Design History File for "Pixel Perfect" Video Upscaler

Documentation

Submitted for 2023 Group Consultancy Project

By

Jiaqi Ge, Jue Wang, Kiril Avramov, Tianyu Zhao, Zihao Xue

Supervised By

Dr. Cong Ling

Department of Electrical and Electronic Engineering
Imperial College London

July, 2023

Contents

List of Figures **5**

1 Introduction, Purpose and Scope **7**

2 Design Planning and Management **9**

2.1 Requirements Gathering 9

2.1.1 Stakeholders and Role Assignment 9

2.1.2 Requirements of Project 9

2.1.3 Length of Project 9

2.1.4 Project Risk 10

2.2 Overall Design Process 10

2.3 Responsibilities of Team Members 11

2.4 Record of Meetings and Decisions Taken 13

3 Design Concept Development **16**

4 Literature Review **18**

4.1 Interpolation 18

4.1.1 Bilinear 18

4.1.2 Bi-cubic 19

4.1.3 Lanczos 19

4.2 Frequency Domain methods 20

4.2.1 Fourier Based Methods 20

4.2.2 Wavelet Based Methods 21

4.2.2.1 Direct Wavelet Transform Based Methods 21

4.2.2.1.1 Wavelet Zero Padding 21

4.2.2.1.2 Image Registration with Block Matching . . . 21
4.2.2.1.3 Weighted Motion Compensation . . . 22
4.2.2.1.4 Image Registration with Non-Local Means . . . 24
4.2.2.1.5 Enhanced with Stationary Wavelet Decomposition . . . 25
4.2.2.1.6 Enhanced with Edge Directional Interpolation . . . 27
4.2.2.2 Complex Wavelet Transform Based Methods . . . 27
4.2.2.2.1 with Bicubic Interpolation . . . 28
4.2.2.2.2 with Edge Directional Interpolation . . . 28
4.3 Optical Flow and Back-projection . . . 29
4.4 Adaption based on local information . . . 30
4.5 Neural Network . . . 34
4.5.1 Neural Network performance with software . . . 34
4.5.2 Parallel threads and GPU utilization . . . 35
4.5.3 Reducing the computing complexity from the architecture . . . 36
4.5.4 The state-of-art architecture for Super-resolution . . . 36
4.5.5 Transformer . . . 38

5 Implementation **40**
5.1 Back-end Development: Algorithms . . . 40
5.1.1 Interpolation . . . 40
5.1.1.1 Bicubic Interpolation . . . 40
5.1.1.2 Lanczos Interpolation . . . 42
5.1.2 Wavelet Based Methods . . . 42
5.1.2.1 Dual Tree Complex Wavelet Transform (DT-CWT) with Lanczos Interpolation . . . 42
5.1.2.2 Wavelet Zero Padding . . . 44
5.1.2.3 Discrete Wavelet Transform with Stationary Wavelet Transform . . . 45
5.1.2.4 Discrete Wavelet Transform with NEDI . . . 48
5.1.3 Local Structure Estimation . . . 49
5.1.3.1 Code . . . 50
5.1.4 Sobel Operator Based Methods . . . 50
5.1.4.1 Sobel Operator Based . . . 50
5.1.4.2 Sobel Operator Based with Zig-Zag Suppression . . . 52
5.1.5 NEDI (New Edge-Directed Interpolation) . . . 54
5.1.6 Deep learning ESRGAN . . . 55
5.2 Front-end Development: Graphical User Interface (Jiaqi) . . . 58

5.3 Integrating Algorithms with GUI 60

6 Final Built Design Specifications **62**

6.1 Functions and Features 63

6.2 System Requirements 63

6.3 Intended Users 63

6.4 Evaluation Metrics and Performance considerations 64

7 Design Evaluation **65**

7.1 Testing Results on Images 65

7.2 Testing Results on Videos 72

8 Sustainability Report **75**

8.1 Environmental Sustainability 75

8.2 Social and Economic Sustainability 75

9 Ethical Considerations and Consequences **77**

10 Conclusion and Future Work **78**

11 References **80**

A Code and Licenses **87**

A.1 Local Structure Based Methods: Naive Python implementation with C implementations available on GitHub. 88

A.2 NEDI and Improved NEDI Code 91

A.3 Full license for GNU tools 94

A.4 Full license for numpy library 110

A.5 Full License for OpenCV Library 111

A.6 Full License for dtcwt Library 114

A.7 Full License for PySimpleGUI 115

List of Figures

2.1 Main Stages of Overall Design Process 10
2.2 Gantt Chart for the Project 12
2.3 Record of Meetings and Decisions Taken with Rationale (Part 1) 14
2.4 Record of Meetings and Decisions Taken with Rationale (Part 2) 15

4.1 Wavelet Zero Padding [48] 21
4.2 Alignment and Assignment Process for Block Matching Image Registration [52] 23
4.3 Weighted Motion Compensation [43] 23
4.4 Overall WMC DWT Algorithm Structure [43] 24
4.5 Overall DWT with NLM Algorithm Structure [76] 25
4.6 Overall DWT with SWT Algorithm Structure [16] 26
4.7 DWT with EDI Algorithm Structure [48] 27
4.8 Typical Structure for DT-CWT Based Algorithms [15] 28
4.9 Structure of Multi-Frame EDI DT-CWT Algorithm [34] 28
4.10 From [6], the two columns show the two different possibilities 31
4.11 illustration of geometric duality: R is the covariance 32
4.12 "training" window used to optimise prediction coefficients 32
4.13 Sobel based Edge Interpolation [11] 33
4.14 Zig Zag Suppression (Axis Rotated) [10] 34
4.15 Implementation of thread for convolutional operation 35
4.16 The architecture flowchart of SR-GAN with corresponding kernel size (k), number of feature maps (n) and stride (s) indicated for each convolutional layer. [42] 37

5.1 SRGAN Architecture 58
5.2 GUI Configuration 59

6.1 Flowchart for the Video Upscaler Application Software 62

7.1 Original High Resolution Image 0882 from DIV2K 67
7.2 Original, Bicubic and Lanczos Interpolation 67
7.3 Wavelet Based Methods with Haar and Biorthogonal 1.3 Wavelet 69
7.4 Wavelet Based Methods with CDF 9/7 and Antonini 9/7 wavelet 69
7.5 Adaptation Based on Local Information Interpolation and ESRGAN 71
7.6 Comparison between V1 and V2 Local Structure Estimation 72
7.7 One Random Frame of Original High Resolution London Video (1920x1080) 72
7.8 One Random Frame of Original High Resolution Las Vegas Video (3840x2160) 72

Chapter 1

Introduction, Purpose and Scope

The advent of cutting-edge screen panel technology has led to the proliferation of devices and televisions equipped with 4K and 8K Ultra High Definition (UHD) screens. Despite these advances, the majority of TV stations continue to stream videos at lower resolutions, predominantly 480p or 720p, via cable or satellite. As such, to maximize the potential of UHD screens, there exists a compelling need for a proficient video upscaling solution capable of converting incoming low-resolution videos into 4K or 8K UHD videos.

At present, there are several mature video super-resolution solutions, each with its own set of strengths and weaknesses. Notably, the Lanczos algorithm, currently a widely-used solution, strikes a balance between computational consumption and output video quality. However, it lacks sufficient focus on edge preservation, resulting in sub-optimal sharpness and an overall perception of blurriness. On the other hand, more modern, artificial intelligence-based methods, while yielding superior results, necessitate intensive computation, making real-time video upscaling a challenging task.

In view of this, our design project aims to explore and compare different video upscaling algorithms, with the objective of identifying one that produces satisfactory output quality with acceptable speed, and potentially suitable for implementation on Field-Programmable Gate Arrays (FPGAs) for real-time upscaling. As the designed product of this project, the "Pixel Perfect" video upscaler application software is developed to showcase and apply our research outcomes.

Super-resolution (SR) methodologies, both Single Image SR (SISR) and Multi-Image SR (MISR), form the cornerstone of our analysis. SISR techniques enhance image resolution by predicting missing pixels using interpolation and statistical analysis, while MISR leverages multiple displaced images, or frames, of the same scene to recover lost information. We delve into frequency domain approaches that attempt to recover high-frequency components, primarily by extending the frequency spectrum, and spatial domain techniques that infer high-resolution images from pixel relationships in local neighbourhood, as well as methods combining frequency and spatial domain analysis.

Lastly, our exploration extends to learning-based methods, which enhance resolution by inferring high-frequency components from large datasets.

The significance of our design project and the upscaler application developed lies not only in their potential to aid in the selection and implementation of high-quality, real-time video upscaling algorithms on FPGAs, but also in its potential as a valuable reference for further algorithmic comparisons in the field of video upscaling. With a long-term vision, our work and design is anticipated to contribute towards achieving super-resolution videos with accurate color, well-preserved brightness, and sharp edges on screen devices such as televisions. Our ultimate goal is to ensure that the capabilities of modern UHD screens are fully exploited, offering viewers an uncompromised, high-quality viewing experience.

