Length of thesis: 40

| | |
|---|---|
| Deadline for submission of Bachelor thesis: | 16. 05. 2022 |
| Approval of assignment of Bachelor thesis: | 23. 11. 2021 |
| Assignment of Bachelor thesis approved by: | doc. Ing. Valentino Vranić, PhD. – Study programme supervisor |

# Čestné prehlásenie

Čestne vyhlasujem, že som túto prácu vypracoval samostatne, na základe konzultácii a s použitím uvedenej literatúry.

Samuel Bubán

**V Bratislave, 15.5.2022**

# Poďakovanie

Chcel by som sa poďakovať môjmu vedúcemu práce Ing. Marekovi Jakabovi, PhD. za jeho ochotu a konzultácie, či už osobne alebo cez slack, dokonca aj počas víkendov. A taktiež za jeho rady a odbornú pomoc, ďakujem.

Ďalej by som sa chcel poďakovať Ing. Lukášovi Hudecovi, PhD. za jeho pomoc a ochotu konzultovať problémy.

Moje poďakovanie patrí aj mojej rodine a kolegom, za ich podporu a pomoc.

# Annotation

Slovak University of Technology Bratislava
FACULTY OF INFORMATICS AND INFORMATION TECHNOLOGIES
Degree Course: Informatics

Author: Samuel Bubán
Bachelor's Thesis: Image processing using methods of artificial intelligence
Supervisor: Ing. Marek Jakab, PhD.
May 2022

The goal of image super-resolution, as a method of image processing using artificial intelligence, is to enhance an existing image and make it a higher resolution without any loss in the level of detail. The applications of image super-resolution are various: real photo enhancing, video upscaling, higher video game resolution, and many others.

This paper aims to explore existing image super-resolution methods and extract core model features from them based on their results. Then suggest a new approach combining the explored methods and compare it to the existing ones. The suggested model takes an image and the upscaling factor and produces the resulting image that is upsampled by the upscaling factor, allowing for variable super-resolution, while preserving the level of detail.

The effect of different datasets on the results is also researched, like using a larger varying dataset versus a smaller single-area focused one.

# Anotácia

Slovenská technická univerzita v Bratislave
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ
Študijný program: Inofrmatika

Autor: Samuel Bubán
Spracovanie obrazu metódami umelej inteligencie
Vedúci bakalárskej projektu: Ing. Marek Jakab, PhD.
Máj 2022

Cieľom super rozlíšenia obrazu, ako spôsobu spracovania obrazu pomocou umelej inteligencie, je zvýšiť kvalitu existujúceho obrazu, zvýšiť jeho rozlíšenie, ale zároveň zachovať úroveň detailov. Aplikácie zvyšovania rozlíšenia obrazu sú rôzne. Vylepšovanie existujúcich fotografií, zvyšovanie rozlíšenia videa, vyššie rozlíšenie v počítačových hrách a mnoho ďalších.

Cieľom tohto článku je preskúmať existujúce postupy na super rozlíšenie obrazu a extrahovať z nich základné modelové vlastnosti na základe ich výsledkov. Následne je navrhnutý nový prístup, ktorý kombinuje preskúmané metódy, a výsledky tohto nového modelu sú porovnané s existujúcimi modelmi. Navrhovaný model dostane na vstupe obrázok, a okrem neho pracuje aj so zväčšovacím faktorom. Rozlíšenie obrázoku zväčší o tento faktor, čo umožňuje nastaviteľné super rozlíšenie, pričom zachová pôvodnú úroveň detailov.

Skúma sa aj vplyv rôznych vstupných dát na výsledky, ako je použitie väčšieho rozličného datasetu v porovnaní s menším súborom, zameraným na jednu konkrétnu oblasť.

# Contents

# Image super-resolution

Image super-resolution (SR), also called image upsampling or image upscaling, is a method used to create a high-resolution image (HR) from a low-resolution input image (LR). This is a seemingly difficult task to generate new data and combine them with the LR image to produce a higher quality result.

Multiple methods have been developed to address this issue. Some of them require training a model over loads of input data, others were made without it, not utilizing neural networks.

## 1.1 Interpolation approach

There are numerous ways to upsample an image. The most basic approach is to duplicate all pixels to the next one on the side and then duplicate all rows under each other. This approach is called the nearest-neighbor algorithm and is commonly used. The result is blocky and jagged with sharp edges. It does not produce any new information, only duplicates the existing.

Another approach that also alters the data is bilinear interpolation upsampling. This method creates new pixels based on its four closest neighbors and approximates a new pixel by computing a weighted average of these values. The result is smoother than that of the nearest neighbor.

Bicubic interpolation works similarly to bilinear interpolation but over a larger area of 4x4 nearest cells. The weighted average is again calculated, so the outer pixels affect the result less than the closer ones. [27]

These interpolation examples are shown in Figure 1.1.

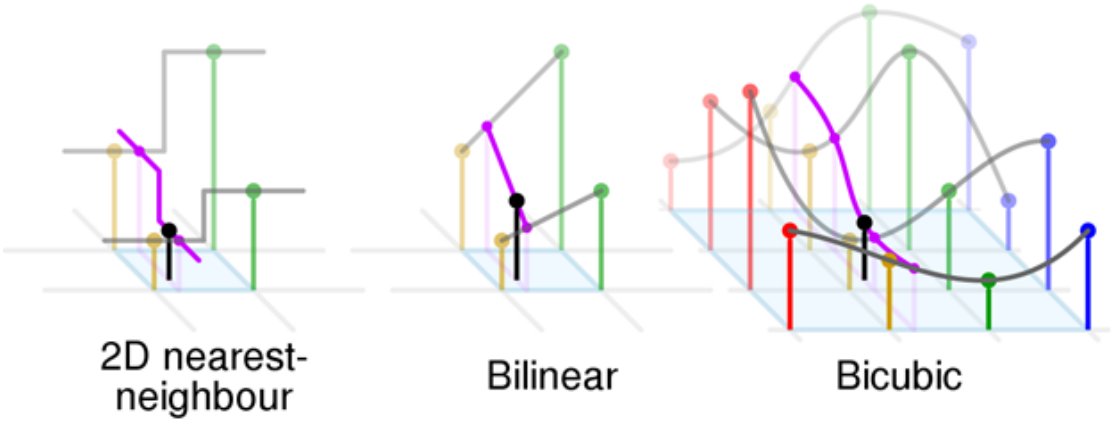


Figure 1.1: Upscaling algorithms showing the method how to calculate a new pixel value. 2D nearest neighbor takes the closest value, bilinear and bicubic calculate a weighted average of 4 and 16 neighbors respectively.[1]

## 1.2 Super resolution in games

Super-resolution is a sought-after technology in the gaming industry. This is because it is much easier for the GPU to render the scene at a lower resolution and then display a high-resolution image based on an algorithm instead of rendering the scene at full scale.

Therefore, a solution that can upscale the scene with seemingly no loss of detail is the target in the fastest possible time. Depending on the settings, the quality may decrease slightly while the frame rate increases, as seen in Figure 1.2.

AMD FidelityFX Super Resolution (FSR) uses upscaling on each image by taking the anti-aliased frame without relying on other data such as frame history or motion vectors. It works in two main passes: an upscaling pass, where the input

[1] https://en.wikipedia.org/wiki/Bicubic_interpolation#/media/File:Comparison_of_1D_and_2D_interpolation.svg

frame is analyzed, and the algorithm detects how neighboring gradients differ from a set of input pixels, and a second pass that extracts pixel details in the upscaled image. [2]


Figure 1.2: The difference between AMD FSR modes. The lower the original image, the higher frame-rate can be achieved.[2]

### 1.2.1 Training model

In recent years, a new possible approach has been tested. This method trains a neural network on a large dataset of images to learn how to predict new images from LR to HR. This was made possible by the new GPUs that contain tensor cores, which were explicitly designed to be utilized by neural networks.

#### 1.2.1.1 NVIDIA Deep Learning Super Sampling

The first version of NVIDIA Deep Learning Super Sampling (DLSS) was based on single-image upsampling. That is precisely the goal of this paper, but instead of

[2]https://www.tomshardware.com/reference/amd-fsr-fidelityfx-super-resolution-explained

upscaling real-time computer games, the model will only be used for single images or image sequences.

This method utilizes neural networks (NN), so the model must first be trained on a specific game to make it compatible. This is the reason why not many games support DLSS.

Because it is based on single-image upscaling, the method does not provide perfect results for video or video games. Many artifacts are present on smaller objects, like meshes, and blurred frames were also reported. This is the same problem as the previous game upscaling algorithm FSR is experiencing. It does not take other factors into the final image calculation other than the input image.

#### 1.2.1.2 NVIDIA Deep Learning Super Sampling 2.0

Because of the issues mentioned above, NVIDIA worked on a second iteration of the model, where DLSS not only calculates the HR image from the input LR render but also takes into account other factors, like object vectors (what direction the character is moving, where will it go or how fast is it moving). This change significantly improved the model over its first version and removed most of the artifact problems, displayed at Figure 1.3.

However, this improvement cannot be applied to single image super-resolution because the images do not have any information regarding these vectors or history.

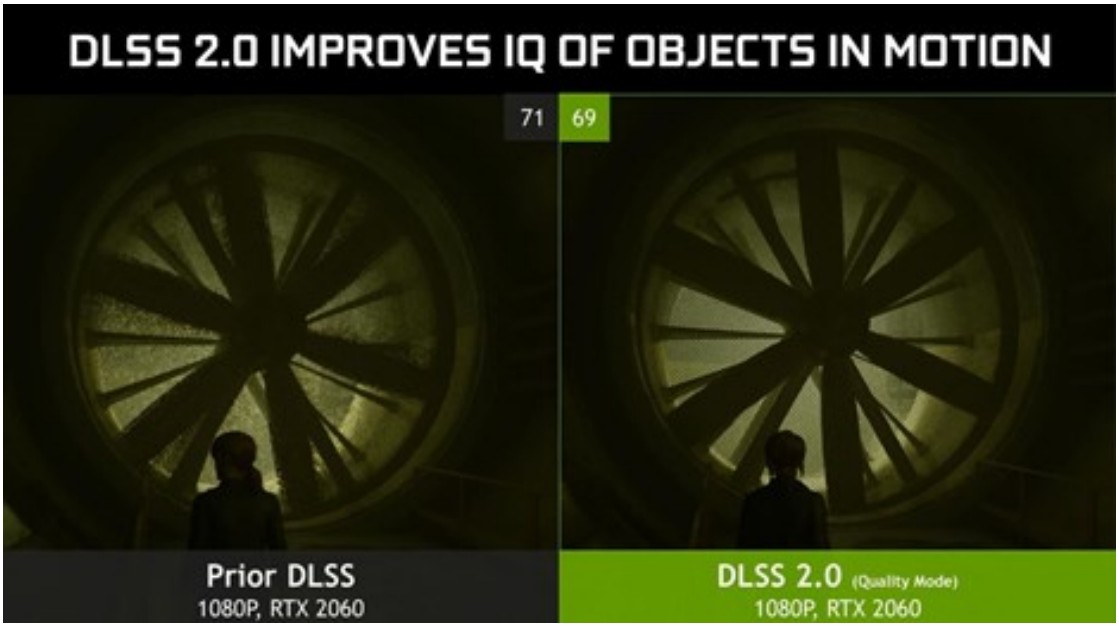


Figure 1.3: Comparison between the DLSS and DLSS 2.0. The first iteration suffers from blurry mesh with artifacts, but this is completely removed in the later version.[3]

[3]https://www.pcworld.com/article/398924/nvidias-faster-better-dlss-20-could-be-a-game-changer.html

# Neural networks

Based on biological nervous systems, such as the human brain, neural networks, also called artificial neural networks (ANN), work similarly. Both consist of large amounts of interconnected computational nodes referred to as neurons. Artificial neural networks are designed to process input information and output some result: a label to classify an image, trend predictions, object recognition, or others. Due to the thousands or millions of neurons, a neural network (NN) can learn complex patterns and apply them to never-before-seen data.

