UNIVERSITY OF ZAGREB

FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS No. 255

DEEP LEARNING MODELS FOR REAL-TIME VIDEO UPSCALING

Dino Grgić

Zagreb, June 2023

UNIVERSITY OF ZAGREB

FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS No. 255

DEEP LEARNING MODELS FOR REAL-TIME VIDEO UPSCALING

Dino Grgić

Zagreb, June 2023

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Zagreb, 10 March 2023

MASTER THESIS ASSIGNMENT No. 255

Student:	**Dino Grgić (0036516270)**
Study:	Computing
Profile:	Computer Science
Mentor:	prof. Željka Mihajlović

Title: **Deep Learning Models for Real-Time Video Upscaling**

Description:

Investigate algorithms for upscaling image resolution. Particular attention should be focused on the possibilities of applying deep learning to the investigated procedures. Develop and implement various techniques for increasing image resolution achievable in real-time. Consider a generative deep learning GAN model for upscaling video resolution. Develop program implementation that enables the analysis and comparison of the performed procedures. Show achieved results on different examples and evaluate the results as well as implemented algorithms. Implement the appropriate software solution. Use the Python programming language. Make the results of the thesis available online. Supply algorithms, source codes, and results with appropriate explanations and documentation. Reference the used literature and acknowledge the received help.

Submission date: 23 June 2023

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 10. ožujka 2023.

DIPLOMSKI ZADATAK br. 255

Pristupnik:	**Dino Grgić (0036516270)**
Studij:	Računarstvo
Profil:	Računarska znanost
Mentorica:	prof. dr. sc. Željka Mihajlović

Zadatak: **Modeli dubokog učenja za povećanje rezolucije video sekvence u stvarnom vremenu**

Opis zadatka:

Proučiti načine povećanja rezolucije slike. Posebice obratiti pažnju na mogućnosti primjene dubokog učenja za proučene postupke. Razraditi i implementirati razne tehnike povećanja rezolucije slike ostvarive u stvarnom vremenu. Razmotriti generativni model dubokog učenja GAN za povećanje rezolucije. Načiniti programsku implementaciju koja omogućuje analizu i usporedbu načinjenih postupaka. Na različitim primjerima prikazati ostvarene rezultate. Načiniti ocjenu rezultata i implementiranih algoritama. Izraditi odgovarajući programski proizvod. Koristiti programski jezik Python. Rezultate rada načiniti dostupne putem Interneta. Radu priložiti algoritme, izvorne kodove i rezultate uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 23. lipnja 2023.

Thanks to my parents Ksenija and Stipe, sister Nika and grandmother Anđa who gave me never-ending love and support through all my years in education. The person I am today is mostly because of you. All the close ones who are unfortunately not here to see me finish my degree, you will always be in my heart. Many thanks to all the amazing professors and teachers I had over my years in education. Hopefully one day you will be universally more appreciated (and paid) for the amazing job you are doing. From west to east: Luka, Mirta, Dunja, and Kruno, thank you for all the patience and love you give me. Because all of you my life is just a tad bit sweeter. Thanks to Unitfly for paying me all these years even though I am always annoyingly reminding them that I am not still finished with my degree. You are an amazing group of individuals and I cannot wait to start my professional career with you. Finally, thanks to my mentor prof. dr. sc. Željka Mihajlović for her academic input for this and previous academic work I made.

Hvala mojim roditeljima Kseniji i Stipi, sestri Niki i baki Anđi za vječnu ljubav i podršku koju su mi dali kroz sve godine mojeg obrazovanja. Vi iste imali najveći utjecaj na osobu koja jesam danas. Svi bližnji koji nažalost nisu više ovdje s nama da vide kako završavam fakultetsko obrazovanje, uvijek ćete biti u mom srcu. Puno hvala svim divnim profesorima i učiteljima koje sam imao kroz godine obrazovanja. Nadam se da ćete jednog dana biti više cijenjeni (i plaćeni) za nevjerojatan posao koji radite. Od zapada prema istoku: Luka, Mirta, Dunja, Vrc, hvala vam na svoj ljubavi i strpljenju kojeg mi pružate. Zbog vas moj život je barem malo slađi. Hvala Unitfly-u što me plaćaju sve ove godine iako ih stalno dosadno podsjećam da još nisam gotov s fakultetom. Divna ste grupa ljudi i jedva čekam započeti svoju profesionalnu karijeru s vama. I na kraju, hvala mojoj mentorici prof. dr. sc. Željki Mihajlović za akademske savjete za ovaj i sve prijašnje radove na fakultetu.

CONTENTS

INTRODUCTION

Image upscale is one of the seamless constant processes used within computer devices without us even realizing it. This process operates continuously within video players, image previewers, and image editing software like Photoshop. With the amount of different device screen resolutions we have today, image upscale is a necessary method of showcasing the same image on them. This thesis dives into the challenges of upscaling raster images, exploring the challenges involved. Additionally, the thesis provides an optimization process that can facilitate the successful implementation of real-time video upscaling in the future.

The structure of this paper is as follows. In the first chapter, we discuss the issues related to image, video, and real-time video upscaling. We then review the existing literature and state-of-the-art methods for real-time image upscaling. Subsequently, we explore mathematical upscaling methods, focusing on interpolation as the primary technique. Additionally, we delve into modern upscaling approaches utilizing convolutional neural networks (CNNs) and generative adversarial networks (GANs). In the subsequent chapter, we provide detailed insights into real-time upscaling methods that rely on GPU inference and further optimization. We also address the challenges associated with constructing a dataset for high-quality image upscaling. Furthermore, we examine metrics and the difficulties involved in assessing image quality using automated computer processes. In the following chapter, we offer a comprehensive overview of the program implementation of these methods. Finally, we present the results, comparing CPU, GPU, and optimized GPU inference for real-time video upscaling.

1. Background

Two most commonly used types of image formats are vector and raster graphics (Figure 1.1). Both types have different approaches to storing and visualizing image data.

Rasterized images contain pixel magnitude information for each pixel of the image in a two-dimensional data matrix. Values of pixel magnitude vary between image formats. Due to the limitation of the format, each image contains an exact number of pixels, which can be calculated by multiplying the height and width of the image in pixels. The most popular formats of images using raster graphics are .jpeg, .png, .gif, .bmp, etc.

Vectorized images on the other hand are not limited by pixel data information rather they define objects on images with specific mathematical formulas which describe shapes and curves. This allows us to view vectorized graphics in any resolution we want. Commonly used image formats for vector graphics are .svg, .wmf, .eps, .ai, etc.

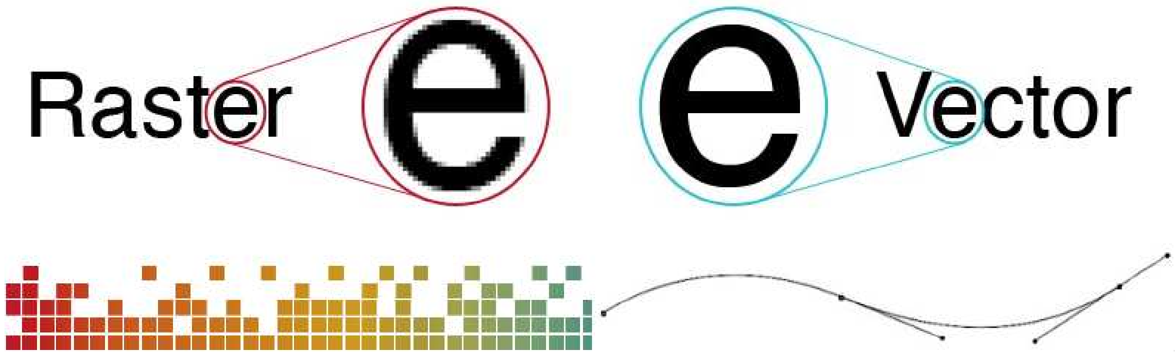


Figure 1.1: Comparison of raster (left) and vector graphics (right)

Some of the advantages of the usage of raster images over vector images are the possibility of single-pixel manipulation, simplicity of the data format, and matrix operations for image manipulation. However, vector images are usually small in size, accurately define curves, and can be both upscaled and downscaled without loss in quality.

Information about the most used image formats on the Internet shows us that mostly rasterized graphics is used[1]. Due to screens being different resolutions, the problem happens

[1]`https://w3techs.com/technologies/overview/image_format`

with displaying rasterized graphics on a higher-resolution screen. We can proceed with this problem in two ways: just show the image in the original resolution which will be small or transform the image to a higher resolution (upscale it).

The problem with upscaling the raster image is the loss of quality due to missing pixel data. When upscaling the LR image of dimensions $(H \times W)$ with the factors of x and y, there is $(xH * yW) - (HW)$ number of unknown pixels in the image. For the sake of simplicity and performance comparison, in the rest of the paper, we will scale both height and width with the same factor so $x = y$. In that case, each pixel in the LR image gets upscaled into x pixels in the HR image.

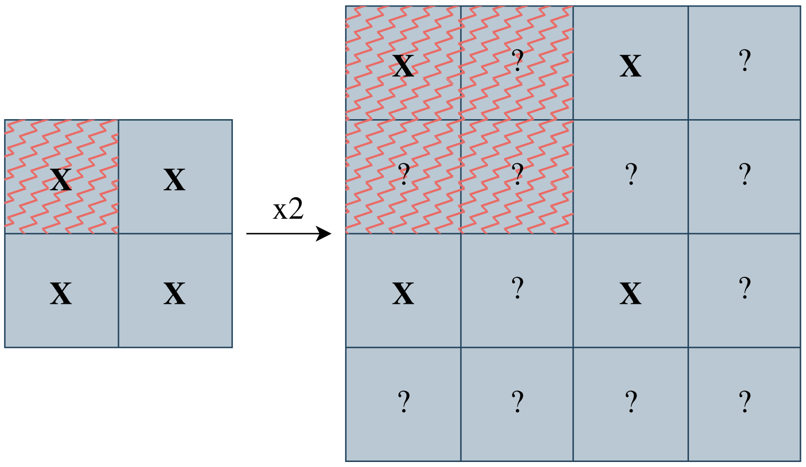


Figure 1.2: Example of missing pixels during x4 image upscale

This paper overviews two types of methods of image upscale: mathematical methods and modern (ML) methods. Mathematical methods mostly rely on interpolation which recreates pixel data using the given LR image pixel data. These methods are usually fast but tend to lack the quality of the generated image, especially if the factor of upscale is larger than two times. Modern methods rely on machine learning to predict the intensity of unknown pixels. This is the natural usage of artificial intelligence as we try to predict the distribution of pixels and recreate pixels from the given distribution. The downside of these methods is that they are slower than mathematical methods and sometimes rely heavily on the usage of GPU.

Because video is just a sequence of images in time (frames), the same principles apply to video upscale. Video upscale present another set of challenges. Using a naive approach upscaling a video can be implemented to iterate through all frames of the video, escape them and save them to a file. However, due to image upscale models focusing on image quality and not that much on the performance of the models, video upscale can take a lot of time. The most impressive challenge in video upscale then would be to optimize models so much that they can upscale video in real-time as the LR video is being buffered. These models are a sweet spot between the quality of the frame generated and the speed of inference that model is reproducing an HR frame.

2. Related work

Shi et al. (2016) showcases real-time image upscale research with the usage of a convolutional neural network (CNN). They demonstrate that deconvolution layers in CNN can perform upscale of the image at the later stages of the model architecture.

Galteri et al. (2019) in their paper Towards Real-Time Image Enhancement GANs explore increasing the size of the discriminator and decreasing the size of the generator. Both parts of a GAN are used in training but only the generator is used for inference as presented by Goodfellow et al. (2014). This allows for more efficient and faster generation of HR images. They also conclude that technology such as TesnorRT[1] may be used to further enhance inference performance which is explored in this paper.