Chapter 2

Design Planning and Management

2.1 Requirements Gathering

At the start of the design process, requirements for the design were firstly gathered through meeting with the client.

2.1.1 Stakeholders and Role Assignment

The external stakeholder of this project is Intel UK Ltd, who is the client represented by Mr. Kieron Turkington. The internal stakeholders of this project are the five members of our Video Scaler team and Dr. Cong Ling, who is the supervisor of the project. All five team members have identified themselves as developers. All team members also act as testers, project managers and project administrators when needed. The exact responsibilities are detailed in the Responsibilities of Team Members section.

2.1.2 Requirements of Project

The client's requirement of the project is reviewing a good range of existing, particularly edge adaptive, video upscaling algorithms; implementing and evaluating some in software, ultimately produce one literature review. The client is more interested in algorithms that could be implemented in FPGA, preferably for real-time video processing; but other algorithms are also accepted.

2.1.3 Length of Project

The project has a duration of 8 weeks and 2 days, commencing from 2nd of May 2023 and ending on 29th of June 2023.

2.1.4 Project Risk

The main risk involved in this project comes from the time constraint. It is highly probable that external deadlines for this project will not be met, if there is not limitation to the range of algorithms being reviewed, implemented and evaluated; and if no internal deadlines are set. To prevent the risk, limitations to the range of algorithms explored and internal deadlines were set at the planning stage of the project and were being evaluated and updated regularly. On the other hand, this project would not involve any purchase of hardware equipment, therefore, the budget constraint is highly unlikely to cause any risks.

2.2 Overall Design Process

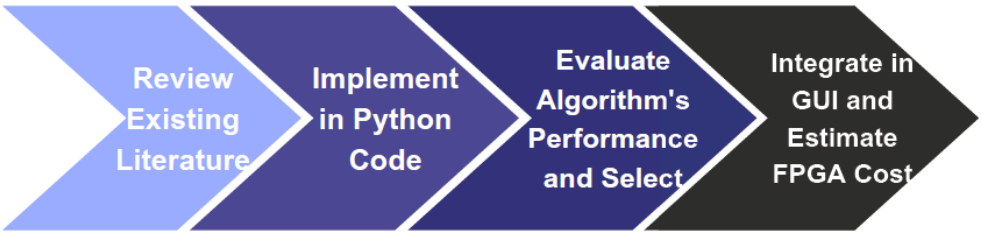


Figure 2.1: Main Stages of Overall Design Process

In the context of our project, our innovative approach was manifested through several aspects, as we sought to tackle the challenge of real-time video up-scaling for screen devices like TVs. We primarily focused on innovating through strategic methodology, thorough research, comprehensive comparisons, advanced implementation, and user-friendly presentation. The overall design process involve four main stages executed in chronological order:

- Reviewing existing literature: Our process commenced by extensively researching published papers and articles from IEEE, which helped us gain a holistic understanding of video up-scaling solutions. Unlike conventional approaches that may rely on a few known methods, our team took an innovative step by evaluating numerous methodologies and algorithms for potential applicability and effectiveness.
- Implementing in Python or MATLAB code: We further showcased our innovative spirit by creating a software implementation of the selected algorithms. Not only did this allow us to test their performance in a realistic setting, but it also helped us anticipate potential implementation challenges on FPGAs. This preemptive step underscores our innovative problem-solving strategy, as it enabled us to foresee issues that may arise during hardware implementation.
- Testing and evaluating algorithm's performance and selecting five out of them: Our innovative approach was further highlighted in the selection of algorithms to test and evaluate. Our client's interests guided us, and we

chose not only the most suitable algorithms but also those that piqued our client's curiosity. This innovative criterion allowed us to maintain a balance between fulfilling current project requirements and looking ahead to possible future enhancements.

- Integrating the algorithms in graphical user interface (GUI) and estimating the FPGA cost of implementing these algorithms in terms of number of adaptive logic modules (ALM) needed: We developed detailed evaluations and comparisons of the selected algorithms, assessing their computational complexity and the potential logic gates they may consume if implemented on FPGAs. This unique, predictive approach demonstrated our forward-thinking mindset, as we not only assessed the algorithms' present performance but also their future impact on hardware resources.

Before carrying out the four main stages, there was also a requirements gathering and planning stage. The detailed overall design process with relevant timeline is shown in figure 2.2.

This design process is planned at the onset of the project, as it is the most appropriate design process to satisfy the requirements from the client.

2.3 Responsibilities of Team Members

Every team member is responsible for reviewing one direction of video upscaling algorithms and implementing some of the algorithms in Python or MATLAB. The five directions were decided at start of the project after a broad review of existing literature on video upscaling, which are interpolation, local structure based methods, explicit edge detection based methods, wavelet based methods and neural network based methods. There are some other tasks being allocated, including testing algorithms' performance on images and videos, front-end development of the software application.

The five directions were refined throughout the reviewing process, with one sub-direction of Sobel operator based methods added under the explicit edge detection methods; the local structure and explicit edge detection based methods were combined into one direction in the literature review produced and a new direction of optical flow and back projection based methods was added.

The detailed responsibilities of each team member are listed below:

- Jiaqi Ge: Literature review and Python implementation of wavelet based methods and Sobel operator based methods; Front-end development (Graphical User Interface) of the video upscaler software application; Testing results on DIV2K images

- Jue Wang: Literature review and MATLAB implementation of explicit edge detection based methods; Testing results on videos

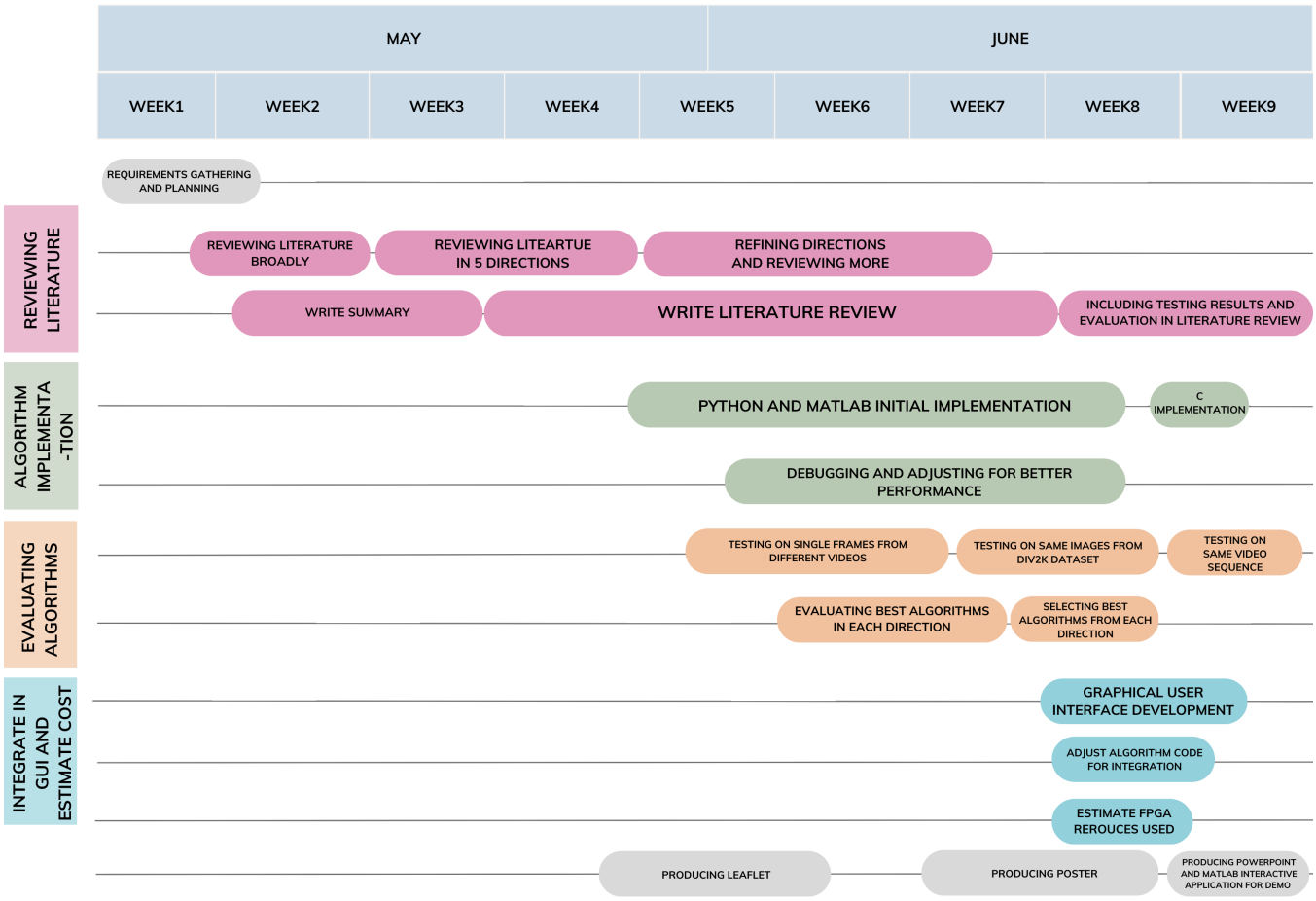


Figure 2.2: Gantt Chart for the Project

- Kiril Avramov: Literature review, Python and C implementation of local structure based methods, literature review of optical flow and back projection based methods
- Tianyu Zhao: Literature review and Python implementation of neural network based methods; Testing results on DIV2K images and videos
- Zihao Xue: Literature review and Python implementation of interpolation, neural network based methods and Sobel operator based methods