However, one NN model cannot be used in multiple areas. Each NN must be specifically designed and trained for the unique task. Neural networks require being properly trained over hundreds or thousands of iterations (epochs). After each epoch, the model is adjusted based on the result it provided, compared to the target desired output, and the process repeats. They are designed to collectively learn from a large input dataset and apply this knowledge to produce an output. [15]

## 2.1 Neural network structure

Feedforward NNs (FNNs) process data in a feedforward manner, from left to right or layer-by-layer. A counterexample would be the human brain, which does not consist of layers. Instead, neurons in the human brain are connected in groups, and each group communicates with one or more others. [22, 23]

The data flow in FNNs is directed, so it cannot loop back but always continues

in the general direction of the network. The first layer receives the data from the input - the data we feed into the NN. A fully connected network with one hidden layer is displayed at Figure 2.1.

The output of one layer goes to the next layer as the input. NNs can be fully connected, or each neuron can feed only to a couple of other neurons in the next layer, as shown in Figure 2.1.

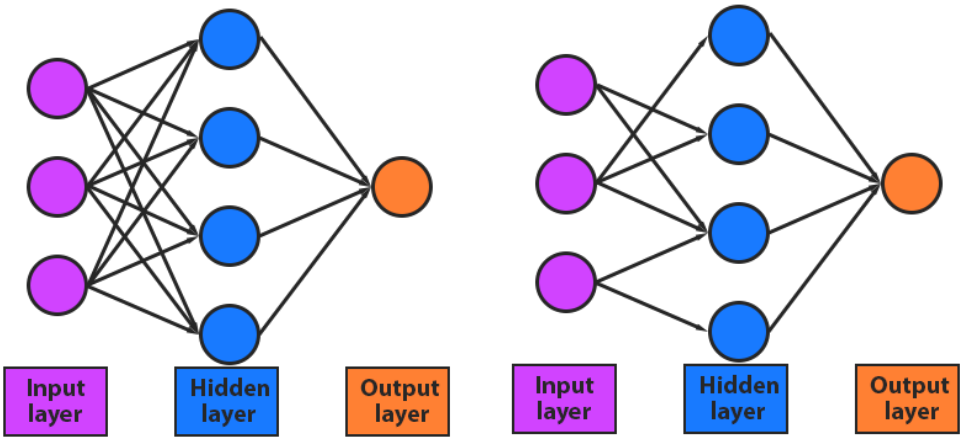


Figure 2.1: Network models. The left model is a fully connected neural network, where each node receives input from all previous nodes and sends it to all nodes in the next layer. The right model shows another example, where each node only connects to some nodes in the next layer.

Once the output is reached, it is evaluated. In supervised learning, which is described in Section 2.2.2, the target output is known. The output of the first pass and the target are compared, and based on the difference, the weights in all nodes are adjusted layer by layer utilizing backpropagation.

One of these passes is called an epoch, and a neural network requires to be trained over several of them. The more complex the model, the longer it takes to train properly. For example, a simple model may only require ten epochs, but a complex deep network will require thousands.

At the end of each epoch, the NN is evaluated against a validation dataset to prevent overfitting, as explained in Section 2.2.4.

When an NN is done training, the weights inside all the nodes are saved and now can be used to predict new, never seen data. This process effectively saves the whole model, which can be later loaded again with the training already done. [19, 24]

### 2.1.1 Perceptron

The first and simplest NN models introduced were perceptrons, also called single-layer neural networks. They consist of one visible layer, the input, and one output layer. The structure of these models consists of: visible input units $v_i$, trainable weights $w_i$, one for each input, bias $b$, and output unit with activation function $f()$, and it is displayed in Figure 2.2. [23, 16]

Getting the final output consists of 4 steps:

1. multiplying all inputs with corresponding weights
2. adding all products together (sum)
3. adding bias
4. calculating output via an activation function

$$y(v) = f(b + \sum_{i=1}^{n} v_i w_i)$$

Figure 2.2: The perceptron equation displayed as an activation function of the sum of all inputs multiplied by their respective weights, with added bias after the sum.

The activation function is chosen based on the task for which an NN is trained. For perceptrons, these simple NNs were used mainly for binary classification, so a Sigmoid activation function displayed at Figure 2.3 was used. It is a normalizing function that maps all real numbers in the range $[0, 1]$. This choice allows for a simple classification: if $y < 0.5$, the image fits into class A; otherwise, class B.

Multiple output nodes can be used for a multi-class problem, all with sigmoid activation functions. Each node represents one class. This addition is how the model is applied to classify more than just two classes: the node with the highest value is chosen as the most probable class type. [23, 16]

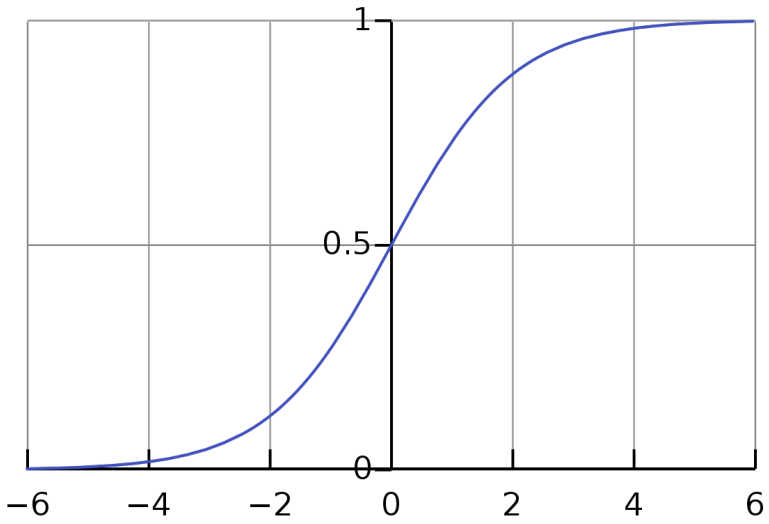


Figure 2.3: Plotted sigmoid function, that is approaching 0 on the negative x, and 1 on the positive x.[1]

### 2.1.2 Deep neural networks

A neural network can consist of any number of layers. The layers that are not input or output are hidden layers. NNs that consist of at least three hidden layers

[1] https://commons.wikimedia.org/wiki/File:Logistic-curve.svg

are generally considered deep neural networks. Deep neural networks allow for better abstraction of data. Each layer can have its activation function, depending on the purpose it is supposed to serve. [23]

When an NN needs to be expanded, it can be widened by increasing the number of neurons in each layer, or it can be deepened by including more layers. The latter solution is regarded as better, as it allows the model to discover good features automatically in a hierarchical manner. However, deeper NNs take longer to train, as backpropagation loses strength with each new layer. An example showing 2-layer network equation is displayed in Figure 2.4, and a diagram showing a deep network is displayed in Figure 2.5. [23, 13]

$$y(v) = f_2(b^2 + \sum_{i=1}^{n} w_i^2 f_1(b_i^1 + \sum_{j=1}^{n} v_j w_j^1))$$

Figure 2.4: 2-layer network equation.

### 2.1.3 Autoencoders

Autoencoders are a type of NNs. They learn to encode input data in a reduced number of nodes, which can be effectively used as compression. The goal is to learn a compact representation of the input while retaining the most critical information. Then the representation is expected to reconstruct the original input completely. [28, 4]

An autoencoder consists of two NNs: one to compress the input image into a smaller 1D array of fewer values and a second that can reconstruct the image from this reduced array. This creates an autoencoder that takes an image as input and outputs its reconstructed state when chained together. This example is visible in

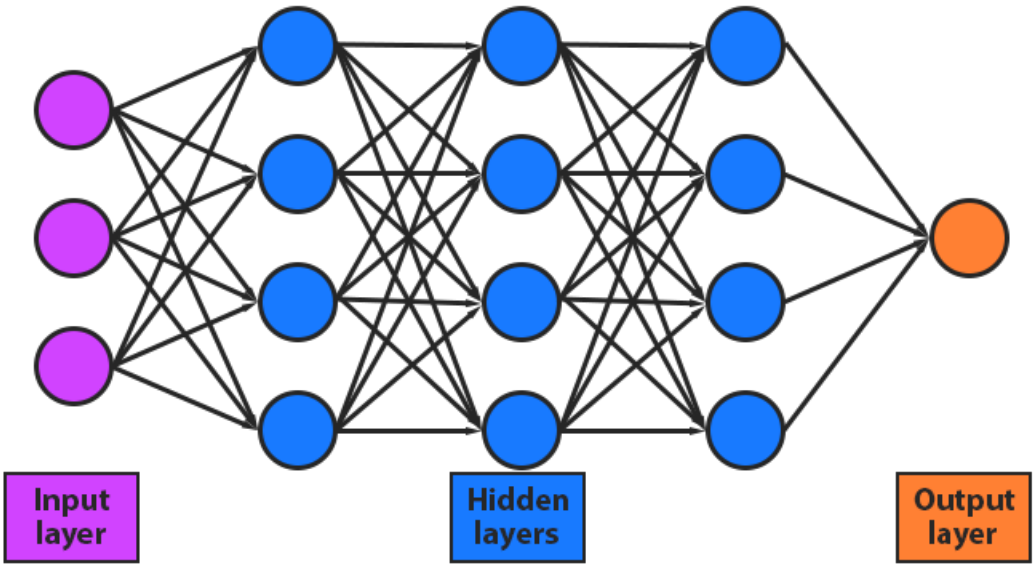


Figure 2.5: A model showing a deep neural network with an input layer, three hidden layers, and one output layer.

Figure 2.6. [28, 13]

As with all NNs, autoencoders are also data-specific, so they can only compress and reconstruct data trained on or similar to them. They are also lossy, so the reconstructed output will be close but not the same as the original input.

### 2.1.4 Convolutional neural networks

When processing a more considerable image input, breaking the 2D pixel structure into a 1D vector will destroy some extra information from neighboring pixels since flattening the 2D space into just one dimension removes a part of it. Until this point, the NNs received their input as a vectorized array, but the layers will have to be different to keep the structure. [23, 15, 7]

This area is where convolutional neural networks (CNN) outperform regular densely connected networks. A CNN can take a 2D information matrix and preserve all the

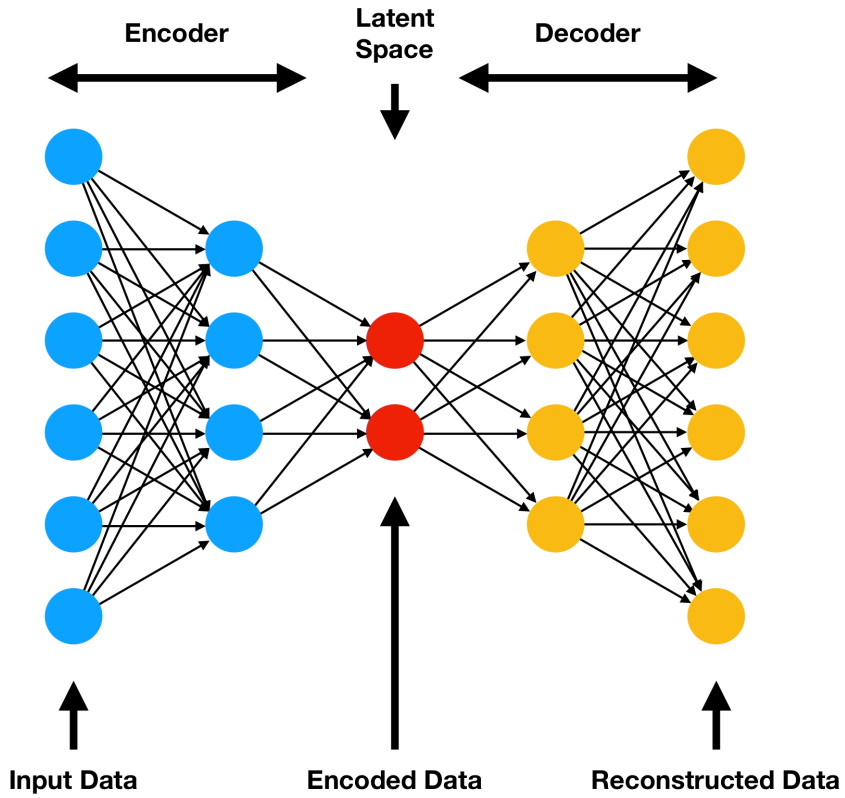


Figure 2.6: Autoencoder network. If used for compression, this network would reduce the input data from size 6 to just 2.[2]

neighboring information. Each convolutional node has a kernel (smaller learnable weighted matrix), which is how it learns.