Yang et al. (2022) review submissions of AIM 2022 Challenge on Super-Resolution of Compressed Image and Video[2] contestants. Authors showcase that research is mostly focused on image upscale. Not many teams competed in Task 2 of the challenge, which focused on upscaling video. The reason is that some of the state-of-the-art upscale method winning methods take up to 120 seconds to generate the HR image scaled up 4 times[3]. The metric used for comparing the speed of the methods in the competition was runtime per image (in seconds). As a baseline model bicubic interpolation was used which is further explained in Section 3.3.

[1]`https://developer.nvidia.com/tensorrt`
[2]`https://data.vision.ee.ethz.ch/cvl/aim22`
[3]`https://codalab.lisn.upsaclay.fr/competitions/5076`

3. Mathematical upscale methods

Most commonly used mathematical methods for image upscale are interpolations. Interpolation is a mathematical estimation method that allows us to approximate values of new data points based on existing discrete data points. In particular, interpolation methods estimate function values between known discrete data points. The data we are going to be estimating in raster images are pixel intensity values that are unknown. As functions can be N-dimensional, points can be interpolated in any dimension (Figure 8.1). Although for some raster formats (like RGB), pixel intensity values are 3-dimensional we can showcase them in a 2D matrix. So for image upscale, we are only interested in two-dimensional interpolations.

These methods are really fast but all of them have their specific advantages and disadvantages. In the following sections, we will describe the most used interpolations methods from the method of lowest computational complexity to the method of highest computation complexity.

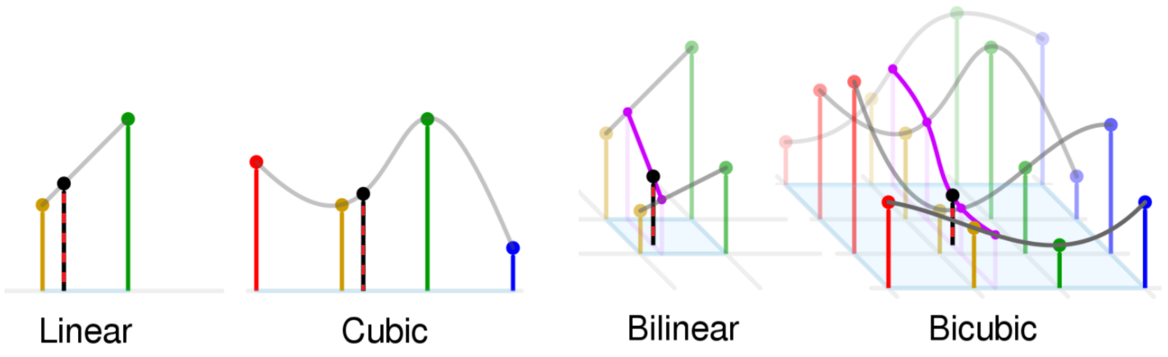


Figure 3.1: Example of one-dimensional (linear and cubic) and two-dimensional (bilinear and bicubic) and interpolation methods. The dashed black and red line represents an interpolated data point. Solid colored lines represent known data used for interpolation.

In the rest of the paper following mathematical notation will be used. We will represent images as two-dimensional matrices of size $(N \times N)$. For accessing and setting the pixel intensity on coordinates (x, y) we will use notation $I(x, y)$. I_{HR} is the intensity of pixel (x, y) on a high-resolution image whilst I_{LR} is the intensity of pixels on a low-resolution image. The top left corner of the image will have coordinates $(0, 0)$.

3.1. Nearest neighbor interpolation

The simplest image upscale method is the nearest neighbor interpolation. The intensity of the unknown pixel is calculated by directly copying the intensity value of the nearest neighbor pixel (Figure 3.2).

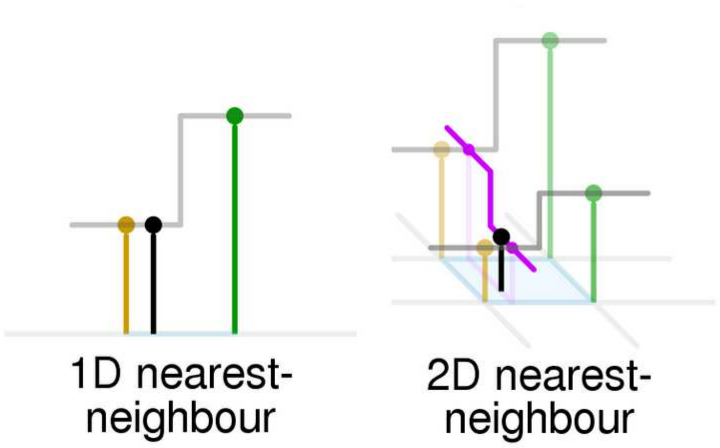


Figure 3.2: Example of nearest-neighbor interpolation on one-dimension and two-dimensions. The interpolated data point is displayed with a black line.

With (x, y) coordinates from the HR image, we can calculate neighboring pixels in the LR image using $factor = \frac{width-1}{(scale*width)-1}$. We can calculate the intensity of pixels in the HR image with the following equations:

$$I_{HR}(x, y) = I_{LR}(\lfloor x * factor \rceil, \lfloor y * factor \rceil) \tag{3.1}$$

where $\lfloor x \rceil$ is the notation of rounding decimal numbers to the nearest integer value (equivalent to `round()` functions).

This method preserves details in the image but loses object sharpness. For an image with high contrast (black text on white background), we have an unwanted stairway-like object border (Picture). Because of mentioned disadvantages, nearest neighbor interpolation finds adequate usage in upscaling bit-art icons and pixel-art (Picture 3.3.)

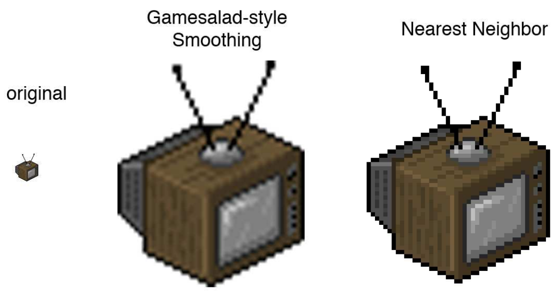


Figure 3.3: Example of nearest-neighbor interpolation providing pleasing results pixel-art images.

3.2. Bilinear interpolation

Bilinear interpolation method is based on linear approximation between two data points in one dimension and four data points in two dimensions.

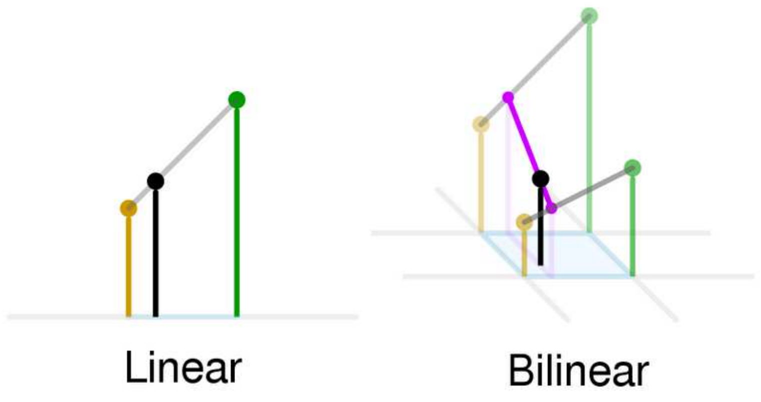


Figure 3.4: Example of linear (one-dimensional) and bilinear (two-dimensional) interpolations. The interpolated data point is displayed with a black line.

Four neighboring pixels (x_1, y_1), (x_1, y_2), (x_2, y_1), and (x_2, y_2) are needed to calculate the intensity of the pixel on the HR image. With (x, y) coordinates from the HR image, we can calculate neighboring pixels in the LR image using $factor = \frac{width-1}{(scale*width)-1}$, $x_1 = \lfloor x * factor \rfloor$, $x_2 = \lfloor x * factor \rfloor + 1$, $y_1 = \lfloor y * factor \rfloor$, $y_2 = \lfloor y * factor \rfloor + 1$. We can calculate them using the following:

$$\begin{aligned} w_x &= x * factor \\ w_y &= y * factor \\ I_{HR}(x, y) &= (1 - w_x) * (1 - w_y) * I_{LR}(x_1, y_1) \\ &\quad + w_x * (1 - w_y) * I_{LR}(x_1, y_2) \\ &\quad + (1 - w_x) * w_y * I_{LR}(x_2, y_1) \\ &\quad + w_x * w_y * I_{LR}(x_2, y_2) \end{aligned} \tag{3.2}$$

In comparison to nearest neighbor interpolation, bilinear interpolation preserves the smoothness of the object lines. On the other hand detail of the object may not be preserved as well using nearest neighbor interpolation. Images that work the best with bilinear interpolation are images with smooth pixel intesnity transitions and already blurred images. When an image is already blurred at the beginning of the upscale bilinear interpolation might be an interpolation method of choice as it is faster then bicubic interpolation.

3.3. Bicubic interpolation

Bicubic interpolation is mathematically and computationally more complete than bilinear interpolation. In the case of interpolating a point in one dimension, we would need four known data points. For interpolation, a point in two dimensions sixteen known data points are needed (Figure 3.5).

There are several ways to implement bicubic interpolations. One of the implementations methods is using B-splines.

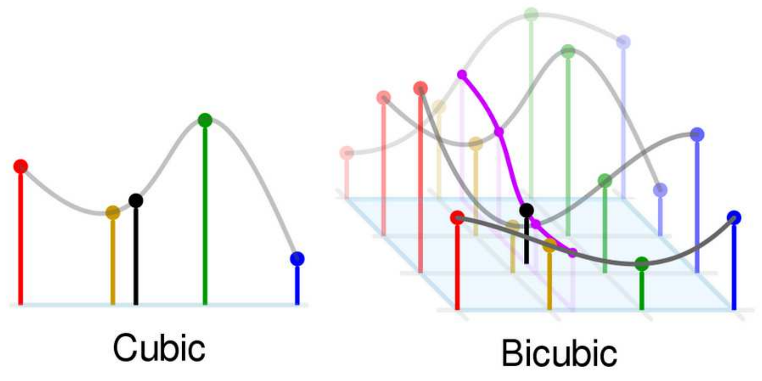


Figure 3.5: Example of cubic (one-dimensional) and bicubic (two-dimensional) interpolations. The interpolated data point is displayed with a black line.

This method is used as a baseline when comparing the quality of modern upscale methods to mathematical upscale methods. It is also the upscale method of choice in most image viewer and video player software offering an adequate trade-off between speed and image quality in comparison to other interpolation methods.

3.4. Lánczos interpolation

Named after Hungarian mathematician Cornelius Lánczos this interpolation method is heavily relied on normalized $sinc$ windowed mathematical function which is defined as $sinc(x) = \frac{\sin \pi x}{\pi x}$. Functions are defined as windowed when it evaluates to 0 outside a predefined interval. Normalized functions are functions that integrate to 1 on their whole domain.

Lánczos kernel function is defined with normalized windowed $sinc(x)$ function as such:

$$L(x) = \begin{cases} sinc(\pi x)sinc(\frac{\pi x}{a}), & \text{if } -a < x < a \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

The kernel is used to sample pixel intesnity values from its given pixel window in original image.

$$
\begin{aligned}
factor &= \frac{width - 1}{(scale * width) - 1} \\
x_{LR} &= x * factor \\
y_{LR} &= y * factor \\
I_{HR}(x, y) &= \sum_{i=\lfloor x_{LR} \rfloor - a + 1}^{\lfloor x_{LR} \rfloor + a} \sum_{j=\lfloor y_{LR} \rfloor - a + 1}^{\lfloor y_{LR} \rfloor + a} I_{LR}(i, j) L(x_{LR} - i) L(y_{LR} - j)
\end{aligned}
\tag{3.4}
$$

The size of the Lánczos kernel window is defined with the hyperparameter a. The quality of the image may improve with an increase in a because the Kernel window for interpolation will be bigger which may lead to smoother image details. Kernel size is determined with the equation $K = 2a + 1$ for 2D images. Lánczos interpolation upscale with $a = 2$ already has a bigger kernel size (5×5) than bicubic interpolation (4×4).