2.4 Record of Meetings and Decisions Taken

As shown in figure 2.3 and figure 2.4, meetings were conducted regularly throughout the project, once a week with the client mostly at 11am on Friday; at least once a week within the project team, mostly at 1pm on Thursday. Meeting was also conducted with the supervisor when advice and feedback was needed. The important decisions made during the meetings and their rationale are also recorded in the tables.

Time	Location	Attendees	Main Actions	Decisions Taken with Rationale
11am on 5th of May	On Campus	Whole Video Scaler Team and Mr. Kieron Turkington from Intel	Identify the client's requirements for this project: reviewing a good range of existing video upscaling algorithms, particularly edge adaptive algorithms, implementing and evaluating some in software, ultimately produce one literature review; the client is more interested in algorithms that could be implemented in FPGA preferrably for real-time video processing; learn that Lanczos filtering is the current industry standard video upscaler and its limitations	Initiate to review existing edge-adaptive video upscaling algorithms broadly, which is suitable as starting point; Set the gathered requirements as the design goal, to make sure the end product meet the client's need
12pm on 9th of May	On Campus	Whole Video Scaler Team and supervisor Dr. Cong Ling	Share and discuss the gathered requirements for this project and refine them; Discuss planning for the project	Refine the gathered requirements, set rough internal deadlines for submitting deliverables for the college and for making important progress in the project, to ensure external deadlines can be met
1pm on 11th of May	On Campus	Whole Video Scaler Team	Share and discuss findings from every team member's own literature review: Neural network based approaches appear to be the mainstream algorithms being researched; Share summary written for some papers	Decide that neural network based approaches should be only a subsidiarypart of this project, since the client mentions that neural network based approaches are too computationally expensive for them to bring into production on FPGA and we should not put too much emphasis on them; Send written Summary to client for review and confirm literature review's expected structure
11am on 12th of May	Microsoft Teams	Whole Video Scaler Team and Mr. Kieron Turkington from Intel	Discuss written summary and papers reviewed; discuss about splitting up review directions into 5; the client recommended finding one interesting core algorithm for each direction, then read through their references, for writing the literature review	Split up into 5 directions: Interpolation, Local structure based methods, Explicit edge detection based Methods, Wavelet based Methods, Neural Network Based Methods, after summarizing the existing research directions based on the broad review and confirming with the client; interpolation is involved as a direction for the sake of completeness and as an option for optimal speed
1pm on 18th of May	On Campus	Whole Video Scaler Team	Discuss papers reviewed in each direction; discuss about the cost and benefits of multi-frame and single-frame methods	Decide that single frame methods should be the focus of this project, since most modern methods are single frame based, they are easier to be implemented on FPGA and have less computational cost; multiframe methods may have better performance, but disproportional to their computational cost, nonetheless could be further researched
11am on 19th of May	Microsoft Teams	Whole Video Scaler Team and Mr. Kieron Turkington from Intel	Update progress of literature review, share findings on reviewed papers	Decide to produce a first version of the literature review using Overleaf by 23rd of May, to meet the client's proposed deadline
1pm on 22nd of May	On Campus	Whole Video Scaler Team	Edit the first version of the literature review, make sure it is ready to be sent to the client for review	Finalize the first version of literature review by 23rd of May, start producing the leaflet to meet college's deadline
1pm on 24th of May	On Campus	Whole Video Scaler Team	Produce a draft version of the leaftlet	Decide to use canva.com for designing the leaflet, as it offers versatile functionalities and user-friendly operations; Set structure of the leaflet to be a z-fold with an opening that mimics a magnifying mirror and can vividly show the video upscaling concept; describe our product more as a software application, since that is what the client expects us to produce as a final product for this project and is the most likely product given the time constraints
10am on 26th of May	On Campus	Whole Video Scaler Team and supervisor Dr. Cong Ling	Present draft version of the literature review and the leaflet, receive feedback from the supervisor	Maintain the current structure and content of the leaflet, add more citations and diagrams to the literature review, based on the supervisor's feedback
11am on 26th of May	Microsoft Teams	Whole Video Scaler Team and Mr. Kieron Turkington from Intel	Present draft version of the leaflet, receive feedback from the client; discuss about the literature review and implementing the algorithms	Maintain the current structure and refine some content of the leaflet, add more citations and diagrams to the literature review, based on the client's feedback; initiate algorithm software implementation in Google Colab; decide to use Python for the implementation as it has a lot of useful libraries and is the most common programming language for neural networks and image/video processing; decide to start from single frame methods as they are simpler to implement

Figure 2.3: Record of Meetings and Decisions Taken with Rationale (Part 1)

Time	Location	Attendees	Main Actions	Decisions Taken with Rationale
1pm on 1st of June	Microsoft Teams	Whole Video Scaler Team	Share Google Colab Jupyter Notebooks and discuss implemented algorithms, particularly around the performance and bugs occurred	Decide to test results and debug on images first, as it saves a lot of processing time and the methods being implemented are all single frame
1pm on 7th of June	Intel Marlow Off	Whole Video Scaler Team and Mr. Kieron Turkington from Intel	Present currently implemented algorithms and results, discuss some bugs and possible solutions	Decide to fix the bugs based on the recommended solutions, to ensure the algorithms function properly
10am on 9th of June	Microsoft Teams	Whole Video Scaler Team	Finalize the leaflet; discuss about whether presenting it as FPGA IP Core or software application	Decide to present the product as FPGA IP Core in the leaflet, since that is the end product that could be potentially created from our literature review; Finalize the leaflet, make sure it is ready for submission; Decide to continue implement algorithms, refine and debug
11am on 16th of June	Microsoft Teams	Whole Video Scaler Team and Mr. Kieron Turkington from Intel	Present currently implemented algorithms and results, ask for help to estimate the ALM usage for the implemented algorithms	Acquire estimated ALM usage for the implemented algorithms, for giving a analysis of quality versus cost in the literature review
1pm on 16th of June	On Campus	Whole Video Scaler Team	Test and evaluate the implemented algorithms	Decide that only single-frame algorithms should be implemented, since all the already implemented algorithms are single frame and because of the time constraints; Decide to implement the product as software application due to time constraints, initiate to produce graphical user interface for the software application; decide to review and implement two more algorithms based on sobel operators, since they are highly representative edge adaptive algorithms
11am on 19th of June	Microsoft Teams	Whole Video Scaler Team	Present and discuss the developed graphical user interface, add more algorithms to it; create a Github for the software application	Decide to limit the preview image size, make the user interaction more intuitive, due to inconvenience revealed when group members use the GUI; decide to add video output to the spftware application, so it can be actual video upscaler and suitable for taking into production
12pm on 20th of June	On Campus	Whole Video Scaler Team and supervisor Dr. Cong Ling	Present and discuss the literature review at the current stage, discuss the potential design of poster	Maintain the current structure and content of the literature review, add more test results, based on the supervisor's feedback; Decide to include more graphs in the poster design for maximum visual impact based on the supervisor's advice
1pm on 20th of June	On Campus	Whole Video Scaler Team	Produce draft version of the poster	Decide to continue to use canva.com for the poster design due to the familiarity gained from the leaflet design process; Set the structure; decide to have pictures showing results for each algorithm and a table for maximum visual impact; decide to combine local structure algorithm and explicit edge detection algorithms in one section in the poster and the literature review, since they are all essentially different interpolation based on local neighbourhood of pixels
11am on 22nd of June	Microsoft Teams	Whole Video Scaler Team	Refine the poster, test all algorithms on the same image from DIV2K dataset to have a fair comparison and evaluation of performance and cost	Decide to only use bullet points in the poster and include one diagram for each algorithm for maximum impact, decide to use the same images from DIV2K dataset to test all the algorithms, for fair comparison; decide the evaluation metrics to be PSNR, SSIM, execution time and estimated adaptive logical module usage; use 4x scaling ratio for testing for more significant differences between algorithms; decide to use one 4k video for testing all algorithms' performance on actual videos
11am on 23rd of June	Microsoft Teams	Whole Video Scaler Team and Mr. Kieron Turkington from Intel	Present and discuss the newly developed algorithms, the latest version of literature review and the poster, ask for ALM usage estimate	Maintain the content and structure of the literature review and the poster, refine some details, based the client's feedback
11am on 28th of June	On Campus	Whole Video Scaler Team	Integrate all the chosen algorithms from every direction in the GUI, finalise the software application and literature review	Decide to keep the improved NEDI algorithm implementation in MATLAB, calling it from python using MATLAB engine module, only the MATLAB implementation yield satisfactory results and a fast execution time; decide to improve GUI's visual appearance for better user experience

Figure 2.4: Record of Meetings and Decisions Taken with Rationale (Part 2)

Chapter 3

Design Concept Development

In the concept development phase of our project, our team was guided primarily by the client's requirements, which included a comprehensive literature review of video super resolution methodologies, their comparative study, and a specific interest in edge-adaptive, FPGA-implementable, and real-time functioning algorithms.