Each layer has multiple nodes. Each node can learn to abstract a feature, so the total number of features a layer can abstract equals the total number of nodes the layer has. The advantage over other types is that learning a feature means that the network will be able to recognize it anywhere in the picture – the features are not directly connected to the space where they were spotted. [15, 11]

[2]https://www.compthree.com/blog/autoencoder/

#### 2.1.4.1 Pooling layers

Convolutional layers are usually connected to a pooling layer, for example, max-pooling, visible in Figure 2.7. It downsamples the output of a convolutional layer by choosing the highest output and removing the rest. This data reduction makes learning easier for an NN because each pooling layer greatly reduces the resolution. A pooling layer with a pool size of 2x2 will reduce the resolution four times, as displayed in Figure 2.8. This reduction makes it easier to generalize on an image, which also helps prevent overfitting, as described in Section 2.2.4.

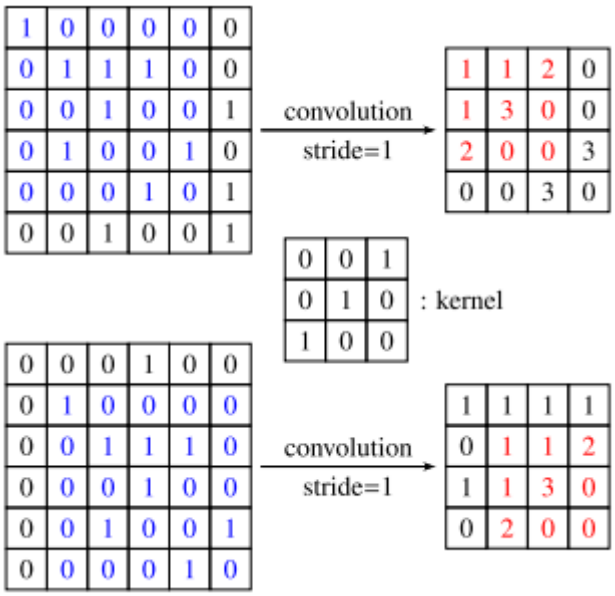


Figure 2.7: The kernel is applied to the input on the left side. It goes from one corner to the other, moving by the stride amount. Multiplication is applied to each cell, and the total sum is calculated. This process produces the new output, which has reduced the size by kernel size -1. [23]

#### 2.1.4.2 Upsampling layers

Pooling layers reduce the output size, but an upsampling layer must be used if we want to get back to the original size or beyond. The upsampling layer generates

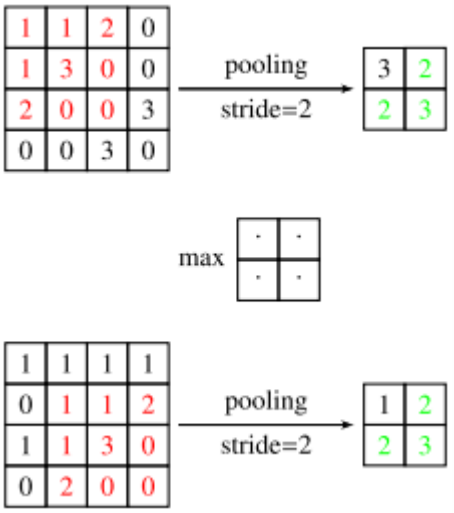


Figure 2.8: The pooling is applied to the input on the left side. Step by step, it picks the largest amount and removes the rest. The stride determines how much it moves with each step. [23]

additional data. For example, if the upscale size is 2x2, it will generate four values from one.

There are multiple algorithms on how to calculate these new values, which were already explained in Section 1.1. In addition, a different upsampling method, sub-pixel convolution is displayed at Figure 3.1.

#### 2.1.4.3 Convolutional autoencoder

It is possible to get better results from image encoding by using convolutional autoencoders rather than dense autoencoders. The reason is that convolutional networks are better suited for image processing than dense networks – they keep neighboring information intact.

A possible use case for convolutional autoencoders is image denoising. An example is shown in Figure 2.9. The autoencoder has to be trained specifically to do this, and the pipeline would be: input noisy image and place the same image with no

noise as the target output. Autoencoder can now learn to map the noisy input into a clear output. [6]

Figure 2.9: Denoising autoencoder. Top row shows the input, and bottom row the output an autoencoder has produced.[3]

## 2.2 Learning

A neural network consists of many nodes with weights. Each weight can be adjusted, which results in a change of behavior – it learns. This task of adjusting node weights can be summed up as error function minimization.

The most commonly used algorithm to update all node weights is backpropagation. This algorithm calculates the loss function gradient for all the weights inside NN. It is then used to update each node's weight from the output to the input. [24, 8]

### 2.2.1 Loss function

A loss function defines how much the output differs from our desired target output. When classification is done, a loss function defines how many outputs the NN got wrong and its distance from the desired output. Each output says how likely the image is to be of one class. When processing images for image upscaling, the loss

[3]https://blog.keras.io/building-autoencoders-in-keras.html

function defines how different our produced image is from the target image.

The goal of a NN is to minimize the loss function. Therefore, backpropagation is calculated based on the loss function.

Mean squared error (MSE) is one of many loss functions used. It also corresponds to an image quality metric, the peak signal-to-noise ratio (PSNR). This metric shows if the predicted image is similar to the target. The higher the PSNR, the better our prediction is. So, by minimizing the MSE loss, we also increase our PSNR. The effect of PSNR on an image is displayed in Figure 2.10. [9]

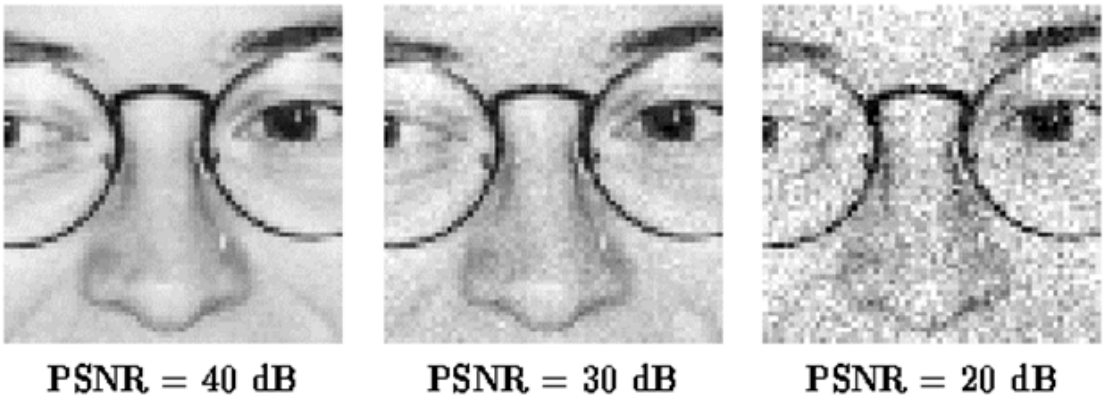


Figure 2.10: Images showing the PSNR value with adding more and more noise. The more noise there is, the lower the PSNR is, and the higher the MSE would be.[4]

MSE is calculated as the average sum of squared errors of each pixel. This value describes how far off the predicted image is from our target. For example, $Y$ is our predicted upscaled image with resolution $[x, y]$ and $\overline{Y}$ is the target image with high resolution, also $[x, y]$. The equation at Figure 2.11 shows how to calculate the MSE value for a single predicted image. [30]

[4]https://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/VELDHUIZEN/node18.html

$$MSE = \frac{1}{xy} \sum_{i=1}^{x} \sum_{j=1}^{y} (Y_{ij} - \overline{Y}_{ij})^2$$

Figure 2.11: MSE Equation as a nested sum of all squared errors.

### 2.2.2 Supervised learning

Supervised learning is the type where the NN knows what the target is. Each image is labeled with the target class if it is image classification. If it is image upscaling, each input image has the target image's second pair. This is because it has a higher resolution, and it is the goal of what the NN is supposed to produce.

The NN's task is to learn the proper mapping function to determine which label or image corresponds to each input. If it learns direct mapping without generalizing, it is a case of overfitting, which is not the desired behavior. [15, 20, 8]

### 2.2.3 Unsupervised learning

The difference between supervised and unsupervised learning is that the NN in unsupervised learning cannot compare its output with the target labels, or target images, because they are omitted.

The NN determines the output validity based on its loss function and adjusts its weights accordingly.

Because unsupervised learning NN does not know what output is supposed to map to which class label, it can better generalize and group the input based on the features it chooses.

The problem is when a NN starts grouping the input based on non-existent features

and creates chaotic classes instead of the desired ones. [20, 8]

If the target is known, it is better to use supervised learning, already explained in Section 2.2.2.

### 2.2.4 Overfitting

Overfitting happens when a NN stops learning general features and instead learns exactly its training input mapping to the output. For example, this issue can happen with large networks training on too small datasets.

This is why just creating a more extensive and deeper network will not solve all problems. Overfitting can also happen in too long trained models. Once the NN hits a point of the best generalization, it will start mapping the input directly to the output target, thus making generalization worse.

There are multiple ways to prevent overfitting: reducing the number of learnable weights in nodes by decreasing the number of neurons, broadening the dataset to include more varied images, which make it easier to generalize over, early stopping when a validation error is detected, or utilizing dropout or residual networks, described more in Section 2.2.8. [24]

An extreme example – if a dataset only contains one image, there is no way for the NN to learn other features, and it will overfit and learn the direct mapping of this image to its target output. Expanding the dataset is always an excellent way to prevent this with no drawbacks other than increased training time. [15]

### 2.2.5 Underfitting

On the other side, there is underfitting. The NN cannot correctly grasp the relationship between the input data and its output part, either the label or the

target image. This results in a high error rate even after hundreds of training epochs.

A way to solve this is to introduce deeper networks or increase the size of layers inside the NN. Both approaches will increase the total number of learnable weights, thus increasing the learning capability of this NN.

### 2.2.6 Vanishing gradient

All the NN structures explained up to this point rely on backpropagation to update their learnable weights. Backpropagation relies on calculating the gradient of loss to decide how to update all these weights, trying to minimize the final loss of the NN model. [24]

However, sometimes the gradient may be too small to update the weights effectively. Backpropagation uses a chain rule to update all the nodes, which means its effect on each layer gets exponentially smaller with each layer inside the model. Therefore, deep networks require longer to train correctly. The further each layer is from the output, the less impact backpropagation has on its weights. [23, 17]

This can also happen when the NN only uses activation functions that produce values from the range [0, 1] (like the sigmoid function). That is why other functions such as rectified linear unit (ReLU) were introduced. They output values in the range [0, inf], thus introducing more variety.

### 2.2.7 Exploding gradient

The other extreme is the exploding gradient problem. This problem happens when the accumulated error is too large, resulting in significant changes in weights, which results in unstable NN models.

The NN is jumping from one extreme to another, unable to learn anything from the data it receives.

A possible way to prevent this is to use weight regularization. Using an L1 or L2 penalty on the recurrent weights can help with exploding gradients. For example, L1 – absolute weights penalty, L2 – squared weights penalty. [17]

### 2.2.8 Dropout, Residual networks

The idea of dropout is to randomly deactivate a fraction of the units in a network on each training iteration. This makes it so each neuron is not entirely dependent on other nodes. This change naturally helps avoid overfitting, making the trained model better generalized. [23]

Residual layers are a second approach for network generalization. These are skip layers – they provide a way to skip through a section of other layers. This addition makes it harder for vanishing gradient problem to arise and to improve deep networks, where being too deep would increase the output error instead of making it smaller.