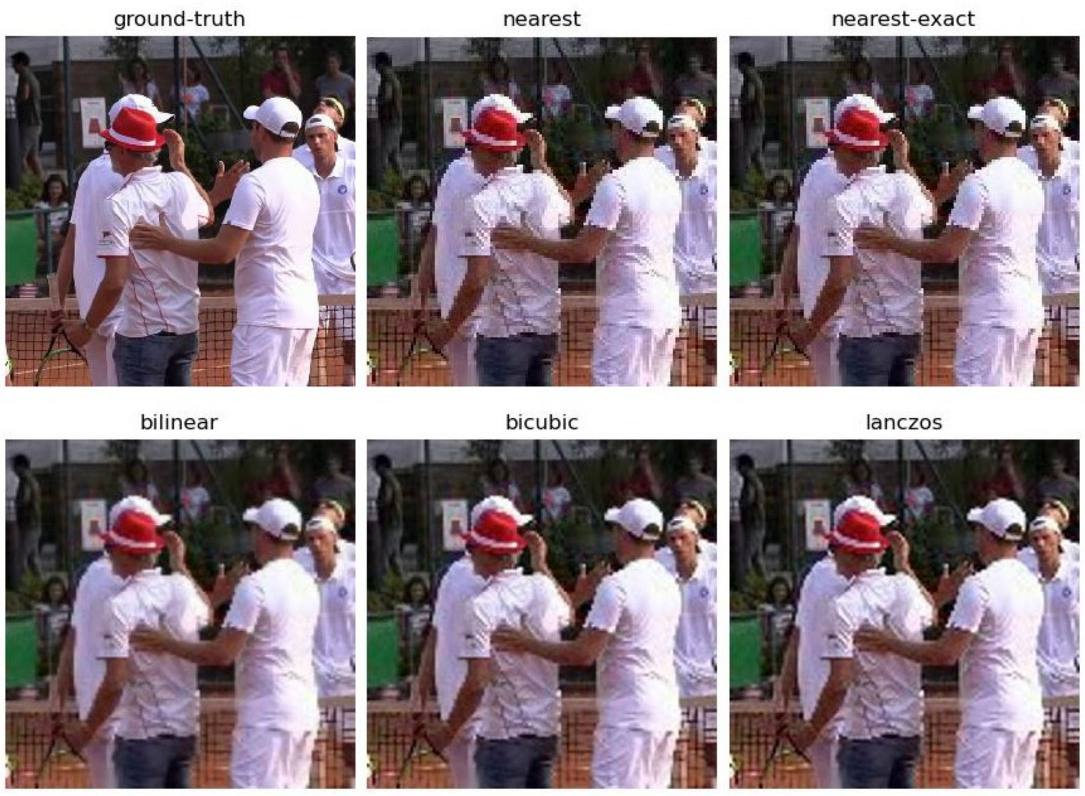


Figure 3.6: Comparison of different interpolation methods for upscaling from 128×128 to 512×512. Particularly all methods struggle with fine sharp details like the net of the tennis court.

When upscaling hyperparameter a is going to be optimized to suit upscale quality needs. As we increase the hyperparameter, upscale algorithms' computational complexity is also decreasing proportionally. It is important to take notice that value a can be a real number but by default, we select an integer anywhere in the range from two to four.

3.5. Mitchell-Netravali filter

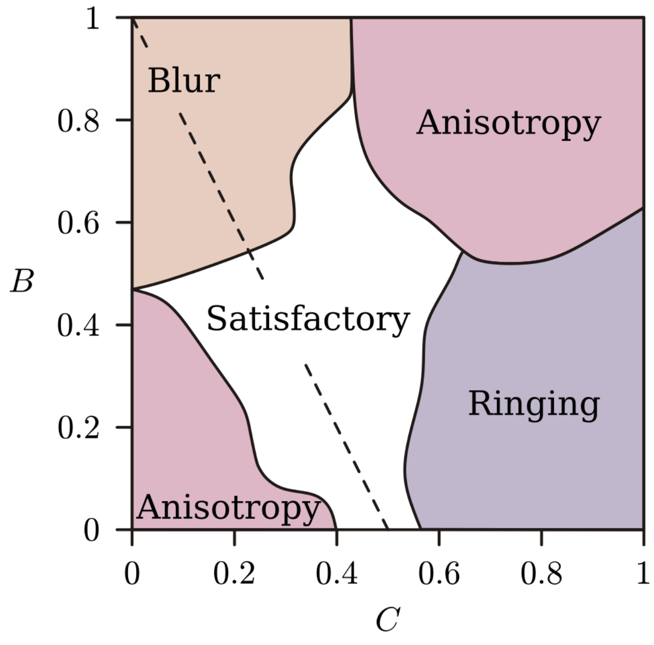
In addition to upscale we can use some popular interpolation filters after the image is up-scaled to improve upon the quality of the HR image. The following filters are applied after upscaling the image with methods mentioned in the previous sections.

As defined by Mitchell and Netravali (1988) Mitchell-Netravali filter is usually compared with bicubic interpolation. Image editing programs usually name this combination a bicubic filter. We define weights of the filter with the following equation:

$$\begin{aligned} w_x(x) &= \frac{1}{6}\left[(12-9B-6C)\,|x|^3 + (-18+12B+6C)\,|x|^2 + (6-2B)\right] \\ w_y(y) &= \frac{1}{6}\left[(12-9B-6C)\,|y|^3 + (-18+12B+6C)\,|y|^2 + (6-2B)\right] \end{aligned} \tag{3.5}$$

where x and y are distances from the original to neighboring pixels. Getting neighboring pixels of the already upscaled image and modifying the pixel multiplying its intensity with w_x, and w_y.

The filter has two parameters B and C which can be modified to suppress blurring and ringing artifacts. The combinations of (B, C) hyperparameter can form specific cubic splines:



(B, C)	Splines
(1, 0)	B-spline
(0, x)	Cardinal splines
(0, 0.5)	Catmull-Rom spline
(1/3, 1/3)	Mitchell-Netravali

Figure 3.7: Satisfactory combinations of (B, C) values based on Mitchell and Netravali (1988). The research was conducted on test subjects that are image processing researchers.

3.6. Raster vectorization

Underlying concept of the following method is not to work with raster images but rather to convert them into vector images. This method is known as raster vectorization (or sometimes raster-to-vector conversion) (Figure 3.8). The fundamental objective of vectorization is to accurately trace or recreate the raster image as a collection of vector primitives. Images can be vectorized via manual and automatic methods.

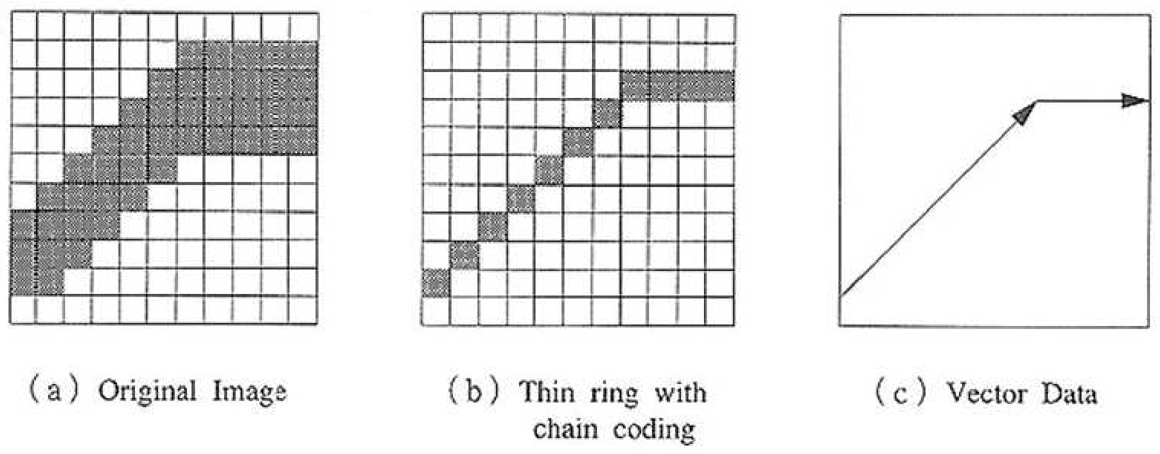


Figure 3.8: Line raster vectorization example

[0]https://videoprocessing.ai/metrics

4. Modern upscale methods

Deep learning models are well-suited for image upscaling tasks due to their ability to capture and recreate image distributions. When scaling up images, the goal is to generate high-resolution outputs that align with the underlying distribution of the original images. Deep learning models excel in this regard by learning patterns and features, enabling them to produce realistic and visually appealing upscaled images.

4.1. Convolutional Neural Network approaches

In Convolutional Neural Network (CNN) approaches, it is common to apply mathematical upscaling methods to the image at the beginning of the network and then combine the output of the network with the upscaled image to enhance its quality. By incorporating deconvolutional layers in the final layers of the network, the image can be upscaled within the model itself. This eliminates the additional computational steps required for traditional method upscaling.

Fast Super-Resolution Convolutional Neural Network

Fast Super-Resolution Convolutional Neural Network (FSRCNN) developed by Dong et al. (2016) in mind of real-time image upscale (and subsequently video upscale).

Architecture consists of two main components: the feature extraction network and the deconvolution network. The feature extraction network learns to extract meaningful vector features from the LR image. These features are then processed and refined by the deconvolution network, which gradually upscales the image while recovering fine details. FSRCNN contains a sub-pixel convolutional layer, which rearranges the learned features in the upscaling process. This layer enhances the final HR image by redistributing the information across different sub-pixel positions, increasing the resolution and sharpness.

This model is the optimization of already existing Super-Resolution Convolutional Neural Networks (SRCNN) as shown in Picture 4.1. It improves upon SCRNN in the following aspects:

1. FSRCNN processes the LR image as input without bicubic interpolation at the start. Often in CNN approaches image is firstly upscaled and then via convolutions and residual blocks additional noise is added to the bicubic HR image. Instead of this approach, the deconvolution layer is introduced to perform upscaling to HR image dimensions

2. Non-linear mappings SRCNN are replaced multiple steps in FSRCNN (specifically shrinking, mapping, and expanding)

3. Filter sizes are decreased and architecture is deeper in structure

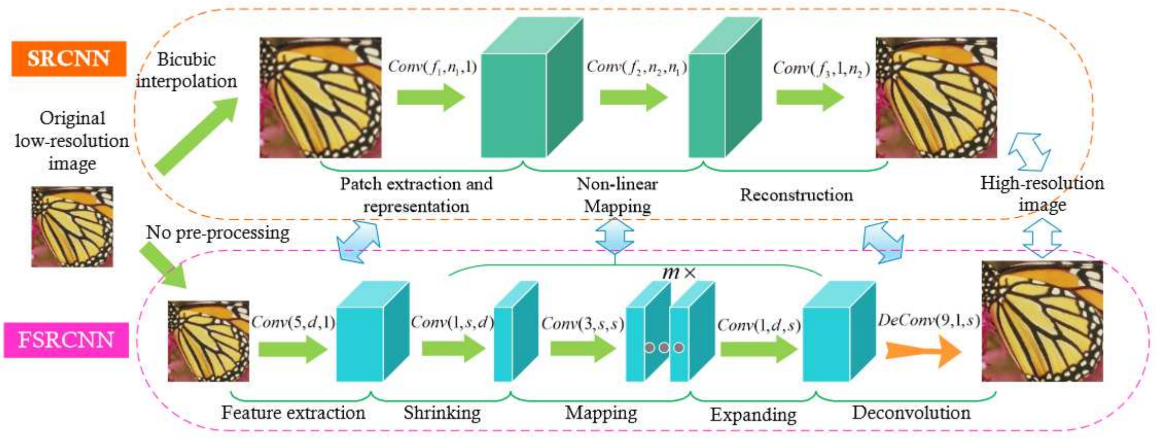


Figure 4.1: Changes of FSRCNN to SRCNN to optimize inference process

4.2. Generative Adversarial Models approaches

Generative Adversarial Models (GANs) are composed of two components: Discriminator and the Generator (Figure 4.2). Components of the GANs are trained with a minimax game where the Generator aims to deceive the Discriminator into misclassifying the generated images. The objective is for the Generator to produce synthetic data that closely resembles real data, while the Discriminator strives to accurately classify between real and fake images. As training progresses, the model ideally converges to a point where the Discriminator is uncertain, classifying both real and generated images with a probability of $\frac{1}{2}$, indicating an equal level of uncertainty in determining the authenticity of the images. In the inference, only the Generator part of the GAN is used to generate new high-resolution images.