To meet these needs, we embarked on an extensive review of academic papers and other relevant literature in the field of video upscaling. Through this rigorous review, we identified five major classes of video upscaling methods: interpolation-based, frequency domain methods, optical flow and back-projection, local structure-based, and machine learning-based methods.

We then selected suitable algorithms from these identified classes that could be implemented and evaluated within our two-month project timeline. To make an informed decision, we considered factors such as the complexity of the algorithms, the depth of our understanding of them, and their suitability for edge-adaptive processing and FPGA implementation.

Once we had chosen our algorithms, we proceeded with their implementation in Python or MATLAB, depending on the complexity and suitability of the algorithm for each programming language. This process served two primary purposes: first, to gain insights into the performance of each algorithm in upscaling videos, and second, to estimate the computational cost on FPGAs in terms of logic gate consumption. Both are potentially useful for developing the final design product.

During the research process before implementation, we decided to not only implement the chosen algorithms in Python or MATLAB, but also implement some in hardware description language(HDL) including Verilog and VHDL. The final designed product was conceived to be a FPGA IP Core that performs video upscaling with selectable algorithms, which would satisfy the client's requirements more. It was though that we could at least implement the algorithms in C after finishing Python and MATLAB implementation, then use C to HDL compilers and only need to verify and validate the HDL design through simulation and on FPGA.

Nonetheless, as we proceeded with the algorithms' implementation in Python or MATLAB, we realized that the two-month project timeline would not permit us to finish implementing the algorithms in HDL, even with the help of C to HDL compilers. Since a large amount of time has already been spent in finishing Python and MATLAB implementation and a significant amount of time is required for debugging, verifying and validating the converted HDL implementation. Therefore, it was decided to produce an application software instead, that performs video super-resolution with a range of parameters that can be adjusted by users to fulfill their needs, including but not limited to super-resolution algorithms and scaling ratio. This design product will also satisfy the client's requirements and allow us to engage in a comprehensive exploration of the selected algorithms.

The knowledge and insights gained from this design concept development phase, which was carried out at the start and repeated in the middle of the project, were invaluable in shaping our understanding and guiding our next steps in this project.

Chapter 4

Literature Review

In order to explore existing video upscaling algorithms and develop an useful end product, we have carried out an extensive literature review of video up-scaling algorithms throughout the last several decades. Our review is focused on five areas includes basic interpolation, frequency domain methods, optical flow and back-projection, adaption based on local information and neural network.

This is the main body of the literature review we completed towards the direct requirement of the client, the full text pdf of the literature review can be found on the project's GitHub repository: https://github.com/jiaqige0612/VideoScaler/blob/main/Literature%20Review/Literature_Review.pdf.

4.1 Interpolation

The earliest methods of super-resolution are based around interpolation and can mostly be classifies as SISR, reconstruction, spatial domain methods. These methods are still used today and due to their simplicity and computational efficiency. However, these methods have drawbacks in terms of the quality of the produced HR image.

4.1.1 Bilinear

Arguably the first and most basic methods were bilinear and nearest neighbour interpolation. Bilinear interpolation simply assumes a linear relationship between the intensities of the pixels in a given neighbourhood. [7] The LR image pixels need to be mapped onto an HR map corresponding to their relative positions in the LR image so that accurate interpolation can be carried out; This process will be true of most SISR super-resolution techniques. The process of interpolating occurs in two orthogonal directions. Note that often, before interpolation a Gaussian filter would usually be applied to smooth the image and remove noise. However, because of the linear assumption beforehand, the linear interpolation may not accurately capture complex or non-linear relationships between data points. [63]

4.1.2 Bi-cubic

Similar to bilinear, bicubic interpolation operates cubic interpolation twice, which uses cubic polynomials to interpolate between data points. A cubic polynomial is in the form of $f(x) = ax^3 + bx^2 + cx + d$. And the goal in cubic interpolation is to find the four coefficients of the cubic polynomial that best fits the given data points. These coefficients are determined using the values of the neighboring data points and at least four data points. [37]
The general form of a bicubic interpolation equation for a target pixel at position (x, y) is:

$f(x, y) = \sum_{i=0}^{3} \sum_{j=0}^{3} a_{ij} \cdot x^i \cdot y^j$

where a(i, j) are the coefficients to be determined. Bicubic interpolation involves more complex calculations and has higher computational complexity compared with linear way. On the other hand, because of the complex calculations, bicubic interpolation is able to capture more intricate patterns and structures and less block artifacts. These make the image after bicubic interpolation have a higher image quality.

4.1.3 Lanczos

Lanczos resampling, named after its originator Cornelius Lanczos, is a sophisticated, windowed sinc-function-based interpolation method commonly employed in digital image processing. It provides superior performance in retaining high-frequency details during the resampling process compared to other standard methods such as bilinear or bicubic interpolation.

The essence of the Lanczos method lies in the application of a sinc filter, coupled with a finite impulse response (FIR) filter, generally referred to as a window function. The sinc function, derived from the Fourier transform of a rectangular function, is ideal for bandlimiting a signal to avoid aliasing. However, due to its infinite extent, it is not practical for direct implementation. Consequently, a window function is introduced to confine the sinc function within a finite extent, resulting in the Lanczos kernel. Lanczos interpolation can be described by the equations below [68]:

$$L(x) = \begin{cases} \text{sinc}(x)\text{sinc}(x/a) & \text{if} -a < x < a \\ 0 & \text{otherwise} \end{cases}$$

Equivalently,

$$L(x) = \begin{cases} 1 & \text{if } x = 0 \\ \frac{a \sin(\pi x) \sin(\pi x/a)}{\pi^2 \pi^2} & \text{if } 0 < |x| < a \\ 0 & \text{otherwise} \end{cases}$$

A distinct advantage of the Lanczos resampling method is its ability to reproduce high-frequency details more accurately compared to other interpolation techniques. This is due to the better frequency response of the Lanczos kernel, particularly in the stopband region, which minimizes the aliasing effects. This characteristic makes it an

attractive option in image processing applications where preserving fine details during resampling is a priority.

The parameter 'a' is a positive integer, typically 2 or 3, which determines the size of the kernel. We determine the weight L(x) corresponding to different positions in the window based on the input point, and then take the weighted average of the point values in the template [55], follows equation like this,

$S(x) = \sum_{i=x-a+1}^{x+a} s_i L(s-i)$

Overall, Lanczos interpolation is widely used due to its fast speed, good result with rather low cost.

4.2 Frequency Domain methods

Analysis of images in the frequency domain has given the basis for much of the research surrounding super-resolution by outlining the fundamental mathematical principles surrounding it. In this domain, one considers the the optical system to have a transfer function which attenuates frequencies beyond a certain cut-off [25]. Therefore, in order to reconstruct the image, one must extend this spectrum. The main principle which allows SISR in the frequency domain is that "a function of a complex variable is determined throughout the entire Z-plane from a knowledge of its properties within an arbitrarily small region of analyticity." [23] This means that the information needed to extend the spectrum (for a finitely sized object) is available in any analytical sample of the frequency spectrum of the function. The proof of analyticity and in-depth analysis of frequency spectrum extension is available in [25].

4.2.1 Fourier Based Methods

The method mentioned here simply outlines the mathematical principles and implements a proof of concept rather than an applicable SR algorithm. This paper proves that the limit to restoration is imposed not by how precise the optical system is but by the noise introduced in the system. Another approach [21], based on the above principles, considers a "segment of the known spectrum to be the sum of the complete spectrum plus an error spectrum which is equal and opposite to the true spectrum outside the given segment." In this way, the authors are able to devise an iterative algorithm to minimise the error spectrum energy.