One way to implement residual layers is to introduce an add layer. It will combine the output of the previous layer and one other. It is possible to chain them together, creating skip connections from multiple layers to the same point. [12, 25, 8]

# Related work

## 3.1 Efficient sub-pixel convolutional neural network

The efficient sub-pixel convolutional neural network (ESPCNN) utilizes sub-pixel upscaling methods. Instead of using interpolation-based upsampling from Section 1.1, this method directly learns the upsampling filters. First, a convolution pass creates several layers, corresponding to upscaling factor squared – x3 upscaling produces nine layers for each color channel. These layers are then placed next to each other to produce a larger output, visualized in Figure 3.1. [21]

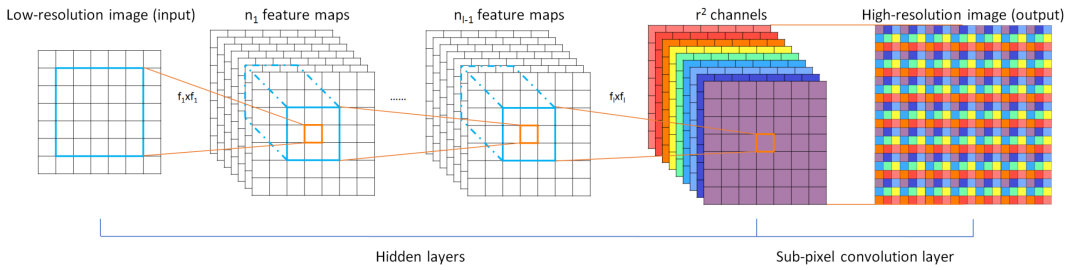


Figure 3.1: ESPCNN with two convolution layers for feature maps extraction, and a sub-pixel convolution layer that aggregates the feature maps from LR space and builds the SR image in a single step. [21]

The suggested upscaling from LR to HR is performed at the network's end, eliminating the need to perform most SR operations in a much larger HR resolution.

### 3.1.1 Datasets

Various datasets are used for this model training: Timofte dataset with 91 training images and 2 test datasets, Set5 and Set 14, providing 5 and 14 images; the Berkeley segmentation dataset BSD300 and BSD500 providing 100 and 200 images for testing and super texture dataset providing 136 texture images. In addition, 50 000 randomly chosen images from ImageNet are used for the final model.

For each scaling factor, a separate specific network is trained. PSNR is used as a performance metric.

### 3.1.2 Training

In order to synthesize low-resolution samples, HR images are blurred using a Gaussian filter and subsampled by the upscaling factor. The noise is added to simulate the camera's point spread function.

For the activation function, tanh is used instead of the ReLU function. This choice was motivated by their experimental results.

The training stops after no improvement of the cost function is observed after 100 epochs. The initial learning rate is set to 0.01, and the final learning rate is set to 0.0001 and gradually updated when the improvement of the cost function is less than a set threshold.

## 3.2 Coupled deep autoencoder

The aim of the coupled deep autoencoder (CDA) for SR is to learn a compact representation of the input while retaining the most critical information. This representation is expected to reconstruct the original input completely. And a

second exact representation for the output. [28]

Autoencoders can learn instrumental representations of their inputs. However, they cannot model the relationship between a sample pair of LR and HR images, only a single sample image.

CDA has a three-stage architecture, as shown in Figure 3.2. The first and third stages utilize two autoencoders. The first autoencoder is used to learn the representations of an LR image, and the second autoencoder is used to learn the representations of the HR image. The second stage incorporates a one-layer neural network to transform the representations from LR to HR.

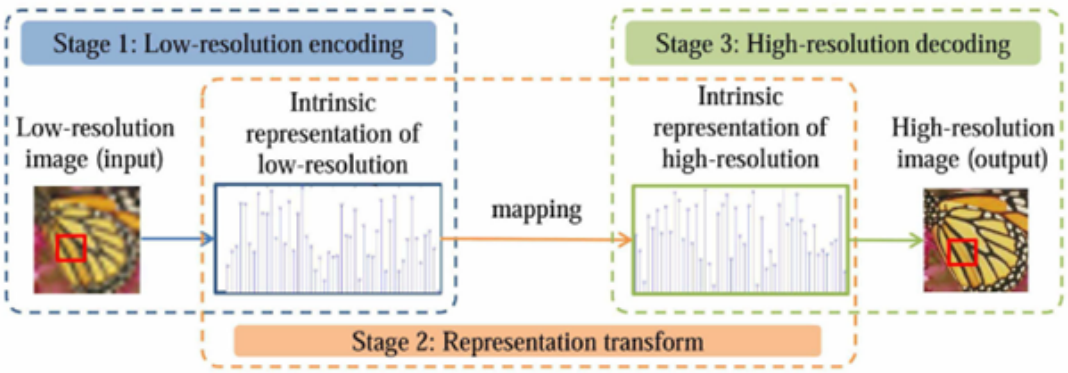


Figure 3.2: Flow diagram showing how the CDA performs a single-image SR. [28]

### 3.2.1 Datasets

Experiments were conducted on 91 images for training. For testing, Set5 containing 5 images was used to assess the performance of upscaling factors 2-4, and Set14 containing 14 images for upscaling factor 3. As an evaluation metric, PSNR was used.

### 3.2.2 Training

CDA needs to discover the LR – HR representations. For this purpose, a 2-part training procedure was designed: the first stage is an initialization, creating the LR representation and the HR representation, and mapping them in a non-linear way; the second stage is used for fine-tuning the weights, utilizing backpropagation. The whole process is visible in Figure 3.3.

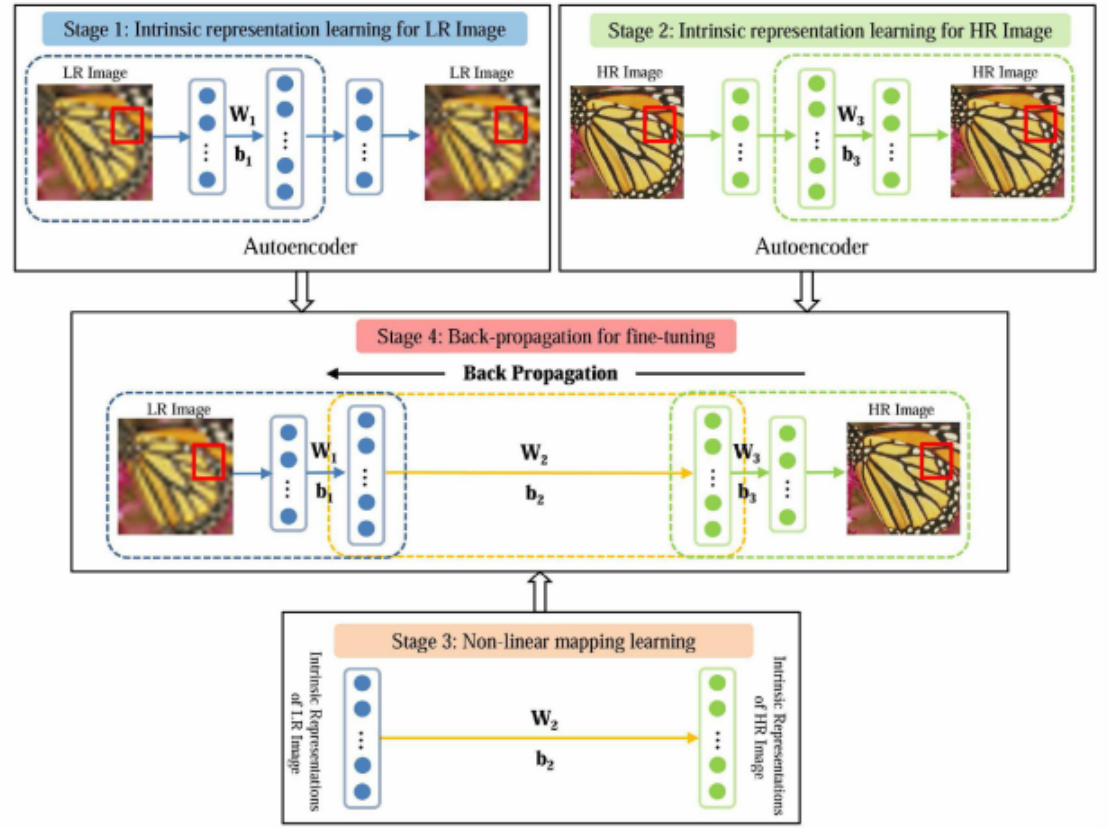


Figure 3.3: Flow chart showing the 2-phase CDA training procedure. [28]

### 3.2.3 Pretraining

To show the importance of the initialization part during the training procedure, two CDAs are used. One uses pre-training, and one does not. In images from

Set5, the difference of average PSNR is 0.86 dB on the 2x upscale factor in favor of the pre-trained model. Furthermore, on the images from Set14, the PSNR is 0.24 dB higher on the pre-trained model.

The parameter space of deep NN is vast (millions of parameters), so an improper initialization of the model may result in a poor solution after training. Therefore, the pretraining sets them as close as possible to the solution, capturing the statistical structure of the input data.

## 3.3 Super-resolution convolutional neural network

The super-resolution convolutional neural network (SRCNN) directly learns an end-to-end mapping between LR and HR images. The difference from existing approaches is that the SRCNN does not explicitly learn the dictionaries or manifolds for modeling the patch space instead of learning them via hidden layers. Furthermore, patch extraction and aggregation are also performed through convolutional layers and are involved in optimization. The SR pipeline is entirely created through learning, with little pre-processing or post-processing. [7]

The SRCNN is proposed with simplicity in mind. It utilizes a moderate number of filters and layers, displayed in Figure 3.4, so the model achieves a fast speed for practical online usage, even on a CPU. This is achieved by using a fully feed-forward method without the need to solve the optimization problem.

The research shows that a better result can be achieved by using a large and more diverse dataset or a deeper structure, but it comes at the cost of computation time.

The presented model is a fully convolutional neural network. It directly learns the end-to-end image mapping between LR and HR, with only a little pre-processing

and post-processing. Furthermore, this model implements a 3-channel image upscaling in YCbCr or RGB color space.

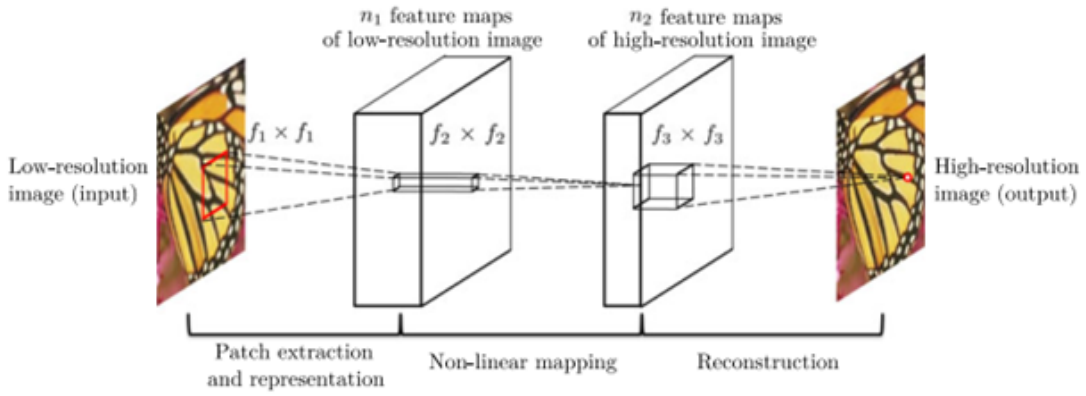


Figure 3.4: Image mapping steps. First is feature extraction, second is non-linear mapping, and third is image reconstruction. [7]

At first, a bicubic upscaling is used, which is the only pre-processing performed in this network. As a first step, a convolutional layer extracts features from the upscaled image. These feature maps are stored as a high-dimensional vector, each layer representing one feature.

The second step is non-linear mapping. This case means mapping the high-dimensional vector onto a second high-dimensional vector, extracting another layer of feature maps.