GANs are capable of generating high-quality, realistic data that closely resembles real data distribution. They have been successfully applied in various domains, including image

[1]`https://sthalles.github.io/intro-to-gans/`

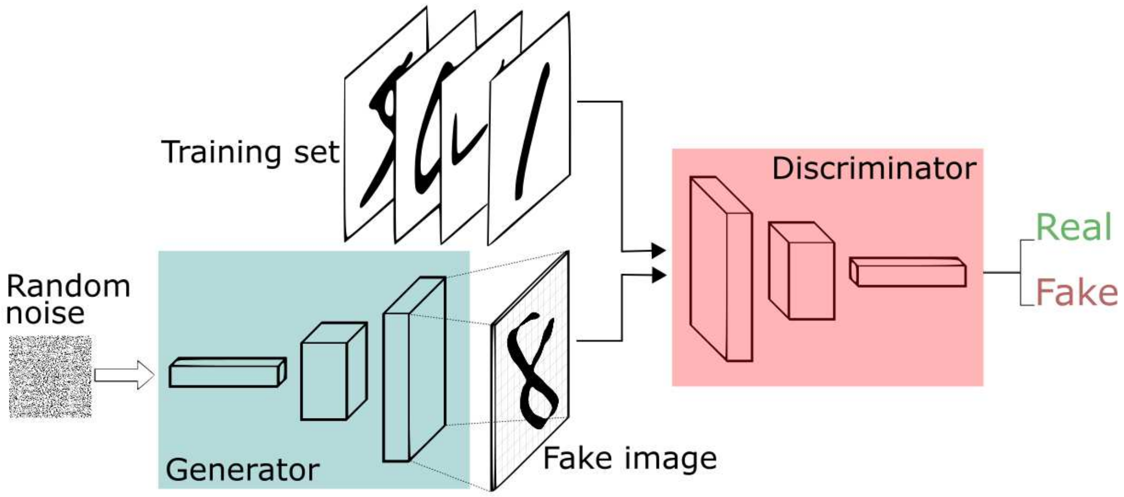


Figure 4.2: Two models used withing GAN. Generator and Discriminator[1].

generation, text synthesis, and even video generation. However, it is worth noting that some GAN generators can be large in size, leading to slower inference times. This is why they need to be heavily optimized before they are used for real-time upscale.

Fast Super-Resolution Generative Adversarial Network

Fast-SRGAN[2] is a modification to Ledig et al. (2017) Super-Resolution Generative Adversarial Network (SRGAN) for real-time image inference capabilities.

SRGAN is a large GAN network architecture that is used for upscaling images up to 8 times with a focus on the quality of the image generated. It was trained on the DIV2K dataset. Although giving state-of-the-art results at the time, it was really slow and only capable of single-image super-resolution image upscale.

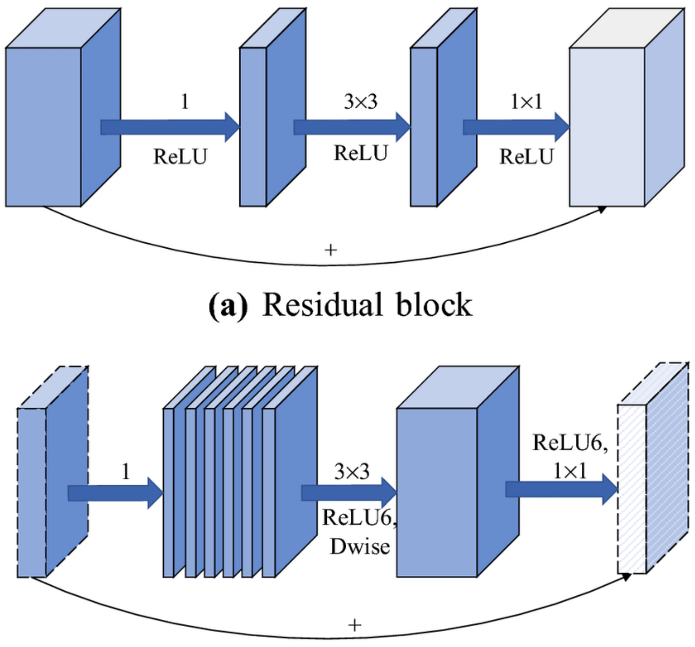
The training process of SRGAN involves a two-step procedure. Initially, the generator network is trained using a mean squared error (MSE) loss function to minimize the difference between the generated high-resolution images and the corresponding real high-resolution images. This process is a standard training procedure for GANs. The key innovation of SRGAN lies in the perceptual loss function, which combines the adversarial loss with a content loss based on high-level feature representations extracted from a pre-trained deep neural network (such as VGGNet). This perceptual loss encourages the generator to not only produce visually plausible images but also capture the perceptual details and structure of the high-resolution ground truth images.

To optimize GANs for real-time inference there are a few key manual architecture changes that can be applied:

[2]`https://github.com/HasnainRaz/Fast-SRGAN`

1. Inference of GANs only depends on the complexity of the Generator. To some extent, Generator can be decreased in size and Discriminator can be increased. This allows us to maximize the capabilities of the original model but speed up the inference immensely.

2. To speed up operations inverted residual blocks are employed instead of normal residual blocks for parameter efficiency. Inverted residual blocks employ a more efficient design by introducing 1×1 convolutional layers to reduce the dimensionality of the input feature /maps before applying 3×3 convolutions (Picture 4.3). This common optimization is often used with CNNs when deploying them to mobile platforms.



(a) Residual block

(b) Inverted residual block

Figure 4.3: Inverted residual block as common optimization in mobile deep-learning networks.

Fast-SRGAN claims to upscale images of dimension 384×384 to 1536×1536 (4 times) in 68 ms on average (or in a maximum of 15 FPS) on GTX 1080 GPU. Usually, manual optimization is hard and we may miss some simple model architecture changes. Applying ONNX (as explained in Section 5.2) optimization techniques to the Fast-SRGAN model can potentially unlock additional gains in performance. By optimizing the model structure and leveraging the capabilities of ONNX, it becomes possible to streamline computations, reduce memory requirements, and take advantage of hardware-specific optimizations. This optimization process enables the model to achieve even higher performance levels, delivering faster image upscaling capabilities while maintaining image upscale quality.

5. Real-time video upscale

In order to achieve real-time video upscaling, we need to have a method that is optimized for inference. The slow-performing method leads to a jittery and non-fluid video. Metrics that will be important in monitoring model inference speed are FPS (frames per second) and RPI (runtime per image). It is desirable that the upper bound for runtime per image metric is 41.67 milliseconds. Further elaboration on these metrics can be found in Section 7.3.

We will use GPU-optimised mathematical upscale methods. In the context of modern upscale models, they will be saved and optimized using the ONNX format.

5.1. Computer Unified Device Architecture

To speed up both mathematical and modern upscale methods we are going to use Computer Unified Device Architecture (CUDA) platform. This platform is developed by NVIDIA and allows developers to execute high-performance code in parallel on NVIDIA GPUs. Although the CUDA model is based on C++, Python implementations are also available. Most of the image manipulation and deep learning frameworks enable us to use CUDA to speed up execution that may be slower on CPUs.

All mathematical methods are implemented and CUDA optimized in most of the image processing and deep learning frameworks. Python code for CUDA interpolation methods is following:

```python
import torch
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
torch.cuda.set_device(device)

output_size = (input_size * SCALE, input_size * SCALE)
interpolation_methods = [transforms.InterpolationMode.NEAREST,
    transforms.InterpolationMode.BILINEAR,
    transforms.InterpolationMode.BICUBIC,
    transforms.InterpolationMode.LANCZOS,
    ...
]
```

```
for method in interpolation_methods:
    output_image = transforms.Resize(size=output_size,
     ↪   interpolation=method).forward(image)
```

It is important to note that CUDA will not be utilized if the environment is not set up correctly. The code snippet above will still work as we are utilizing `torch.cuda.is_available()` function.

When CUDA is enabled in deep learning frameworks, it ensures that the model will utilize the GPU for both training and inference. However, to optimize the model itself, manual adjustments or optimizations for universal model formats like ONNX are required.

5.2. Open Neural Network Exchange

Open Neural Network Exchange (ONNX)[1] is an open format for representing deep learning models. This format type allows us to universally use it across different machine-learning frameworks 5.2. Furthermore, ONNX models can be further optimized to improve execution speed, memory usage, and overall efficiency. The ONNX project is an open-source project supported and advanced by a community of researchers, developers, and industry professionals.

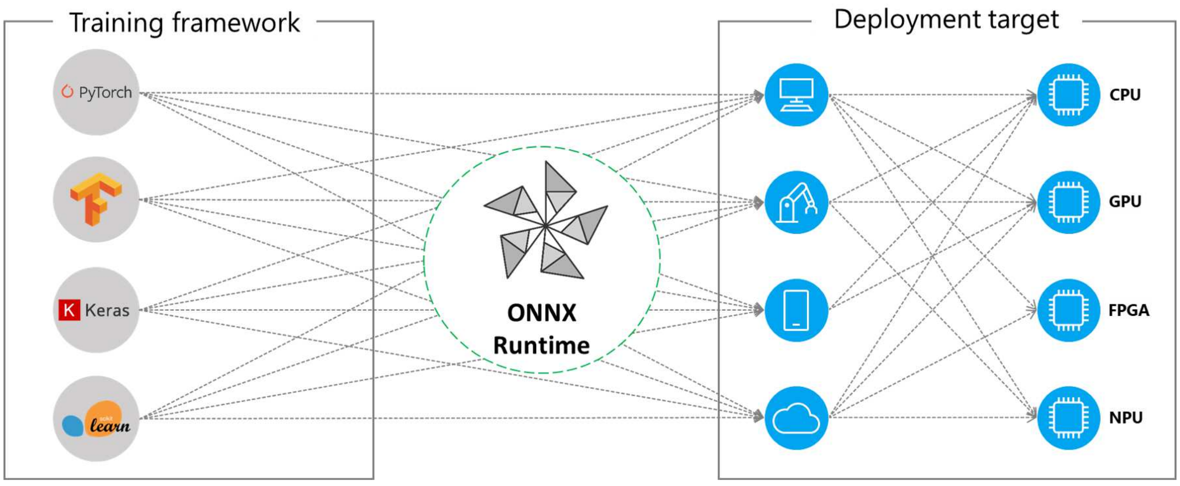


Figure 5.1: Onnx runtime which allows interoperability between deep learning frameworks.

Except for interoperability between frameworks, ONNX also allows for optimization for specific hardware. ONNX model format is represented as a graph that allows for the following usage.