The above methods both propose extending the spectrum of the image and are both SISR methods, the first MISR method was presented in [65]. This method was developed for improving images from satellites which are globally shifted which allows the researchers to use the Fourier transform's shifting property [26] to take into account the shifts. However this method illustrates the flaw in the Fourier approach to super-resolution; though it provides the theoretical basis of SR, this method, as well as those mentioned above, require small, global motion and suffer greatly if the motion estimation is not correct as stated in [65]. For our purposes of video-scaling, where motion is local and complex, Fourier methods are not applicable. We touch on modern methods of motion estimation and MISR in section 3.

4.2.2 Wavelet Based Methods

Wavelet-based methods offer more practicality compared to Fourier-based methods. Instead of merely providing a frequency domain representation of the signal, wavelet transforms represent the signal in both time and frequency domains, which enables access to localized information about the signal. In general, wavelet-based methods use the low-resolution frame to replace the low-frequency wavelet-transformed sub-band [16], then adjust the high-frequency sub-bands with different approaches. All the sub-bands are subsequently fused and inverse wavelet-transformed to produce the high-resolution frame.

4.2.2.1 Direct Wavelet Transform Based Methods

This series of methods utilize direct wavelet transform (DWT) to produce four sub-bands from the original video frame, with one low-frequency sub-band consisting of LL sub-band and three high-frequency sub-bands consisting of LH, HL and HH sub-bands. The LL sub-band is replaced by the original low-resolution frame, optionally up-scaled by bicubic interpolation; whereas the high-frequency sub-bands are generally estimated or refined using motion estimation based on the current and nearby frames and then interpolated using bicubic interpolation. The resultant sub-bands produce the high-resolution video frame through IDWT.

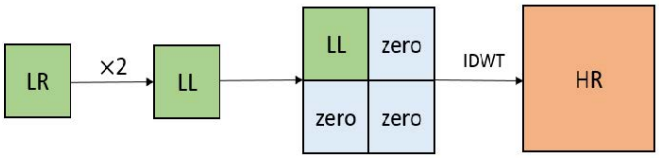


Figure 4.1: Wavelet Zero Padding [48]

4.2.2.1.1 Wavelet Zero Padding Wavelet zero padding is the simplest discrete wavelet transform based method. As shown in figure 4.1, it solely takes in every single low-resolution frame as the low-frequency LL sub-band, after multiplying all its coefficients by 2 and up-scaling it with bicubic interpolation; fills the high-frequency sub-band coefficients with zeros, then performs inverse discrete wavelets transform (IDWT) to reconstruct the single high-resolution frame [48]. It is simple and fast, but the super-resolved frame produced does not have any edge enhancement and appears blurry. Besides, it can only reliably upscale the video frame by 2; other scaling factors are possible through interpolating the LL sub-band, but the performance is not ideal and primarily dependent upon the interpolation method.

4.2.2.1.2 Image Registration with Block Matching Image Registration can ideally yield correct high-frequency sub-band coefficients and high-resolution video frames, if it is done accurately. Block matching is one

of the least computationally demanding image registration methods. Multi-frame registration is a more intuitive approach, yet has low accuracy if the original frame has large motion blur or occluded objects.

To acquire high-accuracy image registration, multi-scale registration, which utilizes self-similarity within a single frame, is proposed in [52]. As shown in figure 4.2, the video frame is first DWTed to multi-scale sub-bands of high and low frequency. In the wavelet domain, using exhaustive search block matching, based on an evaluation function of the sum of squared difference values, overlapping regions of pixels in the original video frame Y^0 are aligned to the down-scaled low-frequency sub-bands $\{Y^n_{LL}|n = 1, 2, 3, 4, 5, 6\}$. Noticeably, the block size is the same for all the scaled sub-bands, meaning that similar objects of various sizes will all be matched together. Using the alignment results, for the matched blocks, the down-scaled high-frequency sub-band $\{Y^n_{LH}, Y^n_{HL}, Y^n_{HH}|n = 1, 2, 3, 4, 5, 6\}$ coefficients are assigned with a window function to the corresponding block within the high-frequency sub-band of the same size as the original frame $\{Y^0_{LH}, Y^0_{HL}, Y^0_{HH}\}$. If after expanding the block by 2, the difference between coefficients of the expanded block from the low-frequency sub-band Y^{n-1}_{LL} and the original frame Y_0 is less than a threshold, the coefficients from corresponding blocks in $\{Y^{n-1}_{LH}, Y^{n-1}_{HL}, Y^{n-1}_{HH}\}$ are assigned to the block in the up-scaled high-frequency sub-bands $\{Y^{-1}_{LH}, Y^{-1}_{HL}, Y^{-1}_{HH}\}$. A single super-resolved video frame is then produced using the original video frame Y_0, two levels of high-frequency sub-bands $\{Y^0_{LH}, Y^0_{HL}, Y^0_{HH}\}$ $\{Y^{-1}_{LH}, Y^{-1}_{HL}, Y^{-1}_{HH}\}$ through two-level inverse wavelet transform.

This method produces videos with noticeably sharp edges and without unpleasing artifacts or blurs, while yielding a very high PSNR and SSIM. Since the threshold ensures that the high-frequency sub-bands of the super-resolved video are similar enough to the original video. The adoption of step-search-like block matching for registration reduces the computational cost and is advantageous for real-time processing. This method is proposed only for upscaling with a factor of 4, however, upscaling with other even numbers based on it is possible with minor modifications, which could further investigated.

4.2.2.1.3 Weighted Motion Compensation In order to accomplish image registration and motion estimation more accurately, it is proposed to utilize both spatial and temporal information in the wavelet domain, through the aid of weighted motion compensation (WMC) in [43]. The low-resolution video frames are firstly Lanczos interpolated and DWTed. Then WMC is performed in the wavelet domain. In other words, for every block of pixels in one frame, the four frequency sub-bands from DWT each produce a motion vector and a predicted block for the next frame. The motion vectors can be estimated using the block-matching method discussed before, which can also be encoded in compressed videos to save computational costs. The final predicted block in each sub-band in the next frame is a weighted sum of predicted blocks from all the frequency sub-bands in the previous frame, which is given by

$$\begin{aligned} B^{sb}_W = {} & w_1 \times B^{sb}_{mvLL} + w_2 \times B^{sb}_{mvLH} \\ & + w_3 \times B^{sb}_{mvHL} + w_4 \times B^{sb}_{mvHH} \end{aligned} \tag{4.1}$$

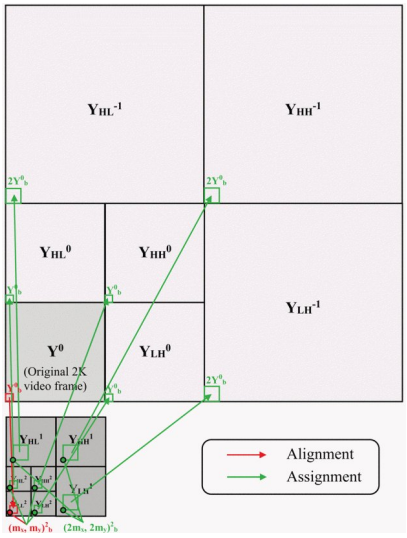


Figure 4.2: Alignment and Assignment Process for Block Matching Image Registration [52]

where sb is any sub-band from LL, LH, HL, HH; B^{sb}_{mvLL}, B^{sb}_{mvLH}, B^{sb}_{mvHL}, B^{sb}_{mvHH} are blocks predicted by using $MV^{B_w}_{LL}$, $MV^{B_w}_{LH}$, $MV^{B_w}_{HL}$, $MV^{B_w}_{HH}$, which are motion vectors obtained by motion estimation in the four sub-bands individually. The process is shown in figure 4.3. The use of WMC greatly enhances the accuracy of estimating complex motions, which are crucial for producing correct super-resolved videos.

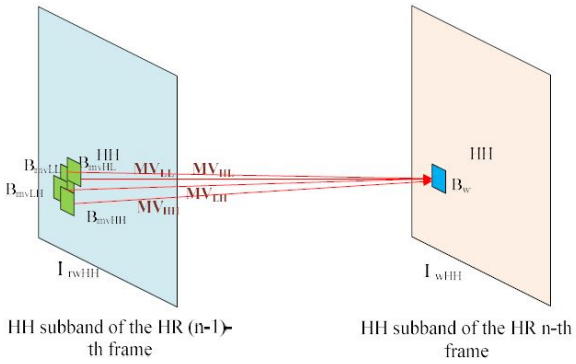


Figure 4.3: Weighted Motion Compensation [43]

To further enhance the edges, wavelet difference coefficients (WDC) are estimated to yield higher frequency components. WDC is the difference of wavelet coefficients between the $n-1$ super-resolved frame and the $n-1$ Lanczos-interpolated frame, which represents the information about the edges missing from the Lanczos-interpolated frame but existing in the detailed super-resolved frame. The $n-1$ super-resolved frame is obtained recursively, except for the first frame which is obtained through bicubic interpolation. The WDCs are processed by WMC and

then added to the Lanczos-interpolated DWT coefficients.