The third step is reconstruction, combining the extracted feature maps from the second layer to create the final HR image.

### 3.3.1 Parameters and filters

This model only uses 8 032 parameters, so it is unlikely to overfit. This decision was made to provide a fast image restoration model. Research shows that a broader or deeper model could provide superior results, but at the cost of computation

time.

Larger filters are also examined. Experiments show that a larger filter can provide better results, so utilizing neighborhood information in the mapping stage is beneficial. However, this increases the number of parameters by a significant amount (twice, thrice) but with little benefit. Therefore, the filter size is a trade-off between performance and speed.

A deep neural network is also experimented with by adding another non-linear mapping layer with 16 filters. The 4-layer network converges slower than the 3-layer network. However, given enough time to converge, the results are comparable to those of the 3-layer network. Furthermore, adding another layer with 32 filters degrades the performance and provides even worse results than the original 3-layer network.

Enlarging the filter size does not help this deeper network either and it still provides inferior results.

### 3.3.2 Datasets

Deep learning generally benefits from big data training. For comparison, a small dataset consisting of only 91 images was used, and a large one consisting of 395,999 images from the ILSVRC 2013 ImageNet detection training partition. This comparison is available in Figure 3.5.

### 3.3.3 Training

MSR is used as a loss function, and PSNR is used as an evaluation metric. The learning rate is 0.0001 for the first two layers and 0.00001 for the last layer. These values were empirically found; lowering the learning rate in the last layer leads to better results. The output image is smaller than expected. This is done to avoid

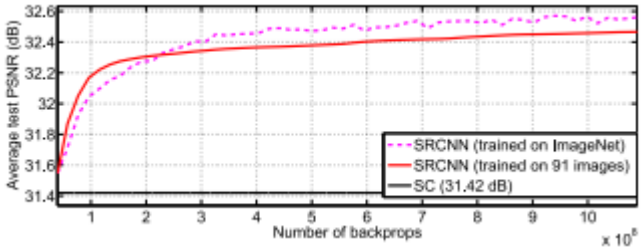


Figure 3.5: Comparison between small and large datasets. The model trained on the larger dataset shows better performance than the smaller one. [7]

border effects during training because all convolutional layers have 0 padding. Therefore, MSE is only evaluated on the central pixels of the two images.

Although the NN was trained on a size-fixed dataset, it can be used on images of arbitrary sizes during testing.

## 3.4 Deep super resolution

The variable deep super-resolution (VDSR) network can upscale an image by any factor. This is achieved by feeding the network an image that has already been upscaled using bicubic interpolation. However, it increases computation time and memory requirements because all operations are done on a larger image. [12]

Enhanced deep super-resolution (EDSR) is a proposed solution for single image SR with a set scale.

The proposed models do not use batch normalization because it removes the range flexibility within the networks. They experimentally show that this change substantially increases performance. This change also reduces GPU load and memory usage, allowing for building a larger and more complex model with better performance.

### 3.4.1 Datasets

The DIV2K dataset is used for the evaluation. It contains 800 training images and 100 validation images. The LR images are split into two categories: images down-scaled using bicubic interpolation and images down-scaled with the unknown operation. Set5, Set14, B100, and Urban100 are also used.

### 3.4.2 Training

PSNR is used as an evaluation metric to compare the model against other state-of-the-art networks. SSIM is also used as an additional metric.

For upscaling factors x4 and x3, the network is initialized with parameters from the pre-trained x2 network. This strategy accelerates training and improves the performance achieved.

During training, the network takes patches of RGB images of size 48 x 48 pixels. Then, these are randomly rotated and flipped by a factor of 90° in any direction. The images are also pre-processed by subtracting the mean RGB value of the whole DIV2K dataset.

The mini-batch size is 16, the initial learning rate is 0.0001, and the ADAM optimizer is used.

## 3.5 Related work conclusion

Based on the related work models, it is apparent that increasing the number of nodes results in better images. However, the trade-off is that the more nodes are added, the less substantial improvements are seen. So, a choice must be made as to whether it is worth keeping increasing the size of the network or sacrificing the

minor potential improvements.

Larger datasets produce more accurate results than smaller ones. However, broader datasets come at the cost of training time. To compare two models, they must be trained on the same dataset.

# Design

Based on the related work, two models are suggested, and one basic u-net as a point of comparison. The first model works with a preset upscaling factor 2. This model takes an image as input and upscales it by the factor to a higher-resolution output. The second model can perform variable dynamic upscaling without a preset factor. This model takes an image plus the final image size as inputs and produces the higher resolution output with the set size. Finally, the u-net model takes a picture input and produces a 2x upscaled output.

## 4.1 Models

All models are created using convolutional layers. The two suggested models are fully convolutional. They do not use any downscaling layers, like maxpool or averagepool. This change allows the models to efficiently work with any resolution inputs without additional pre-processing required to adjust their size. It also allows for the usage of varying dataset image sizes, even the switch from horizontal to vertical.

The u-net model uses maxpool downscaling layers. Therefore, it either has to work with a specific sized input image, or the input images must be pre-processed, adding extra padding to fit the desired output size.

### 4.1.1 Sub-pixel residual convolutional super-resolution

As the title suggests, this model utilizes sub-pixel convolutional upscaling. This choice was made based on „Deep Learning for Image Super-Resolution: A Survey“ [26] article as the most popular upscaling layer method.

The model consists of multiple convolutional blocks, with three convolutional layers each. After each block, there is a residual skip connection, connecting the previous block with the current one, all available in Figure 4.1.

After the sub-pixel convolution layer, there are more convolutional blocks. These blocks can now work with the positional data as it is correctly supposed to be outputted, allowing for better accuracy.

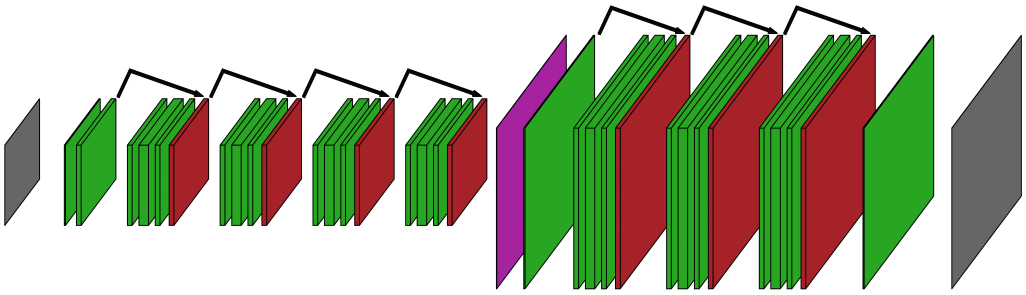

Figure 4.1: Visualization of the SRCSR model to scale. The layer thickness indicates the layer depth. The input goes through pre-processing layers, four blocks of three convolutional and one concatting layer, a sub-pixel convolution layer, three more blocks, and a post-processing layer.

### 4.1.2 Dynamic residual convolutional super-resolution

This dynamic model is based on the variable deep super-resolution model mentioned in Section 3.4.

The base structure of this suggested model is the same as that of the previously suggested model. It consists of convolutional blocks with skip connections that connect them. The difference is that the DRCSR takes an already upscaled image as input, and the size stays the same as the output. The whole model is displayed in Figure 4.2.

The model first processes the input image, immediately upscaling it to the target resolution, and only then performs convolution on this image. The initial pre-processing is done linearly, using a bicubic interpolation method, mentioned in Section 1.1.

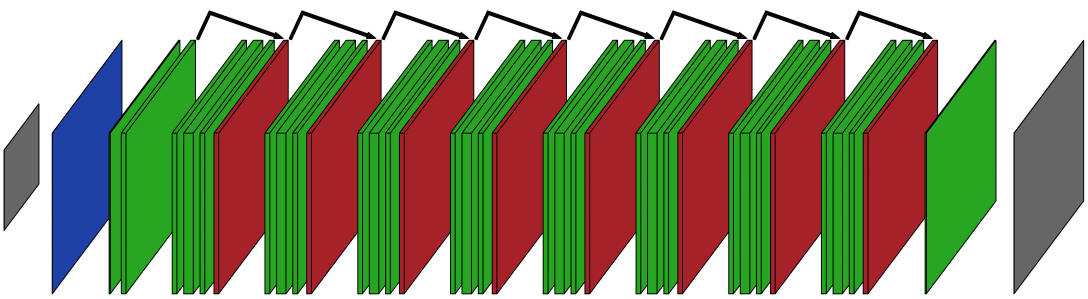

Figure 4.2: Visualization of the DRCSR model to scale. The layer thickness indicates the layer depth. First, bicubic upscaling is applied, and the result is used as input. This input goes through pre-processing layers, eight blocks of three convolutional layers with a concatenating layer, and a post-processing layer.

### 4.1.3 U-net

This convolutional neural network was first developed for biomedical image segmentation.

The architecture of this model consists of multiple parts. In the first half, there

are downscaling blocks. These blocks consist of a few convolutional layers, plus a max-pooling layer at the end. One upscaling block in the second half for each downscaling block consists of convolutional layers and an upscaling layer utilizing bicubic interpolation. These upscaling blocks start with a skip connection, connecting the same-sized downscaling block. [18]

For this autoencoder to be used in super-resolution, it has to have one more upscaling block at the end, allowing for a 2x upscale factor, as shown in Figure 4.3.

One limitation of this model, as mentioned above, is that it only works with specific-sized images, with a resolution divisible by 32. If this model were applied to a different-sized image, it would require some preprocessing to add extra padding by mirroring the edge pixels. Otherwise, the final result will not be exactly 2x the original scale.

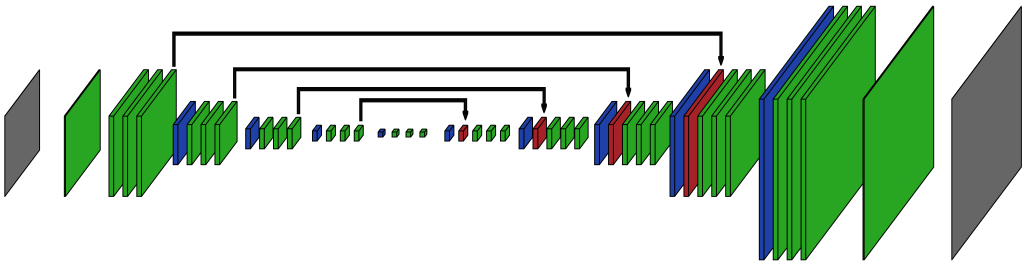

Figure 4.3: Visualization of the U-net model to scale. The layer thickness indicates the layer depth. Each downscale block contains a max-pooling layer and three convolutional layers. Each upscaling block contains an upscaling layer, a concatenating layer, and three convolutional layers.

## 4.2 Dataset

Four datasets are used to compare the results of these three models with state-of-the-art (SOTA) models. In addition, these datasets are used in „Deep Learning for Image Super-Resolution: A Survey“ [26] to compare many SOTA networks, making it easier to compare these models with existing ones accurately.

The datasets are as follows: Set5 [5], Set14 [29], B100 [14] and Urban100 [10].

PSNR is measured for each dataset, and the final result is the mean of the four intermediate results.

## 4.3 Training

The models are trained on the DIV2K dataset [1]. This is a more extensive dataset collection consisting of 800 training images and 100 validation images. The LR images are in two formats: one uses bicubic downsampling and the other an unknown downsampling method.

Due to how LR images are synthesized from HR images using a downscaling algorithm, the neural network is learning the inverse of this algorithm. [26, 3, 21]

That is why the unknown downscaled images are used, where possible, for training in this article.

### 4.3.1 Pre-processing

#### 4.3.1.1 Cropping

Because the U-net only supports images with a resolution divisible by 32, and the DIV2K dataset is used for training, some pre-processing must be done for this

model.

All LR images have cropped sides for their resolution to match a factor of 2, and their HR pairs are twice their resolution. This change is performed before any training, allowing the model to load these pre-processed images and work with them directly.