[1]`https://onnx.ai/`

5.2.1. Optimization

There are several optimizations that can be done to the already existing model to further improve model inference. Python package ‘tf2onnx‘ provides us with the built-in set of functions for loading and optimizing TensorFlow, PyTorch, and Keras models. The following operations are examples of actions taken to further optimize the model:

Data type optimization

Model is analyzed and optimized to use lower precision data types for operations that can tolerate reduced precision. This optimization is specified on memory usage but it can speed up the computations.

Shape inference

All tensor shapes in the model must be resolved to static dimensions. This leads to better usage of memory and speeds up the inference because of a decrease in function calls

Dead code elimination

Operations and subgraphs that are not required for producing output are eliminated. This optimization specifically speeds up the computations as it is eliminating unnecessary computations.

Operator fusion

Multiple consecutive operations are merged into a single operation which can be then executed more efficiently. The number of intermediate tensors is also reduced so this optimization speed up the computations and is lowering the memory usage.

Constant folding

Model is analyzed for the usage of constant values. Subgraphs in the model where the inputs are constants are then replaced with pre-computed constant values. This optimization leads to replacement in computations during inference.

5.2.2. Vizualization

Because of the ONNX graph structure it is easy to develop a visualization tool that will be able to visualize the model in a graph structure from a file. One of those tools is Netron[2]

[2] `https://netron.app/`

which allows us to view the model, input, and output tensor shapes, exporting graph visual representation as an image, etc.

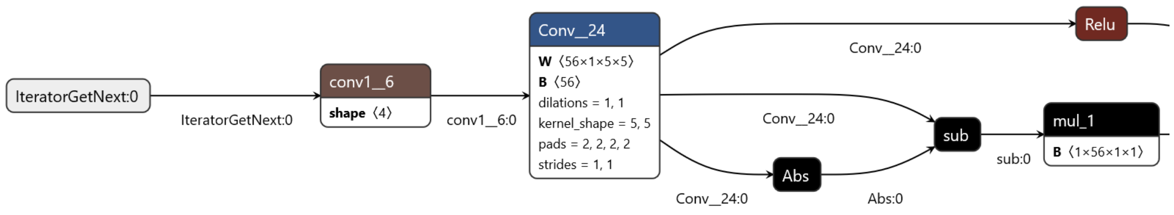


Figure 5.2: Start of the graph for FSRCNN model visualization on Netron portal.

5.2.3. Backend-agnostic

By being backend agnostic, ONNX enables model deployment and execution across a wide range of hardware and software configurations. It allows for flexibility, portability, and efficient execution of ONNX models on different inference engines and hardware platforms, making it a versatile choice for deploying machine learning models.

6. Dataset

Many problems may be encountered while selecting the dataset for image upscaling tasks. The best way to select a dataset is to target a specific thematic of images for upscale. AI image upscale model will be able to upscale images of the same variety the best if it is learned on the dataset for that specific variety. The clean dataset may overfit the model, which can only upscale images without artifacts. Additionally, incorporating data diversity and augmentation techniques can help enhance the model's generalization capabilities and enable it to handle real-world images effectively, even in the presence of various artifacts and imperfections as demonstrated by Shorten and Khoshgoftaar (2019).

6.1. Large-scale Diverse Video Dataset

Large-scale Diverse Video (LDV) Dataset was established first for usage in NTIRE (New Trends in Image Restoration and Enhancement) Workshop 2021[1] by Yang and Timofte (2021). The newest version of the dataset is LDV3[2] which contains 365 videos and it is an update to version 2.0 Yang et al. (2022). The dataset was split with an 80-10-10 ratio split into the train, test, and validation subsets. After the split train subset contains 292 videos, the test subset contains 37 videos, and the validation subset contains 36 videos.

The purpose of the dataset is to have a large collection of videos from diverse categories with various frame rates and different artifacts. As seen in Figure 6.1 dataset contains a large variety of nature, cityscape, people, animals, etc. To showcase the diversity of the dataset we extracted a 4x4 grid of random frames from validation videos using the following Python code using the OpenCV library:

```python
for _ in range(num_rand_images):
    rand_frame = random.randint(0, total_frames)
    vidcap_ground_truth.set(cv2.CAP_PROP_POS_FRAMES, rand_frame)
    success, image = vidcap_ground_truth.read()
    if success:
        cv2.imwrite(output_file_path, image)
```

[1]https://data.vision.ee.ethz.ch/cvl/ntire21/

[2]https://data.vision.ee.ethz.ch/reyang/AIM2022/LDV3.zip

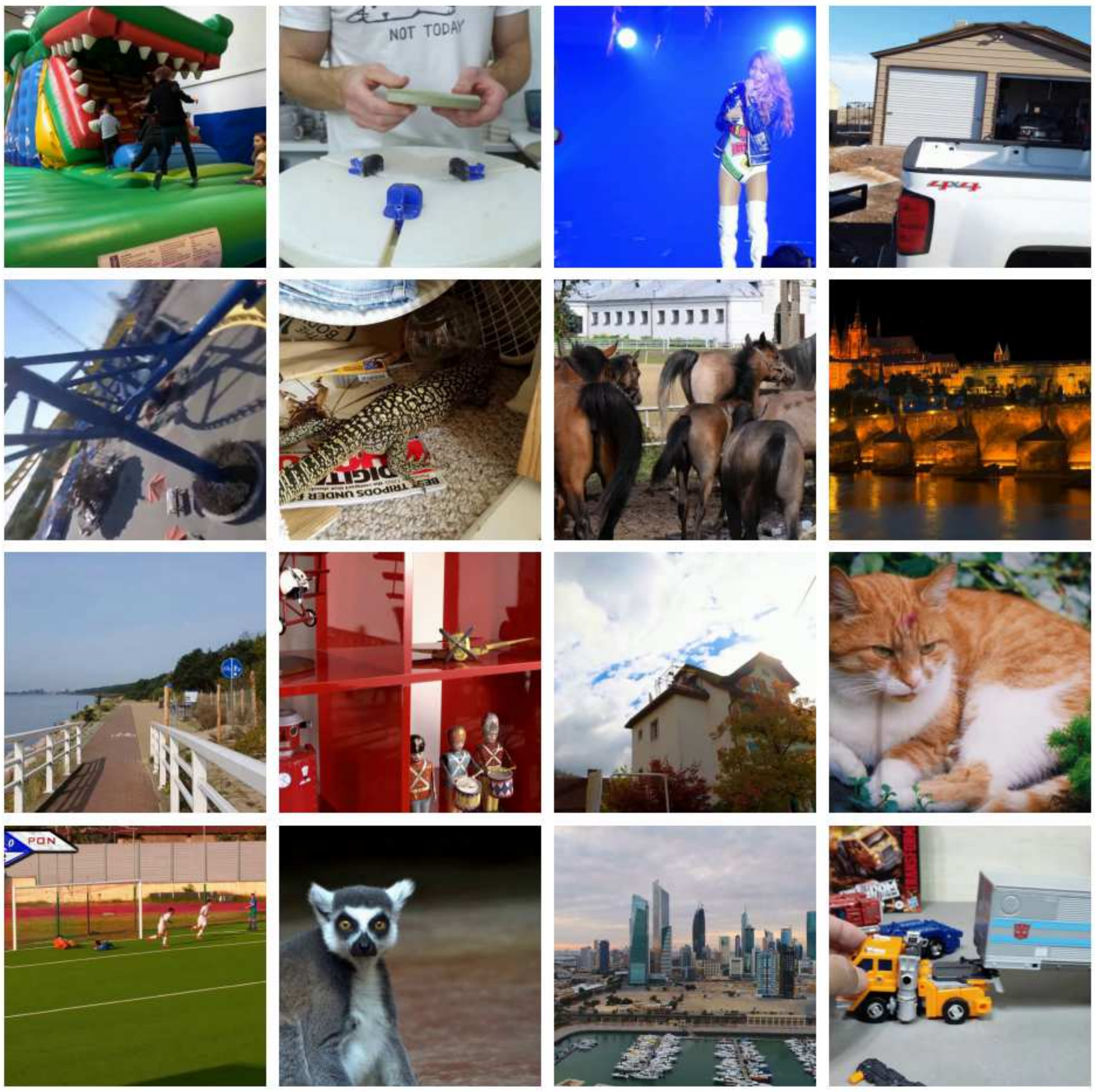


Figure 6.1: LDV3 dataset random image sample from validation part of dataset

6.2. DIV2K Dataset

DIVerse 2K resolution high-quality images (DIV2K) is most commonly used for the task of image super-resolution. Most of the pre-trained models we are going to use are trained on that dataset. The dataset consists of a collection of 2K (meaning images with longer dimensions is at least 2000 pixels) high-quality images. It consists of synthetic and real-world images to provide balance in the dataset for all image characteristics.

As we are working on real-time video upscale DIV2K is not suited as much as LDV3 for that specific task. Video frames have specific video artifacts, such as motion blur or shaky camera movements, that cannot be found on high-resolution image datasets. This is the reason why it is a good idea to fine-tune pre-trained models so that they have additional inputs from the video dataset.

7. Metrics

Calculating the quality of the image upscale is challenging. Usual metrics used in machine learning can usually often fall short, not accounting for image artifacts such as blur, ringing, and overall loss of details. For more plausible metrics, it is important to consider local image patches and the image as a whole.

While metrics can be useful to provide objective measures, best practices show that subjective evaluation of image quality is crucial as discussed by Afnan et al. (2023). Human perception of color is something objective metrics may not yet fully comprehend. More advanced metrics have been developed with the usage of simple neural networks that determine image similarity based on image vector representations and subjective input.

7.1. Full-reference quality metrics

If we have access to the GT image that we want to be most similar to we can use full-reference quality metrics. They allow us to compare the quality of the HR image directly to the GT image. Full-reference metrics can provide pixel-level or structural-level information about image artifacts. An important disadvantage of full-reference quality metrics is that it is hard to evaluate the quality in real-time upscale as we need to have constant access to reference video which may not be available.

Mean square-root error

Mean square-root error (MSE) calculates the mean square distance between two vectors. In our case vectors are the pixel intensity of the pixels. If we have two images GT (ground truth) and HR (high-resolution image) of the same dimensions (height, width), we can calculate MSE between them with the following equation:

$$MSE(GT, HR) = \frac{1}{height * width} \sum_{i=0}^{height-1} \sum_{j=0}^{width-1} [HR(i,j) - GT(i,j)]^2 \tag{7.1}$$

This metric can be really deceiving as it only compares pixel-to-pixel relations between two images. Neighboring pixels are not taken into consideration although they bring a lot of context to the image as a whole.

Peak-signal-to-noise ratio

Peak-signal-to-noise ratio (PSNR) is a simple engineering metric used for the comparison of signals. If we already calculated $MSE(GT, HR)$, PSNR can be easily calculated as follows:

$$PSNR(GT, HR) = 10 * \log_{10} \frac{{MAX_I}^2}{MSE(GT, HR)} \qquad [dB] \tag{7.2}$$

where ${MAX_I}^2$ is the squared maximum intensity of the signal (in our case picture). Maximum intensity can be different for a lot of picture formats. For example, the maximum pixel intensity of a pure RGB image is 255 but it can also be 1 if the picture is normalized. Images with higher PSNR have lower visual qualities. Ideally, we want PSNR to be small.

MSE and PSNR have critical flaws in the evaluation of image similarity. Images with completely different visual qualities may have equal MSE and PSNR scores. This is due to a lack of ability to measure structural distortions (Figure 7.1). Because of this important flaw, MSE and PSNR are not as highly regarded as measures for image upscale.