The Lanczos-interpolated DWT coefficients, WMC and the Lanczos-interpolated DWT coefficients modified based on WDC, are combined linearly to produce the final super-resolved frame. The weight of each term is the normalized reciprocal of the Euclidean norm between the IDWT of them and the original low-resolution frame.

This method, which is shown in figure 4.4, produces videos with more sharp edges and coherent frames, and has

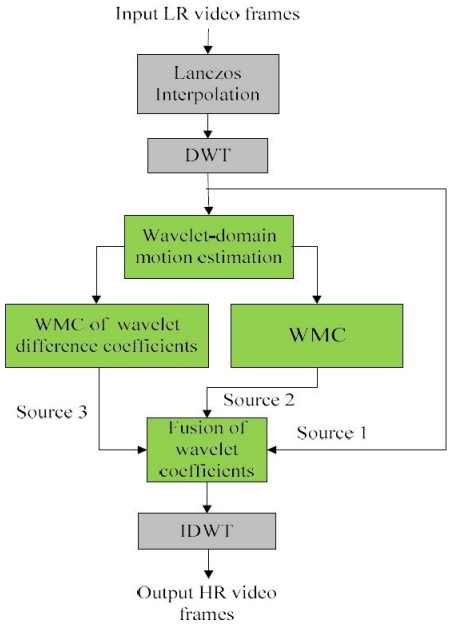


Figure 4.4: Overall WMC DWT Algorithm Structure [43]

a relatively low computational cost compared to other motion compensation methods for improving the motion estimation. The usage of adaptive weights also ensures that the final high-resolution video frame is as close as possible to the original low-resolution frame. Nonetheless, it is experimentally shown that this method, or motion estimation based method, is limited to videos containing local or object motions. For videos containing dynamic and complex motions, the PSNR gain of this method is small compared to the more traditional ones including Lanczos and bicubic interpolation; although for more static videos, the gain is distinctive.

This method has only been experimented for a scaling factor of 2, however, any other scaling factors are possible as long as the interpolation method and motion estimation algorithm used permit them.

4.2.2.1.4 Image Registration with Non-Local Means To bypass motion estimation, which often yields inaccurate motion vectors and final super-resolved results when complex motion changes between frames are involved, Non-Local Means (NLM) based method is used in [76]. With the assumption that a small block of pixels appears several times in one frame and also in different frames, wavelet domain coefficients corresponding to a pixel can be

estimated using a normalized weighted sum of wavelet domain coefficients corresponding to all pixels in current and nearby frames, in other words, by using non-local means. The estimate of a pixel at position (i, j) is given by

$$\hat{z} = \frac{\sum_{t=1}^{T} \sum_{(k,l)\in N(i,j)} \omega_t(k,l) y_t(k,l)}{\sum_{t=1}^{T} \sum_{(k,l)\in N(i,j)} \omega_t(k,l)} \tag{4.2}$$

where y are the original low resolution wavelet coefficients, t is the frame index and w are the filter weights, which are based on the similarity between pixel blocks and given by

$$\begin{aligned} w(k,l) = \exp\{-\frac{\left\|p_{y(i,j)} - p_{y(k,l)}\right\|_2^2}{2\sigma^2}\} \\ \cdot f(\sqrt{(i-k)^2 + (j-l)^2 + (t-1)^2}) \end{aligned} \tag{4.3}$$

The original frame is firstly DWTed and Lanczos interpolated, then coefficients in HL and LH sub-bands are interpolated using an arbitrary wavelet interpolation method and re-estimated through NLM, unless the coefficients have large magnitudes satisfying $|M_{coef}| > \alpha \times std(|M_{coef}|)$; the interpolated original frame forms the LL sub-band and HH sub-band is filled with zeros, as shown in figure 4.5.

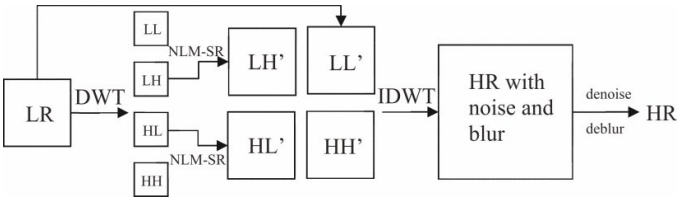


Figure 4.5: Overall DWT with NLM Algorithm Structure [76]

This method avoids explicit motion estimation; instead of estimating a target pixel position in nearby frames, it simply considers all possible positions where the pixel may appear and provides a better estimate of the coefficients when motion changes between frames. This method yields sharper edges and fewer errors around the edges, outperforms former methods including NLM-SR and single-frame wavelet interpolation both in terms of PSNR and visual quality. However, it needs extra denoising and deblurring filters, although it is still a relatively fast method, particularly if fast denoising filters are used; this method also does not consider HH sub-band, which introduces noise and blurs.

This method has only been tested for a scaling factor of 2, however, any other scaling factors are theoretically possible as long as the interpolation method and the denoising and deblurring filters used permit them.

4.2.2.1.5 Enhanced with Stationary Wavelet Decomposition The redundancy and shift-invariance of the DWT, which means that DWT coefficients are inherently interpolable, makes them suitable for preserving high-

frequency components in video super-resolution. Nonetheless, in DWT, the sub-bands are all down-sampled from the original video frame, leading to information loss; stationary wavelet transform (SWT), which is similar to DWT but produces sub-bands with the same size as the original video frame and preserves more high-frequency components, can be used to modify the high-frequency sub-bands of DWT, thereby reduces the information loss and produce super-resolved videos with sharper edges, as proposed in [16].

The low resolution video frame is first SWT and DWT. The DWT high-frequency sub-bands are first bicubic-interpolated and then incremented by the corresponding high-frequency coefficients from SWT. The modified DWT results and original video frame are bicubic-interpolated to be used respectively as the high and low-frequency sub-bands for IDWT to create the super-resolved video frame, as shown in figure 4.6.

This method utilizes only simple bicubic interpolation of wavelet transformed coefficients, without any forms of motion estimation, which greatly reduces the computational cost, since wavelet transform is relatively fast. On the other hand, it is effective at enhancing edges, through the addition of SWT coefficients, providing superior PSNR and visual results compared to simple wavelet transform based methods.

This method only works with even-number upscaling factors, since it involves upscaling all sub-bands by $\frac{overallupscalingfactor}{2}$. Other scaling factors may be possible if a interpolation algorithm for non-integer factors is used instead, such as the method proposed in [28].

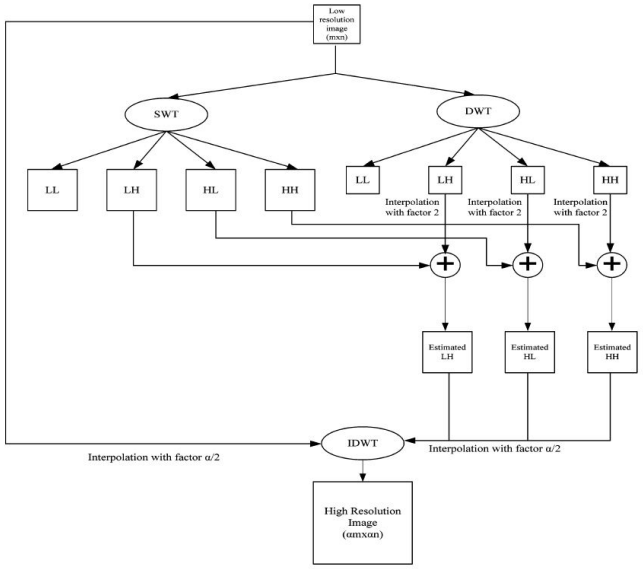


Figure 4.6: Overall DWT with SWT Algorithm Structure [16]

4.2.2.1.6 Enhanced with Edge Directional Interpolation Edge directional interpolation (EDI) classifies pixels as edge and non-edge pixels and applies different interpolation algorithms to preserve the edge structure. It can be used to firstly produce an up-sampled version of the low-resolution input frame, which is then DWTed to yield high-frequency sub-bands. The high-frequency sub-bands are then combined with the original input frame as the low-frequency sub-band to produce the final super-resolved frame, through IDWT, as shown in figure 4.7. Although this method proposed in [48] is mostly a switch of the order of DWT and interpolation, compared to the conventional DWT-based methods, it produces visually sharper frames, which are particularly noticeable when many edges and textures are present.