#### 4.3.1.2 Rescaling

The DRCSR model requires an already upscaled image input. In order to produce an input dataset with a varying scaling factor, the formula described in Figure 4.4 was used. Therefore, the input images will have a different upscaling factor, between 1 and 8.

$$fac = 2^{r*3}$$

Figure 4.4: Formula to produce a random scaling factor, where $r$ is a random number between 0 and 1. This produces 3 sets of factors of equal size, with ranges $[1, 2]$, $[2, 4]$, $[4, 8]$.

#### 4.3.1.3 Transformations

All suggested models can work with images of varying sizes. This allows dataset broadening. For example, rotating each image by a factor of 90°, produces four times the size of our original dataset. Furthermore, all of these images are flipped vertically to create eight times the size of the original dataset.

These simple transformations produce loads of new data, allowing the models to avoid overfitting and better results.

#### 4.3.1.4 Patching

After some experiments, it is apparent that the models cannot train on images with full resolution. So pathing is implemented as a pre-processing step.

Each image is split into patches with resolution $[128, 64]$. This resolution was chosen to fit the U-net size requirement and maximize the number of patches produced from each image.

### 4.3.2 Experiments

All three models were trained and tested. However, after a few iterations, the SRCSR and U-net produced distorted results, mimicking the Moire effect. Therefore, to better focus, only DRCSR was tested afterward.

#### 4.3.2.1 Optimizer

The results, visible in Figure 4.5, were very similar using parameter sweep and testing of both the $ADAM$ and *sgd* optimizers. However, the $ADAM$ optimizer was able to produce better results at the end, which is why it was chosen.

#### 4.3.2.2 Loss function

Using parameter sweep in Figure 4.6 again, the optimizer set to $ADAM$, $MSE$ and *binary cross-entropy* were tested, with $MSE$ producing slightly better results.

#### 4.3.2.3 Kernel size

In order to find the optimal kernel size, a second parameter sweep was used. In this sweep, the optimizer was set to $ADAM$ and the loss function to $MSE$.

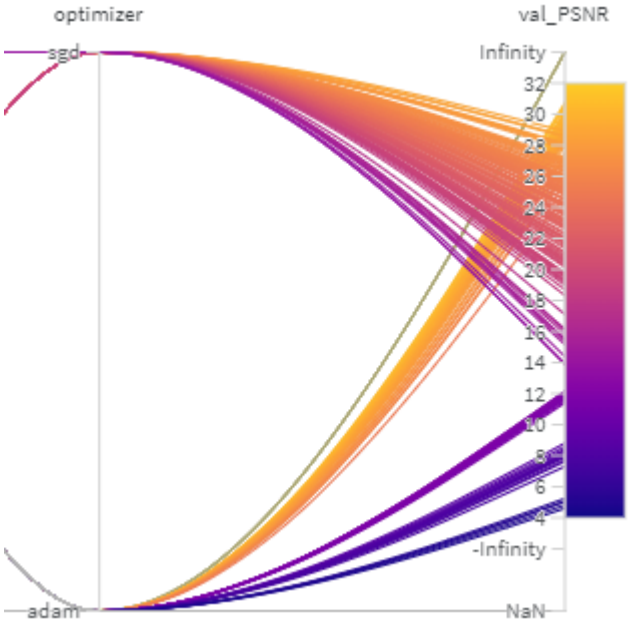


Figure 4.5: Parameter sweep showing the optimizer choice.

The result of this sweep, displayed in Figure 4.7, is that a model with any kernel size can produce a good result. However, the best result was achieved with kernel sizes of $(3, 3)$, but the last was higher $(5, 5)$.

#### 4.3.2.4 Patch count and epoch count

Experiments on the DRCSR model show that a higher patch count produces better results but increases training time per epoch. Increasing the number of epochs also produces better results.

However, this work's time and scope limitations cannot be tested further on larger sizes and longer training times.

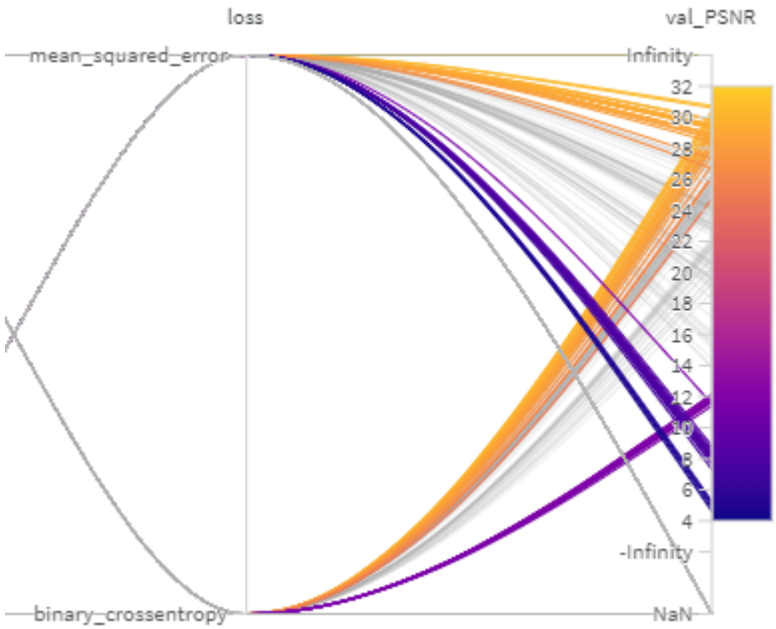


Figure 4.6: Parameter sweep showing the loss choice.

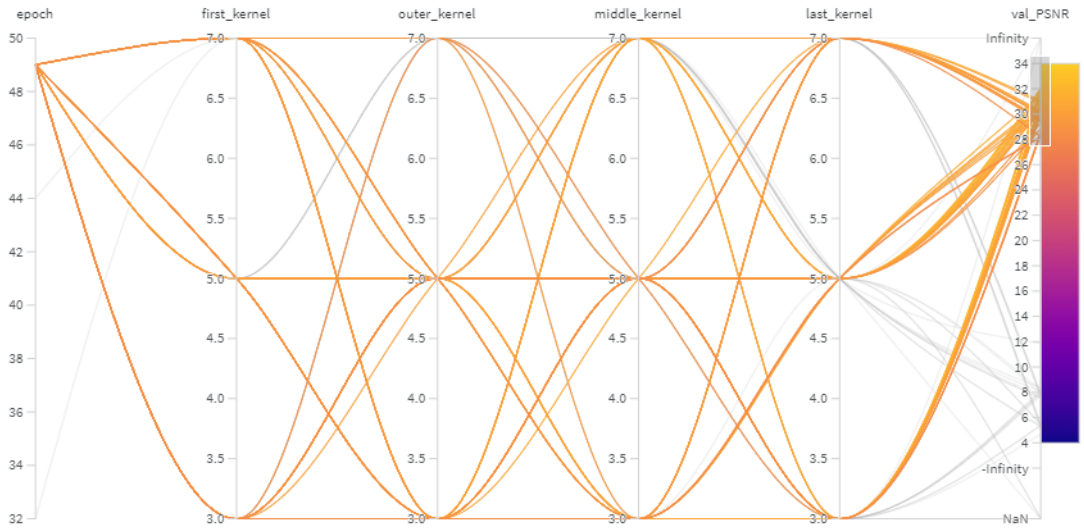


Figure 4.7: Parameter sweep showing the kernel size choice.

# Results

## 5.1 Quantitative evaluation

The final model has 6 million trainable parameters spread across 27 convolutional layers, and the training took 100 epochs.

The DRCSR model achieved a PSNR value of 29.68 for upscaling with an upscaling factor x2, and 27.18 PSNR for upscaling with a random factor, between 1-8. The second dynamic results were achieved with a custom synthesized dataset, as mentioned in Section 4.3.1.2

This evaluation was carried out in the same manner as in „Deep Learning for Image Super-Resolution: A Survey“ [26]. Upon further investigation, the results appear to be the best in Set5 [5], with a PSNR value of 33.04 and 29.60 for dynamic upscaling, and the worst results are for Urban100 [10], with only 27.11 and 24.64 PSNR. These results are visualized in Table 5.1

The lower result may be explained by the lack of buildings and other structures in the training dataset that make up most of the Urban100 dataset.

## 5.2 Qualitative evaluation

All the images displayed in Figure 5.1, Figure 5.2, Figure 5.3, Figure 5.4 and Figure 5.5 are compiled in a manner displayed in Table 5.2.

| dataset | dynamic upscaling | 2x upscaling |
|---|---|---|
| Set5 | 29.60 | 33.04 |
| Set14 | 27.50 | 29.26 |
| BSD100 | 27.00 | 29.30 |
| Urban100 | 24.64 | 27.11 |
| Mean | 27.18 | 29.68 |

Table 5.1: Individual PSNR results.

| dynamic upscaling factor | 2x upscaling factor |
|---|---|
| bicubic input | bicubic input |
| predicted output | predicted output |
| original image | original image |

Table 5.2: Image results composition.

Figure 5.1: Set5 image. The dynamic upscaling image was downscaled by a factor of 4.386. Low-resolution input was syntesized from an image with resolution 116x116 for dynamic, and 256x256 for static upscaling. The final output is 512x512.

Figure 5.2: Set14 image. The dynamic upscaling image was downscaled by a factor of 3.691. Low-resolution input was syntesized from an image with resolution 138x138 for dynamic, and 256x256 for static upscaling. The final output is 512x512.

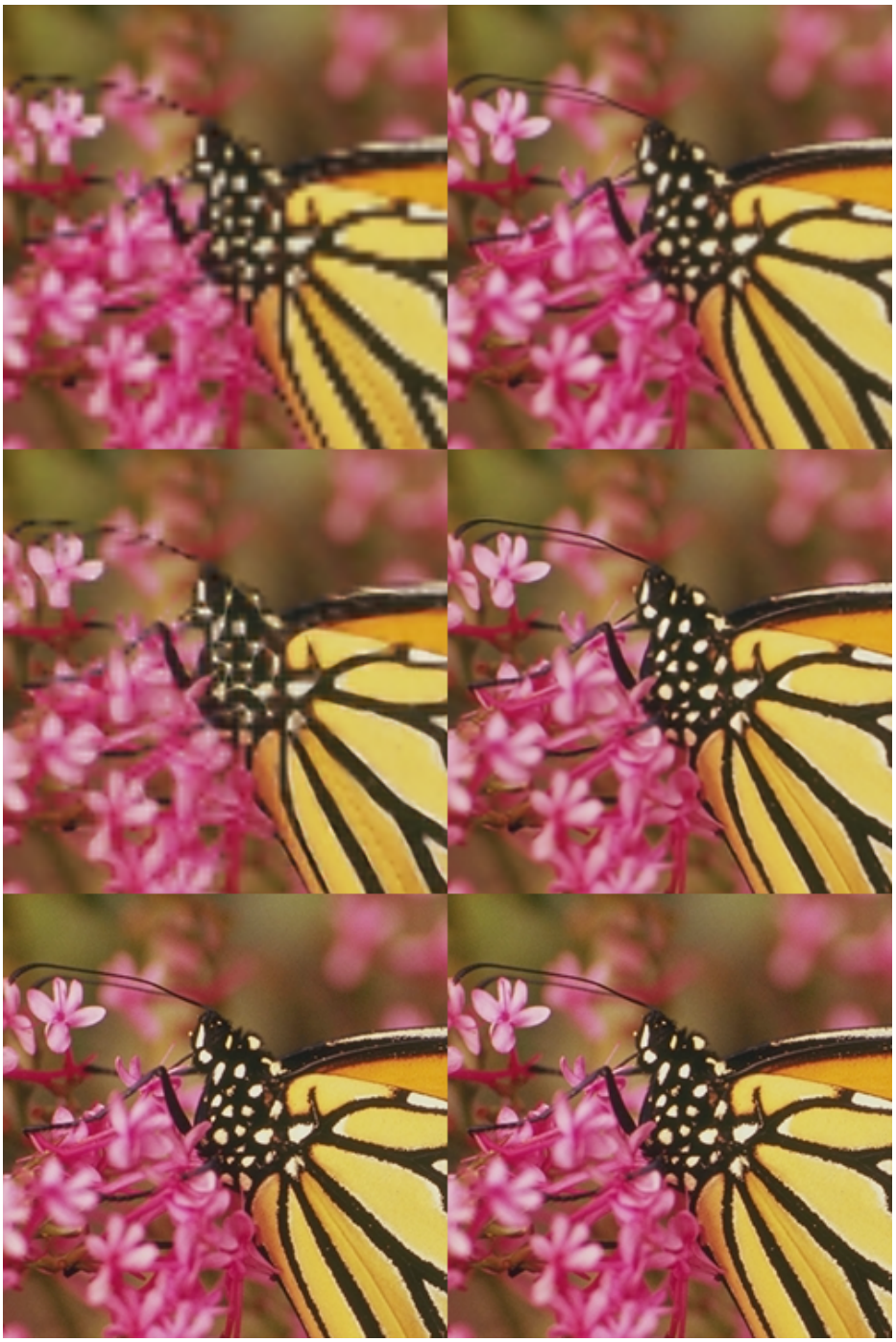

Figure 5.3: Set14 image. The dynamic upscaling image was downscaled by a factor of 3.805. Low-resolution input was syntesized from an image with resolution 201x134 for dynamic, and 384x256 for static upscaling. The final output is 768x512.