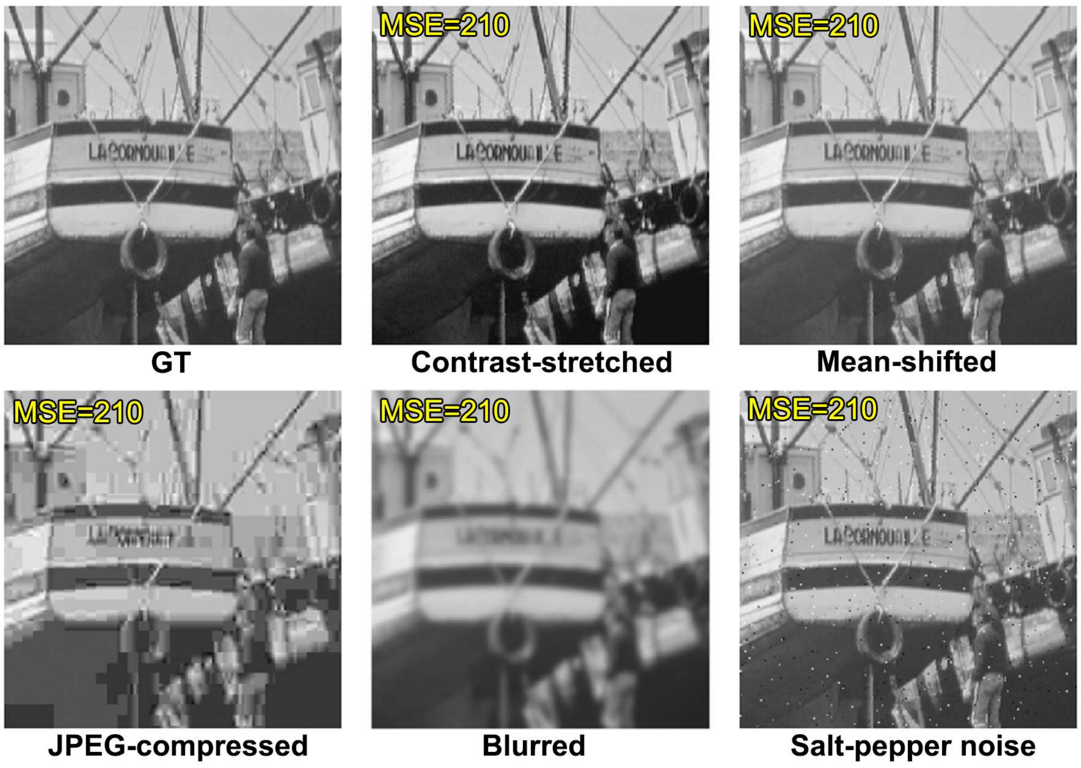


Figure 7.1: Images with different visual qualities but same MSE values[1]

[1]`https://videoprocessing.ai/metrics`

Structural Similarity

Introduced in Wang et al. (2004), structural similarity (SSIM) improves upon flaws of MSE and PSNR. SSIM can have values in the range $[-1, 1]$ where: 1 is the value for two completely similar images, 0 for images that are not similar and -1 is the value for two completely dissimilar images. Calculating SSIM is more computationally complex than calculating MSE and PSNR. The following equation highlights the most important terms in calculating SSIM:

$$\begin{aligned} SSIM(GT, HR) &= l(GT, HR)^{\alpha} * c(GT, HR)^{\beta} * s(GT, HR)^{\gamma} \\ l(GT, HR) &= \frac{2\mu_x \mu_y + c_1}{\mu_x^2 + \mu_y^2 + c_1} \\ c(GT, HR) &= \frac{2\sigma_x \sigma_y + c_2}{\sigma_x^2 + \sigma_y^2 + c_2} \\ s(GT, HR) &= \frac{\sigma_{xy} + c_3}{\sigma_x \sigma_y + c_3} \end{aligned} \tag{7.3}$$

Components of the following Equation 7.3 are: luminance(l), contrast(c), structure(s), and constants c_i are used for stabilizing division. For calculating this metric pixel sample mean, variance, and covariance between images. Usually constants are defined as $c_1 = (k_1 L)^2$, $c_2 = (k_2 HvL)^2$, $c_3 = \frac{c_2}{2}$ where L is dynamic range of pixel-values, $k_1 = 0.01$ and $k_2 = 0.03$. If we set weights α, β and γ to 1 formula resolves into:

$$SSIM(G, HR) = \frac{(2\mu_x \mu_y + c_1)(2\sigma_x \sigma_y + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{7.4}$$

We can conclude from Equation 7.4 that whole image statistics are used to calculate error rather than comparing two images on a pixel-to-pixel basis. SSIM is more robust in comparing images with artifacts than MSE and PSNR as seen in Figure 7.2.

This method, however, has disadvantages. Particularly SSIM cannot detect image artifacts such as color shifts and localized artifacts. Also, big changes in the brightness of the image can result in small changes in SSIM value as seen in Figure 7.3.

Learned Perceptual Image Patch Similarity

With the usage of deep neural networks such as VGG and AlexNet Zhang et al. (2018) created the Learned Perceptual Image Patch Similarity (LPIPS) metric which focuses on high-level visual features and compares them. This method is based on feature representations of the images and compares the perceptually. Such metric shows that it aligns with human perception of images better than other methods (as shown in Figure7.4).

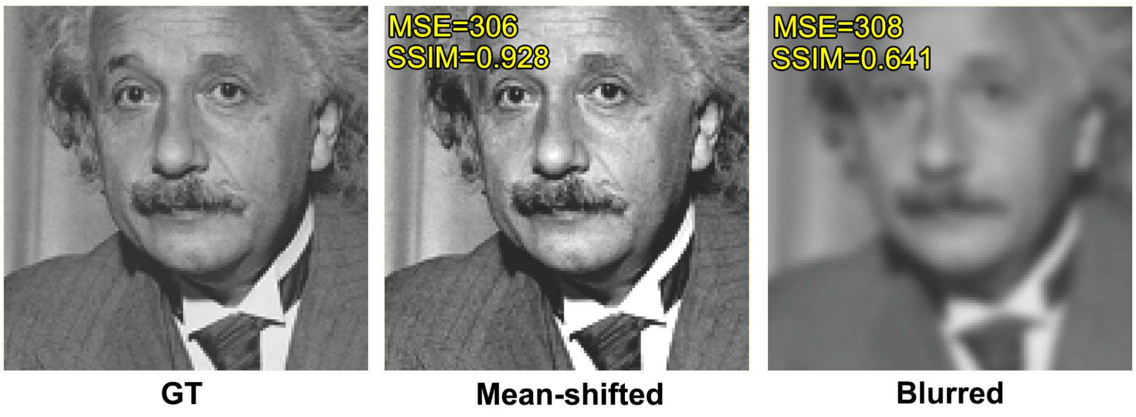


Figure 7.2: Example of SSIM taking into consideration blur inside of images

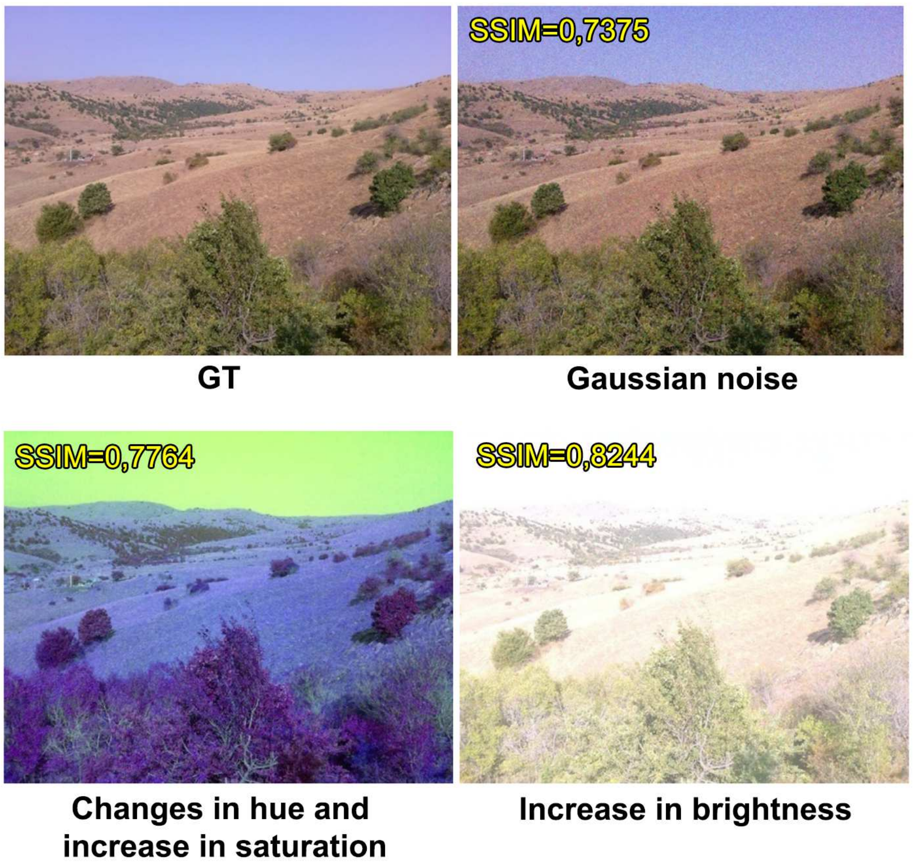


Figure 7.3: Example of similar SSIM values but with the different visual quality of the images

However, it must be noted that LPIPS also has its disadvantages. As the metric is based on learned representations it may not be universally applicable to all image content and image artifacts. Pretrained models can be fine-tuned to fit more various training data, but then it is important that a new model is publicly available so that results may be comparable. If used improperly model behind the metric can be over-trained and may lead to faulty but favorable

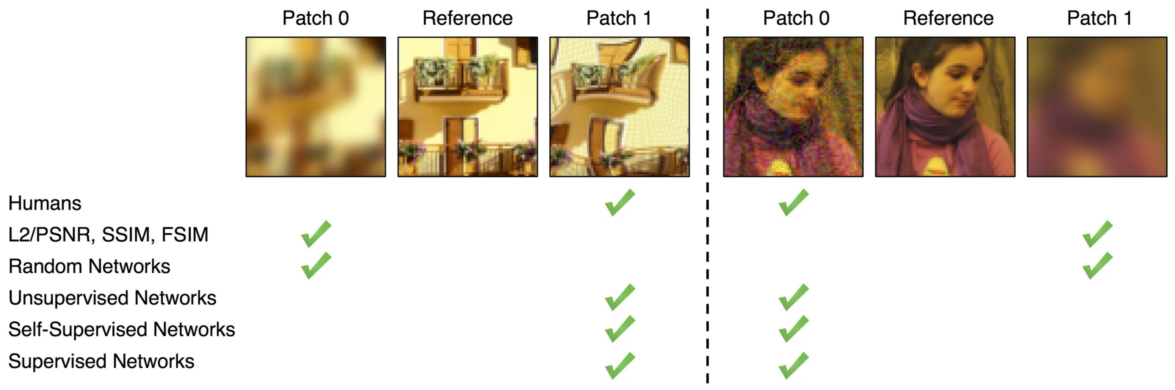


Figure 7.4: Preference of L2, PSNR, SSIM, LPIPS metrics based on image artifacts[2]

results.

7.2. No-reference quality metrics

Advancements in image compression and upscaling led to the development of non-reference metrics which do not require GT images to assess the quality of the image. The disadvantages of no-reference quality metrics are the advantages of full-reference quality metrics. No-reference metrics can not provide information about image artifacts whilst, on the other hand, we do not need access to GT for metric evaluation in real-time.

Blind/Referenceless Image Spatial Quality Evaluator

Blind/Referenceless Image Spatial Quality Evaluator (BRISQUE) is defined by Mittal et al. (2012). The basis for the metric is a premise that all images have similar statistical qualities and that all deviations from image statistics may showcase image artifacts. To evaluate the quality of an image, BRISQUE employs a pre-trained Support Vector Regressor (SVR) model that has been trained on the LIVE IQA database introduced by Sheikh et al. (2006). BRISQUE provides quality scores ranging from 0 to 100, where lower scores tend to indicate higher image quality.