This method is tested for a scaling factor of 2. Other integer scaling factors are possible if the sub-bands are interpolated in the process; non-integer scaling factors should also be possible depending on the interpolation method used.

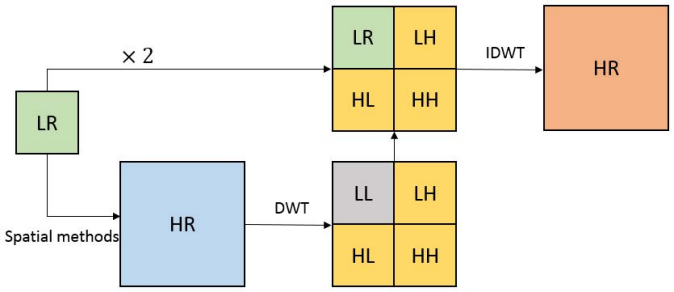


Figure 4.7: DWT with EDI Algorithm Structure [48]

4.2.2.2 Complex Wavelet Transform Based Methods

This series of methods utilize dual-tree complex wavelet transform (DT-CWT) of the low-resolution video frame to produce one low-frequency sub-band consisting of LL sub-band and six high-frequency sub-bands in different directions: $+75°$,$+45°$,$+15°$,$-15°$,$-45°$,$-75°$ [15]. The LL sub-band is replaced by the original low-resolution frame, with its all coefficients multiplied by 2 and up-scaled by bicubic interpolation; the high-frequency sub-bands are also interpolated. The resultant sub-bands produce the high-resolution video frame through inverse dual-tree complex wavelet transform(IDT-CWT), as shown in figure 4.8.

Similar to DWT, DT-CWT is shift-invariant, meaning that the resultant coefficients are inherently interpolable and can be used readily for super-resolution. Different from DWT, DT-CWT can decompose frames in different high-frequency directional sub-bands, which isolates the edge details in different directions and reduces the inter-directional interference in the super-resolution process, resulting in more distinctive edges.

In theory, DT-CWT based methods can upscale video frames with any arbitrary real-number scaling factors. To upscale the final frame by α, the high-frequency sub-bands need to be interpolated with a factor of α and the original

frame needs to be interpolated with $\frac{\alpha}{2}$. Nonetheless, the specific interpolation algorithm limits the range of possible α, typically to integers.

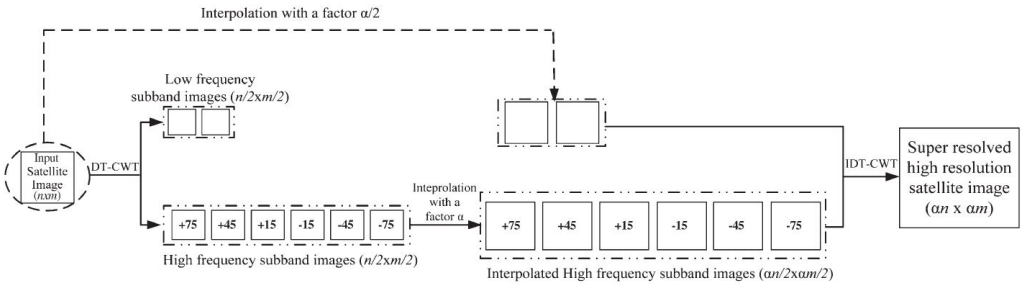


Figure 4.8: Typical Structure for DT-CWT Based Algorithms [15]

4.2.2.2.1 with Bicubic Interpolation The relatively simple method for up-scaling high-frequency sub-bands is bicubic interpolation, as proposed in [15]. However, this results in a relatively poor performance in both PSNR and visual quality. Nonetheless, the computational cost for wavelet transform and bicubic interpolation is relatively low. It also does not require any motion estimations or denoising and deblurring filters. Most importantly, this method can already benefit from the main advantages of DT-CWT based methods, which are the preservation of high-frequency components demonstrated by edges after interpolation and reduction in inter-directional interference.

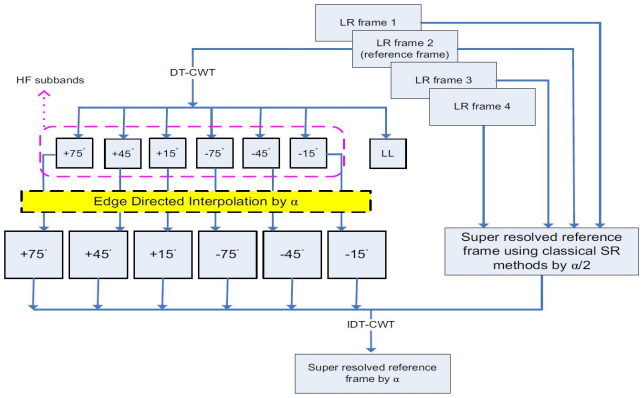


Figure 4.9: Structure of Multi-Frame EDI DT-CWT Algorithm [34]

4.2.2.2.2 with Edge Directional Interpolation Edge directional interpolation (EDI) classifies pixels as edge and non-edge pixels and applies different interpolation algorithms to them to preserve the edge structure. Interpolating high-frequency sub-bands using EDI, as proposed in [35], yields super-resolved videos with noticeably fine and

sharp edge details. Additionally, to acquire a low-frequency sub-band that gives a super-resolved video with finer details, the nearby frames of the current frame being super-resolved can be taken, along with the current frame, for multi-frame super-resolution using an arbitrary method, which is proposed in [34] and shown in figure 4.9. This is particularly helpful for low-resolution video sequences, since the nearby frames can be viewed as multiple samples of the original frame.

This method yields super-resolved videos with sharp edges and considerably outperforms former DT-CWT methods in PSNR. However, this method has not yet been implemented in real-time, although that is a possibility.

4.3 Optical Flow and Back-projection

Fundamentally, MISR can be summarised as mapping an image of the same scene, distorted in some way [**?**] (eg. translation) hereon out referred to as the distorted image onto a reference image; for the purpose of video-scaling this would involve mapping a subsequent frame onto the current one, and finally fusing the two. Back-projection and optical flow methods approach this very directly. One early example [32], inspired by the imaging technology employed in CAT scans involves making an initial guess for the HR image and the "back-projection" of the differences in subsequent images to improve the guess. The problem with the above approach is that it estimates global translation and would therefore not be applicable to video.

In order to perform local motion estimation across the entire image, one can employ optical flow, this method originally described in [50] has become a basis for motion estimation in computer vision. It is founded in three assumptions: motion between frames is small and that pixel intensities are constant. These two assumptions produce the optic flow constraint equation:

$$I_x u + I_y v + I_t = 0 \tag{4.4}$$

This can be rewritten as:

$$[I_x(p) + I_y(p)] \begin{bmatrix} u \\ v \end{bmatrix} = -I_t(p) \tag{4.5}$$

The above is an under-determined system, Lucas and Kanade resolve this by introducing the third and final assumption of spatial coherency which turns the above into an over-determined system of linear equations which can be solved via the Moore-Penrose inverse.

This feasibility of this method in super-resolution is discussed in [75]. This paper concludes that super-resolution with optical flow based methods is feasible, however some frequency-domain analysis suggests that in order to estimate high-frequencies the model needs to be very accurate, however, small gradients in the image present higher error in flow estimation. The method used in this paper is an augmented version of the method proposed in [33] which uses expensive image segmentation and object tracking techniques to register the image which here, has been replaced by the optical flow method while The super-resolution technique remains the same. This technique is a

traditional correcting algorithm which uses N LR images to improve the estimate (of the HR pixel) iteratively, N times. The authors conclude that with small noise, and accurate motion estimation, optical flow is an effective method of super-resolution.

The above method uses dense flow estimation which is computationally expensive, sensitive to image noise and somewhat prone to error in motion estimation. A more modern approach in [61] seeks to perform local flow estimations only at selected interest points which are determined, in this case by the SIFT interest point detector [49]. Interest point matching is far more robust than dense flow estimation due to the accuracy of the SIFT feature, a double check can also be performed between each feature ie. from the 'reference' frame to the 'distorted' frame and vice versa thereby making it extremely unlikely to have a false match.

In order to construct a support region ($\mathcal{R}$) for which to apply the calculated transformations between the interest points (note that SIFT features are invariant to scale and rotation thus allowing affine transformations (3) to represent motion), a confidence map, utilised in [39] and described in (4) is combined with Canny edge-detection [5].

$$T_i = \begin{bmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ 0 & 0 & 1 \end{bmatrix} \tag{4.6}$$

$$\mathcal{R} = |B_k(L^r(T_i\vec{x}) - L^d(\vec{x}))| < \eta_{map} \tag{4.7}$$

Where B_k is the blurring operator, L^r is the reference frame and L^d is the distorted frame. If the difference between the transformed image and the distorted image is small, then the corresponding pixel location in the confidence map is set to 1. Further, for a pixel to be taken in the support region, both: the pixel and its nearest edge point must fall within the confidence map. The neighbourhood of each interest point is expanded until a connected path exists around each support region. If there exists overlap, pixels are assigned to the region of the nearest interest point.