Figure 5.4: BSD100 image. The dynamic upscaling image was downscaled by a factor of 3.164. Low-resolution input was syntesized from an image with resolution 101x151 for dynamic, and 160x240 for static upscaling. The final output is 320x480.

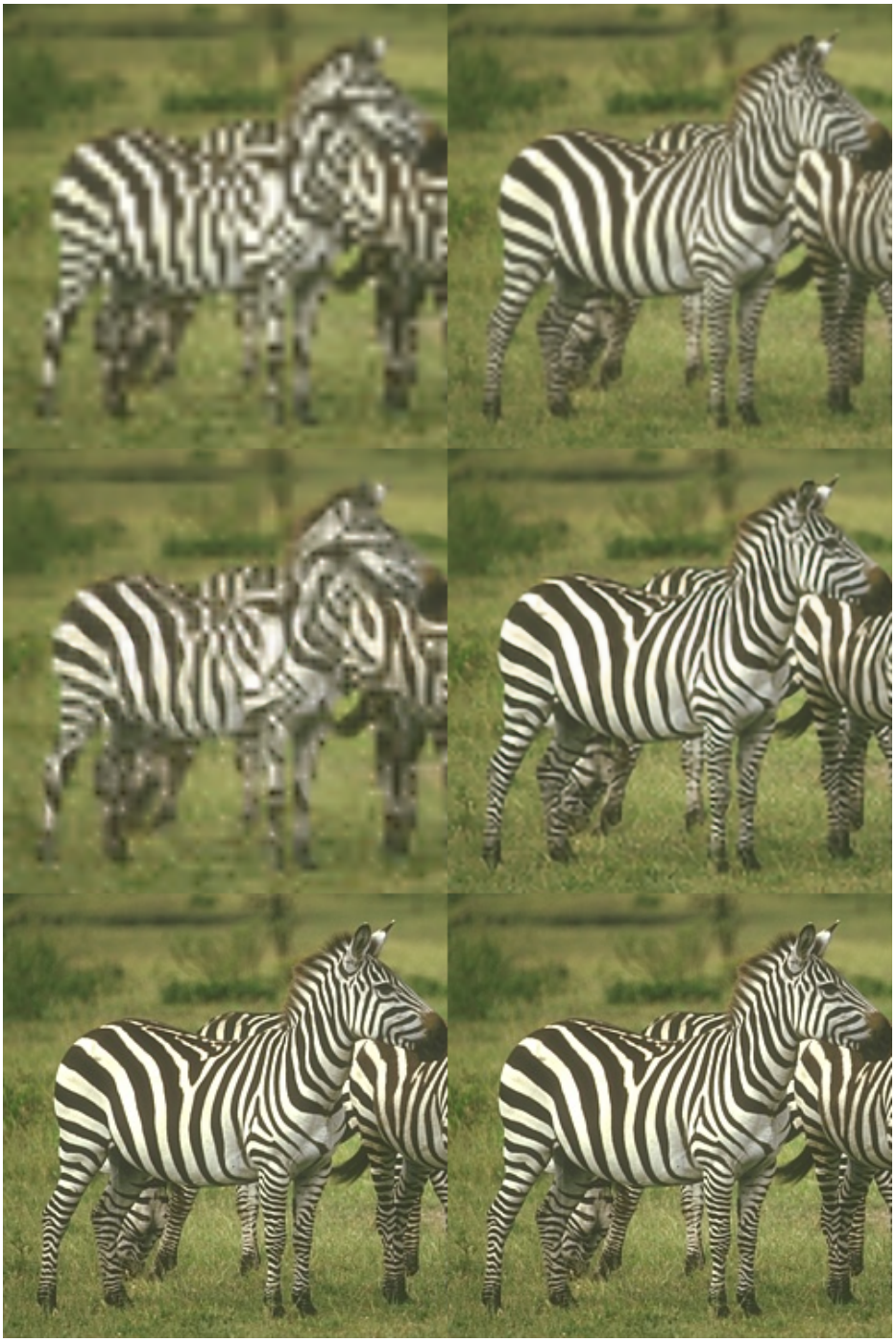

Figure 5.5: BSD100 image. The dynamic upscaling image was downscaled by a factor of 4.634. Low-resolution input was syntesized from an image with resolution 103x69 for dynamic, and 240x160 for static upscaling. The final output is 480x320.

# Conclusion

Multiple approaches were tested to achieve the best super-resolution possible. Unfortunately, some did not provide good enough examples, so they were discarded and discontinued.

The proposed dynamic model successfully achieved dynamic super-resolution. This means that it can upscale an image using a chosen upscaling factor. This factor can be any number, but for numbers greater than 4, the quality of the results achieved diminishes.

This model is created using TensorFlow and Keras platforms. They allow for abstraction over the whole process, making it easier to start prototyping and receive the first results. In addition, it utilizes a convolutional neural network, allowing for better abstraction.

Custom data pre-processing was used to generate more information from a dataset. This includes image rotation and mirroring.

## 6.1 Open research questions

It is possible to achieve better results. Here are some ways.

Smaller highest scaling factor - the results quickly lose quality with a scaling factor over 4. Reducing the maximum scaling factor to just four would produce more valuable data for better training.

More extensive dataset - even though this project's scope did not allow for a larger

dataset, the model would profit from a larger dataset. As a result, the training phase would take longer, but the results would be better.

This project's scope did not allow more extensive training over more data and extended periods. However, training a model for hundreds of epochs would increase the quality of the result.

Bigger model - the model used in this paper has about 6 million parameters. That is considerably less than other SOTA models by orders of magnitude. However, bigger models take longer to train, so it was impossible to utilize them.

# References


[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] *AMD FidelityFX Super Resolution*. URL: `https://gpuopen.com/fidelityfx-superresolution/` (visited on 10/2021).

[3] Saeed Anwar, Salman Khan, and Nick Barnes. "A deep journey into super-resolution: A survey". In: *ACM Computing Surveys (CSUR)* 53.3 (2020), pp. 1–34.

[4] Yoshua Bengio. *Learning deep architectures for AI*. Now Publishers Inc, 2009.

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

[6] Francois Chollet. *Building Autoencoders in Keras*. 2016. URL: `https://blog.keras.io/building-autoencoders-in-keras.html` (visited on 11/2021).

[7] Chao Dong et al. "Image super-resolution using deep convolutional networks". In: *IEEE transactions on pattern analysis and machine intelligence* 38.2 (2015), pp. 295–307.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[9] Alain Hore and Djemel Ziou. "Image quality metrics: PSNR vs. SSIM". In: *2010 20th international conference on pattern recognition*. IEEE. 2010, pp. 2366–2369.

[10] 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.

[11] Phil Kim. "Convolutional neural network". In: *MATLAB deep learning*. Springer, 2017, pp. 121–147.

[12] 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.

[13] Weibo Liu et al. "A survey of deep neural network architectures and their applications". In: *Neurocomputing* 234 (2017), pp. 11–26.

[14] David Martin et al. "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics". In: *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*. Vol. 2. IEEE. 2001, pp. 416–423.

[15] Keiron O'Shea and Ryan Nash. "An introduction to convolutional neural networks". In: *arXiv preprint arXiv:1511.08458* (2015).

[16] Umut Orhan, Mahmut Hekim, and Mahmut Ozer. "EEG signals classification using the K-means clustering and a multilayer perceptron neural network model". In: *Expert Systems with Applications* 38.10 (2011), pp. 13475–13481.

[17] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks". In: *International conference on machine learning*. PMLR. 2013, pp. 1310–1318.

[18] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[19] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536.

[20] Ramadass Sathya, Annamma Abraham, et al. "Comparison of supervised and unsupervised learning algorithms for pattern classification". In: *International Journal of Advanced Research in Artificial Intelligence* 2.2 (2013), pp. 34–38.

[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] Olaf Sporns. "Structure and function of complex brain networks". In: *Dialogues in clinical neuroscience* 15.3 (2013), p. 247.

[23] Heung-Il Suk. "An introduction to neural networks and deep learning". In: *Deep Learning for Medical Image Analysis*. Elsevier, 2017, pp. 3–24.

[24] Daniel Svozil, Vladimir Kvasnicka, and Jiri Pospichal. "Introduction to multilayer feed-forward neural networks". In: *Chemometrics and intelligent laboratory systems* 39.1 (1997), pp. 43–62.

[25] Zhaowen Wang et al. "Deep networks for image super-resolution with sparse prior". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 370–378.

[26] 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* 43.10 (2020), pp. 3365–3387.

[27] *What is the difference between Nearest Neighbor, Bilinear Interpolation and Cubic Convolution?* 2020. URL: `https://support.esri.com/en/technical-article/000005606` (visited on 10/2021).

[28] Kun Zeng et al. “Coupled deep autoencoder for single image super-resolution”. In: *IEEE transactions on cybernetics* 47.1 (2015), pp. 27–37.

[29] 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.

[30] Hang Zhao et al. “Loss functions for image restoration with neural networks”. In: *IEEE Transactions on computational imaging* 3.1 (2016), pp. 47–57.

# Technical documentation

## A.1 Usage

This section describes the user manual on using the model to predict an image for image super-resolution.

### A.1.1 Requirements

Python version 3.9.1 was used for the development, but any Python version above 3.7 will likely work.

Two packages from Nvidia: cudatoolkit 11.2 and cuDNN 8.2.1. The versions of these two packages depend on each other and other Python libraries.

Python libraries (can be easily installed: `pip install library_name==version`):

- keras 2.6.0
- numpy 1.19.5
- opencv-python-headless 4.5.4.58
- tensorflow 2.6.0
- pillow 9.1.0

If WandB logging is used, the wandb package must be installed.

If Azure ML is used, the azureml packages must be installed.

These are the packages and their versions that were used during development.

### A.1.2 Modifications

Some public libraries did not support specific required functions, so they had to be modified. This change is done within the `keras2.py` file. It adds more upscaling methods for the convolutional layers. A later version of Keras should support all of these methods without any modifications.

### A.1.3 Files

```
source
├─ DRCSR.h5   The model itself.
├─ DRCSR.py   Single-instance model training.  Can log info to WandB.
├─ keras2.py   Modified Keras library functions.
├─ nn_plus.py   Utility functions; image generators and callbacks.
├─ predict.py   Module for single-image prediction.
└─ wandb_sweep.py   Parameter sweep using YAML config file.
```

### A.1.4 Single-image prediction

To test the single-image prediction, run `predict.py`, select an image to upscale, choose upscaling factor, and wait for it to save the final output.