Naturalness Image Quality Evaluator

Naturalness Image Quality Evaluator (NIQE) is defined by Mittal et al. (2013). This metric is similar to BRISQUE but it is designed specifically for natural images, which typically exhibit statistical regularities inherent in natural scene statistics. NIQE's quality scores range from 0 to infinity, with lower scores indicating higher image quality.

[2]`https://richzhang.github.io/PerceptualSimilarity/`

Perception-based Image Quality Evaluator

Perception-based Image Quality Evaluator (PIQE) is defined by Venkatanath and S. (2015). It is similar to NIQE but less computationally efficient. With the usage of machine learning techniques, this metric is trained on datasets that contain both pristine reference images and corresponding distorted images, along with subjective quality ratings provided by human observers. The scoring range of PIQE can vary depending on the specific implementation and dataset used for training, but it typically ranges between 0 and 100, with lower scores indicating higher image quality.

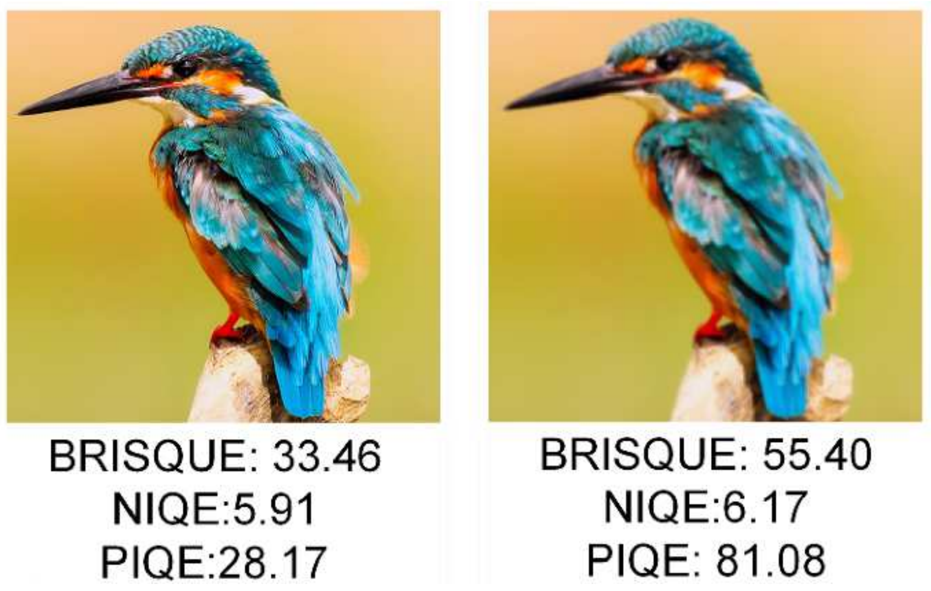


Figure 7.5: Comparison of non-reference image metrics for reference and image with Gaussian noise($\sigma = 10$)

7.3. Video metrics

Frames Per Second

Frames Per Second (FPS) is a commonly used video metric. Simply FPS measures the number of frames displayed per second. It is considered that the minimum viable FPS for video is 24 or 30 as at that FPS video content can provide relatively smooth motion. In the case of video games, explored by Claypool (2007), this minimum viable FPS might be higher (the most commonly used values are 60, 120, and 240 FPS) as it provides a better visual experience and improves responsiveness.

Runtime Per Image

This metric is specific to the case of real-time video upscale and we are going to call it Runtime Per Image (RPI). Due to the need for fluid video, we will measure this measure in milliseconds. If we have an origin video of FPS and we can generate images faster than the paired FPS value, our video will change the duration in real-time (if we are not implementing video buffers). Due to this, RPI is a showcase of upscale performance when we can generate images in less RPI then it is necessary to fluidly run it in real-time (shown in Table 7.1).

FPS	5	15	**24**	**30**	45	60	120	240
RPI [ms]	200	66.67	**41.67**	**33.33**	22.22	16.67	8.33	4.17

Table 7.1: Most commonly used FPS for video paired with minimum inference time needed to upscale video in real-time. Bolded 24 FPS is the most common FPS for video.

7.4. Subjective metrics

Subjective metrics are widely regarded as one of the most challenging yet efficient evaluation methods for image and video upscale quality. These metrics rely on participants who evaluate and provide judgments on the perceived quality of the upscaled video. While subjective metrics require more resources and time compared to objective metrics, they provide valuable insight into the experience of the viewer.

Objective metrics still can not provide valuable enough metrics to provide input that is pleasing to the human eye. Although there are research efforts in deep learning models for quality assessment with subjective input, it is hard to even determine what is objectively pleasing to the human eye's perception. Enjoyment of viewing images and video depends on individual differences, biases, and contextual factors as explored by Reibman and Schaper (2006).

Mean Opinion Score

Mean Opinion Score (MOS) is a widely used standard for subjective media assessment. This assessment requires a user study with a set of images varying discrete number scales mostly from 1 to 5 or from 1 to 10 where higher values indicate better quality. The International Telecommunication Union (ITU) published ITU-R BT.500[3] guideline for video quality assessment, which provides detailed recommendations for conducting MOS experiments. The

[3]https://www.itu.int/rec/R-REC-BT.500

result of the metric is an average quality rating which typically range from 0 to 100. In this case, a higher score is better.

Differential Mean Opinion Score

While MOS focuses on one stimulus (one image) Differential Mean Opinion Score (DMOS) focuses on the difference between two stimuli (two images). A most important difference to MOS is that DMOS represents preference between two images which can provide different valuable input in the quality of upscale. User study asses preference of image on a discrete scale where lower numbers show a preference for image number one and higher numbers show a preference for image two. Typically DMOS values range from 0 to 100, a lower score is better for image one (which is often taken as reference stimuli).

8. Implementation

The implementation of the real-time video application was done in Python using popular libraries such as TensorFlow, PyTorch, Keras, and OpenCV. The main script `rt_upscale.py` can be executed with various command-line arguments to specify the mode and set the required parameters for the application to function properly.

All of the source code is available in the GitHub repository of the project [1]

8.1. Preprocess

Crop video

Crop video mode allows us to crop squares in the middle of the video to be used for upscaling later. This mode is used to create dimensions sizes of the same dimensions to get comparable results. Command line arguments for this methods are following: `--mode preprocess --preprocess_mode crop_video --original_video_path <org_video_path> --crop_video_size <crop_size>`

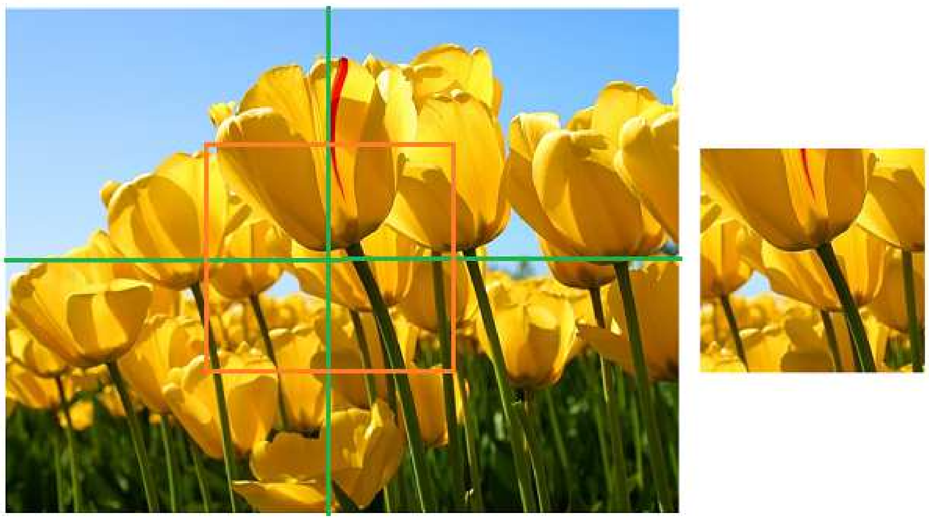


Figure 8.1: Example of rectangle crop implemented

[1] `https://github.com/dinogrgic1/real-time-video-upscale-master-thesis`

Downscale video

Downscale video mode allows us to downscale video with factor `downscale_ratio` to be used for upscaling later. This mode is used to create a downscale video used later for upscaling to the original size (GT video). Command line arguments for this methods are following: `--mode preprocess --preprocess_mode downscale_video --original_video_path <original_video_path> --downscale_video_ratio <downscale_ratio>`

8.2. Onnx

Save ONNX

Save ONNX mode loads the model of the most popular deep learning frameworks and exports it to ONNX format. Command line arguments for this method are following: `--mode onnx --onnx_mode save_onnx --onnx_model_path <onnx_model_path>`

Save Engine

Save engine mode loads ONNX model, optimizes it and exports it to engine file to be loaded for inference. Command line arguments for this method are following: `--mode onnx --onnx_mode save_engine --model_path <model_path>`

Upscale

Within this mode, we initiate the upscaling process for a particular video file by specifying a desired resize factor. Moreover, we have the flexibility to choose the target device for performing the upscaling operation. If the ONNX mode is selected, an additional argument, `onnx_engine_path`, must be provided. Alternatively, when choosing CPU or GPU mode, the argument `--model_path` needs to be specified. Command line arguments for this methods are following: `--mode upscale --upscale_device [cpu, gpu, onnx] --onnx_engine_path <saved_onnx_engine_path> --model_path <saved_model_path> --original_video_path <video_path> --scale <scale_factor>`

Video Stream

To achieve the real-time performance of the model, it is crucial to implement a buffering mechanism that leverages the computational power of the GPU to load frames in parallel. This can be accomplished by utilizing a custom VideoStream class, which utilizes multiple threads and OpenCV to efficiently load multiple frames simultaneously into a buffer. The detailed implementation of the VideoStream class can be in the GitHub repository [2].

8.3. Helper scripts

Powershell helper scripts were used to through all images in the dataset calling the Python script with specified arguments. All scripts can be found under `scripts` folder in GitHub repository.

Crop dataset

Crop dataset script was used to go thorugh all videos in dataset and crop center crop them to four given dimensions (128×128, 256×256, 348×348, 512×512). An example of the script used is the following:

```
Get-ChildItem -Path $folderPath -Filter *.mkv | ForEach-Object {
    $fileName = $_.Name
    $filePath = $_.FullName
    $fileNumber = $fileName -replace '[^0-9]', ''
    $cropVideoSizes = 128, 256, 384, 512
    foreach ($cropVideoSize in $cropVideoSizes) {
        $newFileName = "{0:D3}.mkv" -f $fileNumber
        $command = "python rt_upscale.py --mode preprocess
         ↪  --preprocess_mode crop_video --original_video_path $filePath
         ↪  --crop_video_size $cropVideoSize"
        Write-Host "Processing file: $fileName"
        Write-Host "Executing command: $command"
        Invoke-Expression $command
        Rename-Item -Path $filePath -NewName $newFileName
    }
}
```

Downscale dataset

Similar to cropping dataset downscale dataset scrip was used to downscale all videos by two factors 0.5 and 0.25. An example of the script used is the following:

```
Get-ChildItem -Path $folderPath$cropped_size -Filter *.mkv |
 ↪  ForEach-Object {
    $fileName = $_.Name
```

[2] https://github.com/dinogrgic1/real-time-video-upscale-master-thesis/blob/main/processing/VideoStream.py

```
    $filePath = $_.FullName
    $cropVideoSizes = 0.25, 0.5
    foreach ($cropVideoSize in $cropVideoSizes)
    {
        $command = "python rt_upscale.py --mode preprocess
         ↪  --preprocess_mode downscale_video --original_video_path
         ↪  $filePath --downscale_video_ratio $cropVideoSize"
        Write-Host "Processing file: $fileName"
        Write-Host "Executing command: $command"
        Invoke-Expression $command
    }
}
```

Random frames

The following code snippet from a Jupyter Notebook was utilized to extract random frames from the videos in the validation set and save them as individual images. This Python code was specifically employed to generate a collection of randomly selected frames that would be used for showcasing the grid of upscale methods and datasets.

```
total_frames = vidcap.get(cv2.CAP_PROP_FRAME_COUNT)
for i in range(NUM_RANDOM_FRAMES):
    rand_frame = random.randint(0, total_frames)
    vidcap.set(cv2.CAP_PROP_POS_FRAMES, rand_frame)
    success, image_gt = vidcap.read()
    if success:
        cv2.imwrite(f"{OUTPUT_PATH}_{rand_frame}.jpg", image_gt)
```

9. Results

In order to compare the performance of the models, we will conduct tests on various hardware configurations, including CPU, GPU, and ONNX-optimized model variants, across multiple image dimensions. By increasing the image sizes linearly and ensuring consistent hardware usage across all scenarios, we aim to gain valuable insights into the inference speed of each specific model.

The evaluations were conducted using the following PC configuration and software versions:

- Processor: 12th Gen Intel Core i7-12700F
- Graphics Card: NVIDIA GeForce RTX 3060 (12 GB VRAM)
- RAM: 16GB DDR4
- Python version: 3.9
- CUDA version: 12.1

All experiments will be conducted on all validation subsets of the dataset. Each experiment will be conducted on 4 different dimensions: (128×128), (256×256), (384×384), (512×512). Each video will be upscaled by a factor of 2 and 4.

CPU inference

For the task of CPU inference we will just run all the models on the CPU. This is the default setting for most of the deep learning frameworks.

GPU inference

For the task of GPU inference we will just run the models directly with the framework each model is defined on, without optimizations.

ONNX optimization

For this experiment each model will be converted to ONNX and then an optimized engine will be built for it.

Results show that ONNX optimization may help to optimize the model further if it was not already manually optimized. Even if manual theoretical optimization is made there are still lots of room for small improvement which may lead to a bigger inference speed boost for images of higher dimensions. This is particularly seen in the ONNX optimization of Fast-SRGAN. It is clear that optimization is needed in order to upscale in real-time.

RPI / [ms]		x2 upscale			x4 upscale		
LR dimension	Model	CPU	GPU	ONNX	CPU	GPU	ONNX
(128,128)	Fast-SRGAN	x	x	x	195.612	67.224	**8.105**
	FSCRNN	**19.154**	**12.375**	**2.789**	**19.467**	**13.867**	**3.946**
(256,256)	Fast-SRGAN	x	x	x	836.613	99.334	**30.931**
	FSCRNN	57.061	**14.91**	**5.3779**	68.229	**28.74**	**12.976**
(384,384)	Fast-SRGAN	x	x	x	1867.23	167.532	72.965
	FSCRNN	162.866	**18.507**	**10.656**	168.674	42.562	**29.744**
(512,512)	Fast-SRGAN	x	x	x	3327.278	257.566	126.235
	FSCRNN	275.573	**28.424**	**18.733**	306.881	56.926	46.873

Table 9.1: Results of ONNX optimisations in regards to unoptimized GPU and CPU usage for inference. Bolded measurements show a smooth upscale that is fluid for a minimally 24 FPS video.

		FSRCNN			**Fast-SRGAN**	
	Upscale	**x2**	**x4**	**x2**	**x4**	**x4**
	Dimensions	**(128x128)**	**(128x128)**	**(256x256)**	**(128x128)**	**(256x256)**
Nearest Neihgbour	**PSNR**	12.62	12.79	12.91	12.63	12.70
	SSIM	0.3471	0.3746	0.3794	0.3633	0.3741
	LPIPS	0.5621	0.6957	0.5719	0.7066	0.5751
Bicubic	**PSNR**	12.44	13.00	13.06	12.84	12.85
	SSIM	0.3327	0.3928	0.3926	0.3810	0.3857
	LPIPS	0.5580	0.6798	0.5756	0.6912	0.5785
Lanczos	**PSNR**	12.49	12.85	12.96	12.69	**12.61**
	SSIM	0.3371	0.3841	0.3834	0.3700	**0.3689**
	LPIPS	0.5441	0.6904	0.5586	0.7015	**0.5528**
Model	**PSNR**	**12.37**	**12.67**	**12.84**	**12.51**	12.74
	SSIM	**0.3276**	**0.3638**	**0.3741**	**0.3532**	0.3781
	LPIPS	**0.5355**	**0.6481**	**0.5507**	**0.6577**	0.5615

Table 9.2: Image quality metrics indicate that both FSRCNN and Fast-SRGAN may have higher image quality then standard mathematical approaches

10. Conclusion

In conclusion, ONNX optimization demonstrates a significant improvement in real-time image inference speed. This enhancement is particularly noticeable in models that have not been manually optimized, such as GANs. While additional performance gains may be achieved through manual optimization, this thesis highlights the effectiveness of automated tools for model optimization in inference.

Furthermore, the findings demonstrate that GANs require substantial optimization improvements to effectively generate high-resolution (HR) images. In contrast, CNN approaches exhibit greater potential as they operate more akin to image enhancers in traditional upscaling methods.

Future work could encompass fine-tuning models using the train-test subsets specifically created for this thesis. Furthermore, efforts can be made to identify datasets that contain both low-resolution (LR) and high-resolution (HR) images captured by the same camera. By eliminating the need to downscale images, this approach would provide cleaner LR images for upscaling, thereby minimizing the impact of image artifacts.

BIBLIOGRAPHY

Afnan Afnan, Faiz Ullah, Yaseen Yaseen, Jinhee Lee, Sonain Jamil, and Oh-Jin Kwon. Subjective Assessment of Objective Image Quality Metrics Range Guaranteeing Visually Lossless Compression. *Sensors*, 23(3):1297, Jan 2023. ISSN 1424-8220. doi: 10.3390/s23031297. URL http://dx.doi.org/10.3390/s23031297.

Kajal T. Claypool. On frame rate and player performance in first person shooter games. *Multimedia Systems*, 13:3–17, 2007.

Chao Dong, Chen Change Loy, and Xiaoou Tang. Accelerating the Super-Resolution Convolutional Neural Network, 2016.

Leonardo Galteri, Lorenzo Seidenari, Marco Bertini, and Alberto Del Bimbo. Towards Real-Time Image Enhancement GANs. pages 183–195, Berlin, Heidelberg, 2019. Springer-Verlag. ISBN 978-3-030-29887-6. doi: 10.1007/978-3-030-29888-3_15. URL https://doi.org/10.1007/978-3-030-29888-3_15.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks, 2014.

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

Don P. Mitchell and Arun N. Netravali. Reconstruction Filters in Computer-Graphics. *SIGGRAPH Comput. Graph.*, 22(4):221–228, jun 1988. ISSN 0097-8930. doi: 10.1145/378456.378514. URL https://doi.org/10.1145/378456.378514.

Anish Mittal, Anush Krishna Moorthy, and Alan Conrad Bovik. No-Reference Image Quality Assessment in the Spatial Domain. *IEEE Transactions on Image Processing*, 21(12):4695–4708, 2012. doi: 10.1109/TIP.2012.2214050.

Anish Mittal, Rajiv Soundararajan, and Alan C. Bovik. Making a “Completely Blind” Image Quality Analyzer. *IEEE Signal Processing Letters*, 20(3):209–212, 2013. doi: 10.1109/LSP.2012.2227726.

Amy Reibman and Thilo Schaper. Subjective performance evaluation of super-resolution image enhancement. 01 2006.

Hamid R. Sheikh, Muhammad F. Sabir, and Alan Conrad Bovik. A Statistical Evaluation of Recent Full Reference Image Quality Assessment Algorithms. *IEEE Transactions on Image Processing*, 15:3440–3451, 2006.

Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P. Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network, 2016.

Connor Shorten and Taghi M. Khoshgoftaar. A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*, 6:1–48, 2019.

Medasani Venkatanath, Praneeth aand Sumohana S. and Swarup S. Blind image quality evaluation using perception based features. U *2015 Twenty First National Conference on Communications (NCC)*, pages 1–6, 2015. doi: 10.1109/NCC.2015.7084843.

Zhou Wang, Alan Bovik, Hamid Sheikh, and Eero Simoncelli. Image Quality Assessment: From Error Visibility to Structural Similarity. *Image Processing, IEEE Transactions on*, 13:600–612, 05 2004. doi: 10.1109/TIP.2003.819861.

Ren Yang and Radu Timofte. NTIRE 2021 Challenge on Quality Enhancement of Compressed Video: Dataset and Study. U *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2021.

Ren Yang, Radu Timofte, Xin Li, Qi Zhang, Lin Zhang, Fanglong Liu, Dongliang He, Fu li, He Zheng, Weihang Yuan, Pavel Ostyakov, Dmitry Vyal, Magauiya Zhussip, Xueyi Zou, Youliang Yan, Lei Li, Jingzhu Tang, Ming Chen, Shijie Zhao, Yu Zhu, Xiaoran Qin, Chenghua Li, Cong Leng, Jian Cheng, Claudio Rota, Marco Buzzelli, Simone Bianco, Raimondo Schettini, Dafeng Zhang, Feiyu Huang, Shizhuo Liu, Xiaobing Wang, Zhezhu Jin, Bingchen Li, Xin Li, Mingxi Li, Ding Liu, Wenbin Zou, Peijie Dong, Tian Ye, Yunchen Zhang, Ming Tan, Xin Niu, Mustafa Ayazoglu, Marcos Conde, Ui-Jin Choi, Zhuang Jia, Tianyu Xu, Yijian Zhang, Mao Ye, Dengyan Luo, Xiaofeng Pan, and Liuhan Peng. AIM 2022 Challenge on Super-Resolution of Compressed Image and Video: Dataset, Methods and Results, 2022.

Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric, 2018.

Appendix A
Metrics table

Metric	Media	Type	Better
PSNR	image	full-reference	higher
SSIM	image	full-reference	higher
LPIPS	image	full-reference	lower
BRISQUE	image	no-reference	lower
NIQE	image	no-reference	lower
PIQE	image	no-reference	lower
FPS	video	-	higher
Time per image	video	-	lower

Appendix B
Acronyms

GT	Ground Truth
LR	Low-Resolution
HR	High-Resolution
CNN	Convolutional Neural Network
GAN	Generative Adversarial Network
FPS	Frames Per Second
CUDA	Computer Unified Device Architecture
ONNX	Open Neural Network Exchange
SRGAN	Super-Resolution Generative Adversarial Network

Modeli dubokog učenja za povećanje rezolucije video sekvence u stvarnom vremenu

Sažetak

Ključni fokus ovog rada je tehnike za povećanje rezolucije videa i slike u stvarnome vremenu, specifično je naglašena optimizacija dubokih modela kako bi uspjeli postići performanse potrebne za rad u stvarnom vremenu. Rezultati ovog rada demonstriraju sve potencijalne načine kako se CUDA i ONNX mogu iskoristi snagu GPU i optimizacije kako bi se postigao rad u stvarnome vremenu.

Ključne riječi: povećanje rezolucije slike, povećanje rezolucije videa, real time, duboko učenje, ONNX

Deep Learning Models for Real-Time Video Upscaling

Abstract

The primary focus of this thesis revolves around techniques for real-time image and video upscaling, specifically emphasizing the optimization of deep learning models to attain the necessary speed for real-time inference. The results of this thesis provide a comprehensive demonstration of the potential of CUDA and ONNX in harnessing the capabilities of GPUs and optimization methods to achieve real-time image and video inference.
Keywords: image upscale, video upscale, real-time, deep learning, ONNX