### A.1.5 Model

The DRCSR model consists of multiple convolutional blocks. The convolutional block is displayed in Table A.1, and the whole model is displayed in Table A.2.

| Layer | Kernel size | Depth | Connects to |
|---|---|---|---|
| conv0 | 3, 3 | 128 | input |
| conv1 | 3, 3 | 256 | conv0 |
| conv2 | 3, 3 | 128 | conv1 |
| add | - | - | input, conv2 |

Table A.1: Single convolutional block.

| Layer | Kernel size | Depth | Connects to |
|---|---|---|---|
| conv0 | 3, 3 | 32 | input |
| conv1 | 3, 3 | 128 | conv0 |
| block0 | - | - | conv1 |
| block1 | - | - | block0 |
| block2 | - | - | block1 |
| block3 | - | - | block2 |
| block4 | - | - | block3 |
| block5 | - | - | block4 |
| block6 | - | - | block5 |
| block7 | - | - | block6 |
| conv2 | 5, 5 | 32 | block7 |
| conv3 - output | 3, 3 | 3 | conv2 |

Table A.2: DRCSR model visualization.

# Plán práce

## B.1 Prvá etapa

| Týždeň semestra | Plán práce v prvej etape | Vyjadrenie |
|---|---|---|
| 1. | Oboznámenie sa s problematikou (MIT video kurzy). | Plán bol splnený. |
| 2. | Testovanie zmenšovania rozlíšenia cez cv2. | Plán bol splnený. |
| 3. | Porozumenie problematike (YT DeepLearningAI). | Plán bol splnený. |
| 4. | Vytváranie jednoduchého binárneno klasifikátora pomocou neurónových sietí. | Plán bol splnený. |
| 5. | Podrobná analýza Neurónových sietí na využitie image upscaling-u. | Plán bol splnený. |
| 6. | Vytvorenie prvého konvolučného prototypu. | Plán bol splnený. |
| 7. | Analýza related work. | Plán bol splnený. |
| 8. | Analýza residual neural networks, deep networks. Spísanie všeobecných informácii do BP. Pripravenie Azure prostredia | Plán bol splnený. |
| 9. | Analýza bilinear vs nearest neighbor upsampling. Analýza kernel size, normalizácie. Porovnanie metrík vyhodnocovania. | Plán bol splnený. |
| 10. | Upravenie BP, pridanie algoritmov. | Plán bol splnený. |
| 11. | Pridanie citácii a obrázkov. | Plán bol splnený. |
| 12. | Finalizácia BP. | Plán bol splnený. |

## B.2 Druhá etapa

| Týždeň semestra | Plán práce v druhej etape | Vyjadrenie |
|---|---|---|
| 1. | Opravenie nedostatkov. | Plán bol splnený. |
| 2. | Pridanie návrhu modelov. | Plán bol splnený. |
| 3. | Opísanie testovania a porovnania výsledkov. | Plán bol splnený. |
| 4. | Implementácia pre-processing-u. | Plán bol splnený. |
| 5. | Vytvorenie modelov. | Plán bol splnený. |
| 6. | Príprava generátorov. | Plán bol splnený. |
| 7. | Vizualizácia. | Plán bol splnený. |
| 8. | WandB logging. | Plán bol splnený. |
| 9. | WandB general sweep. | Plán bol splnený. |
| 10. | WandB kernel sweep. | Plán bol splnený. |
| 11. | Trénovanie finálneho modelu. | Plán bol splnený. |
| 12. | Dopísanie práce. | Plán bol splnený. |

# Resumé v slovenskom jazyku

## C.1 Super rozlíšenie obrázku

Je to metóda, ktorá sa používa na zvýšenie rozlíšenia obrázku, zo vstupu s nízkym rozlíšením, na výstup s vyšším rozlíšením. Cieľom v tejto oblasti je získať výstupný obrázok, ktorý bude mať rovnakú úroveň detailov ako vstupný obrázok.

Na riešenie tohto problému bolo vyvinutých viacero metód. Niektoré vyžadujú trénovanie modelu na množstve vstupných údajov, iné takéto požiadavky nemajú, a teda nevyužívajú neurónové siete.

### C.1.1 Interpolačný prístup

Existuje mnoho spôsobov, ako prevzorkovať obrázok. Najzákladnejším prístupom je duplikovanie všetkých pixelov na jednu stranu a následné duplikovanie všetkých riadkov pod seba. Toto sa nazýva algoritmus najbližšieho suseda (nearest neighbor algorithm) a v praxi sa bežne používa. Výsledok je hranatý a zubatý s ostrými hranami. Nevytvára žiadne nové informácie, iba duplikuje existujúce.

Ďalším prístupom, ktorý tiež mení existujúce údaje, je prevzorkovanie bilineárnou interpoláciou. Táto metóda vytvára nové pixely na základe svojich štyroch najbližších susedov a výsledný pixel má hodnotu váženého priemeru týchto hodnôt. Výsledkom sú jemnejšie hrany ako u najbližšieho suseda.

Bikubická interpolácia funguje rovnakým spôsobom ako bilineárna interpolácia, ale využíva väčšiu plochu, konkrétne 4x4 najbližšie pixely. Opäť sa vypočíta vážený

priemer, a vonkajšie pixely ovplyvňujú výsledok menej ako tie bližšie. [27]

### C.1.2 Využitie v hrách

Pre grafickú kartu je jednoduchšie načítať scénu v nízkom rozlíšení, a následne na ňu aplikovať nejaký algoritmus, ako načítavať scénu už priamo vo vysokom rozlíšení. Jedná sa teda o riešenie, ktoré dokáže zvýšiť rozlíšenie scény bez takmer žiadnej straty detailov.

Vďaka tomu, že sa scéna vykresľuje na nižšom rozlíšení je možné viackrát za sekundu aktualizovať snímok, vďaka čomu sa dá získať jemnejší výstup, pri viacerých snímkoch za sekundu.

## C.2 Neurónové siete

Založené na základe biologických nervových systémov, ako napríklad ľudský mozog, neurónové siete fungujú na podobnom princípe. Obe možnosti využívajú veľké množstvo vzájomne prepojených výpočtových uzlov, označovaných ako neuróny.

Umelé neurónové siete sú určené na spracovanie vstupných informácii, na vytvorenie nejakého výstupu - klasifikácia obrazu, predpoveď trendov, rozpoznávanie objektov, alebo iné. Vďaka miliónom neurónov sa dokáže neurónová sieť naučiť aj zložité vzorce, ktoré vie následne aplikovať na údaje, s ktorými sa doposiaľ nestretla.

Jeden konkrétny neurónový model ale je možné použiť len na jeden konkrétny prípad použitia. Každá sieť musí byť špecificky vytvorená pre jej použitie, a následne natrénovaná, počas stoviek alebo aj tisícok iterácii (epoch), pre jej konkrétnu oblasť. [15]

## C.3 Súvisiace práce

### C.3.1 Efficient sub-pixel convolutional neural network

Konvolúcia pod-pixelov (sub-pixel) nahrádza interpolačné zvýšenie rozlíšenia. V tomto prípade sa neurónová sieť priamo naučí, ako generovať nové informácie, a nie je závislá od interpolačného algoritmu, aby to za ňu vykonal. Ukážka sub-pixel konvolúcie je vo Figure C.1.

Namiesto klasickej *relu* aktivačnej funkcie použili v tejto práci *tanh* aktivačnú funkciu.

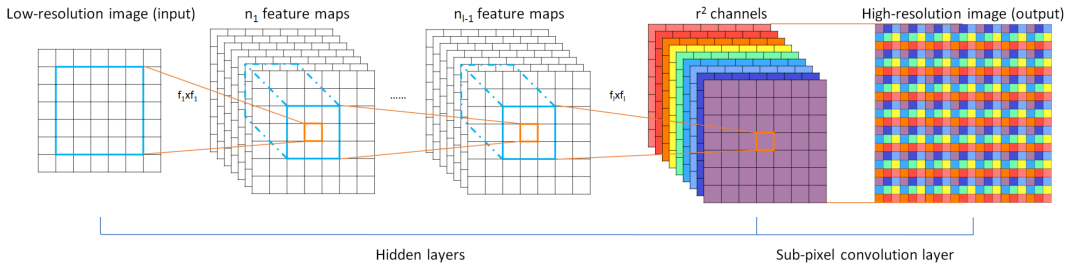


Figure C.1: ESPCNN s dvoma konvolučnými vrstvami na extrakciu máp významných čŕt, a sub-pixel konvolučnou vrstvou, ktorá agreguje mapy čŕt z priestoru s nízkym rozlíšením, a vytvára výsledný výstup v jednom kroku.

[21]

## C.4 Vlastný návrh

Na základe súvisiacich prác bol navrhnutý DRCSR model, na dynamické super-rozlíšenie. Výhoda tohto modelu je, že dokáže spracovať obrázok s ľubovoľným rozlíšením, a vytvoriť z neho nový obrázok, taktiež s ľubovoľným rozlíšením.

### C.4.1 Dynamic residual convolutional super-resolution

Tento model je plne konvolučný, takže nepoužíva žiadne vrstvy na zníženie rozlíšenia (ako u-net). Vďaka tomuto rozhodnutiu je model schopný spracovať obrázok s ľubovoľným rozlíšením bez potreby predspracovania na upravenie veľkosti.

Model je založený na VDSR modeli, ktorý je spomenutý v Section 3.4.

DRCSR model sa skladá z viacerých konvolučných blokov, ktoré sú navzájom prepojené aditívnymi vrstvami. Tento model dostane na vstupe už obrázok s vyšším rozlíšením, a nad ním následne vykonáva potrebné zmeny. Celý model je zobrazený v Figure C.2.

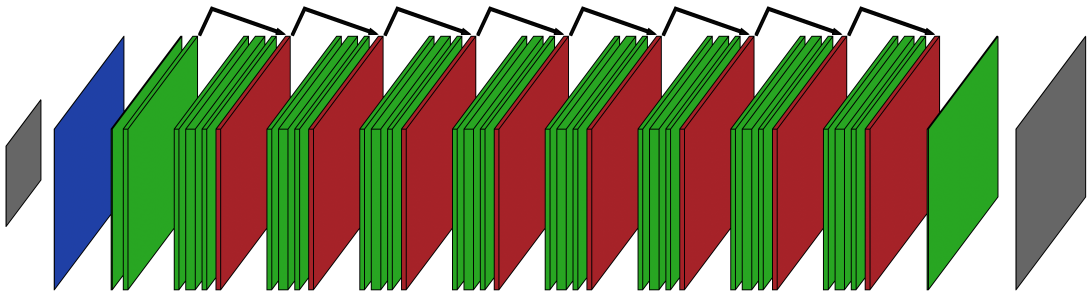

Figure C.2: Vizualizácia DRCSR modelu. Hrúbka každej vrstvy naznačuje jej reálnu hĺbku. Najprv je aplikované bikubické prevzorkovanie, ktoré prejde cez niekoko začiatočných vrstiev, osem blokov, z ktorých každý obsahuje tri konvolučné vrstvy, a jednu aditívnu, a na konci jedna výsledná vrstva.

## C.5 Záver

Navrhovaný model dosiahol dobré výsledky, takmer porovnateľné s inými modelmi zo súvisiacich prác. Tento model dokáže úspešne aplikovať dynamické super-

rozlíšenie na ľubovoľný vstupný obrázok, s ľubovoľným faktorom na zvýšenie rozlíšenia. Ak je faktor číslo väčšie ako 4, výsledky začínajú mať minimálne zlepšenie.

Tento navrhnutý model využíva TensorFlow a Keras platformy. Vďaka ich využitiu je možné aplikovať abstrakciu nad celým procesom, čo umožňuje rýchlejšie prototypovanie a získanie prvých výsledkov.

Model ďalej využíva vlastné predspracovanie údajov, na generovanie viacerých vstupov. Jedná sa o otáčanie a zrkadlenie obrazu.

### C.5.1 Otvorené výskumné otázky

Výsledky modelu je možné vylepšiť viacerými spôsobmi.

Nižší faktor na zvýšenie rozlíšenia - ak je väčší ako 4, výsledky rýchlo strácajú kvalitu.

Väčší dataset, dlhšie trénovanie, a väčší model. Aj keď rozsah tejto práce neumožňuje ani jedno z nich, spolu dokážu zlepšiť výsledky modelu.