The problem achieving MISR through mapping frames onto one another is a difficult one in real world scenarios due to noise. However it can be simplified by focusing on specific regions. Drawbacks in terms of computational complexity are prevalent however, and the above algorithms would need to be simplified significantly in order to be applicable to real-world scenarios.

4.4 Adaption based on local information

Often, algorithms which are optimised for computational efficiency are SISR, reconstruction algorithms which adapt interpolation coefficients based on local structure. An early and incredibly simple algorithm is proposed in [6]. This algorithm is best understood in 1D then generalised to 2D.

Figure one shows the process from image capture, which is characterised by a low-pass filter effect and decimation

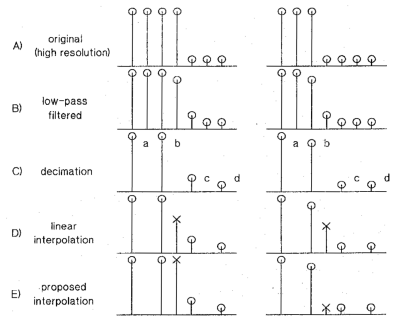


Figure 4.10: From [6], the two columns show the two different possibilities

to the interpolation. The proposed interpolation method is defined as follows:

$$x = \mu b + (1 - \mu)c \tag{4.8}$$

$$\mu = \frac{k(c-d)^2 + 1}{k((a-b)^2 + (c-d)^2) + 2} \tag{4.9}$$

Where k is a user-selected parameter which determines the edge sensitivity. When the edge is midway between b and c, a-b = c-d, so that $\mu = 0.5$ and x = (b+c)/2. Under this condition, the filter behaves as a linear interpolator. However, when the edge is asymmetrically located, eg. when a - b < c - d, so that $\mu > 0.5$, x is now $\approx$ b. The above can easily be extrapolated to two dimensions where the interpolation is simply performed along the horizontal and vertical axes.

This method is incredibly simple, however, it forms the basis of this family of methods which have become more complex and far better performing. A famous and often cited approach is the NEDI (new edge-directed approach) [47]. This approach is based on a statistical approach enabled by the geometric duality between the LR and HR co-variances. Geometric duality refers to the correspondance between local covariances of an LR and HR pixel which are along the same orientation (relative to an edge).

The approach is a continuation of a previous work which performs least-square based adaptive prediction for lossless compression; the details of which are available in [46]. In this paper is addressed the edge-directedness property in detail, however, for completeness, it can be summarised as such: A double rectangular window (fig. 3) is used to obtain prediction neighbours in matrix C in (7).

$$C = \begin{bmatrix} X(n-1-1) & \dots & X(n-1-N) \\ \vdots & & \vdots \\ X(n-M-1) & \dots & x(n-M-N \end{bmatrix} \tag{4.10}$$

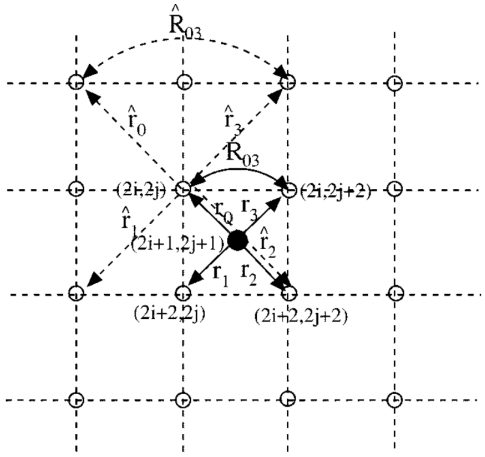


Figure 4.11: illustration of geometric duality: R is the covariance

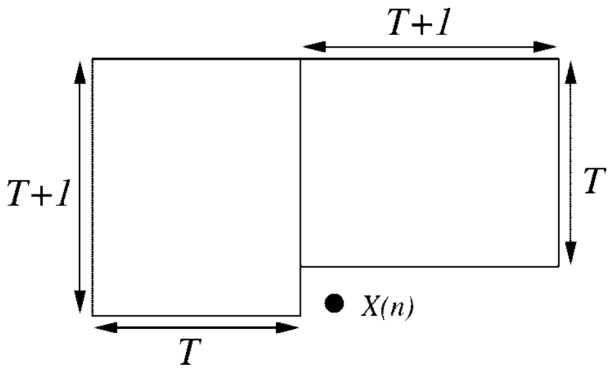


Figure 4.12: "training" window used to optimise prediction coefficients

It is then straight forward to see that when the matrix C contains edge neighbours, it is often full rank. The authors implement a hybrid method between covariance-based adaptive interpolation and linear interpolation to greatly reduce computational complexity.

Despite the efforts to make the algorithm efficient, auto-regression methods will always suffer from long computation times due to the need to solve sets of linear equations. An alternative method, proposed in [62] suggests obtaining HR pixels via a weighted sum resulting from some arbitrary interpolation in two directions, where the weights are chosen based on the error of this arbitrary model when applied to existing LR pixels:

1. apply arbitrary interpolation along the 45° and 135° diagonals for the four (LR) nearest neighbours of the missing (HR) pixel as exemplified in fig. 4.

2. calculate the difference in values obtained from the interpolation and the true pixel values for the neighbours.

3. sum the errors for each diagonal for the four neighbouring pixels such that two error values: one for the 45° and one for the 135° diagonals are obtained.

4. weight the obtained values (in step 1) for the missing HR pixel according to the weights, (8) shows the weighting used by the authors.

$$W_{45} = \frac{Err_{45}^{sf}}{Err_{45}^{sf} + Err_{135}^{sf}} \tag{4.11}$$

Where sf is a scaling factor obtained experimentally in order to make the algorithm more sensitive to the edges. W_{135} is obtained by 1-W_{45}.

This algorithm, in some sense, implements the covariance properties ina much more efficient way. The authors also propose a pipe-lined VLSI implementation which is able to support the up-scaling to UHD at 30fps with clock frequency of 297MHz.

There are other methods like importing sobel filter. In that situation, the edge of the original image is detected by using four Sobel spatial operators, which is proposed in [11] and shown in figure 4.13. And the direction of edge is then measured by using two newly proposed edge detectors employing two experimentally determined threshold values. Based on these, pixels are interpolated adaptively, where non-edge pixels are interpolated through bilinear interpolation and edge pixels are interpolated according to the edge direction:

1. if edge is in 0° direction, pixel a is assigned with value $\frac{I+II}{2}$

2. if edge is in 90° direction, pixel c is assigned with value $\frac{I+IV}{2}$

3. if edge is in 45° direction, pixel b is assigned with value $\frac{II+IV}{2}$

4. if edge is in 135° direction, pixel b is assigned with value $\frac{I+III}{2}$

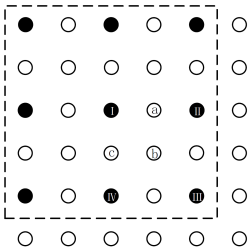


Figure 4.13: Sobel based Edge Interpolation [11]

This algorithm can achieve edge adaptive interpolation, with very little computational cost compared to NEDI. The resultant upscaled video frames have relatively sharp edges, but also with zig-zag artefacts on the edges.

To suppress the zig-zag artefacs, the algorithm is improved in [10]. The improved algorithm uses canny edge detector to create an edge map; the sobel operators are then applied to the edge map to evaluate the edge direction, using Sobel gradients and threshold values obtained from the mean and standard deviation of Sobel gradients. The edge pixels are still interpolated using the same adaptive method proposed in [11]. Pixels next to every detected edge pixels, which lie along the edge pixel's edge direction, are also interpolated adaptively as a weighted sum of its original value and the nearest pixel in the same direction, as shown in figure 4.14. For instance, if the edge pixel is in 45° direction, $f(x-1, y-1)$ is assign with $W_{45}\dot{f}(x-1, y-1) + (1-W_{45})\dot{f}(x-2, y-2)$, in which $W_{45} = \frac{f(x-1,y-1)}{f(x-1,y-1)+f(x-2,y-2)+1}$.

This algorithm can effectively smooth the zig-zag artefacts along the edges produced by adaptive interpolation in [11]. It also achieves this with more complicated adaptive interpolation, but maintains approximately the same low computational cost as the former algorithm through only using Sobel operators on the Canny edge map. Nonetheless, the resultant video frames may have less sharp edges.

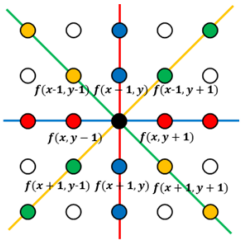


Figure 4.14: Zig Zag Suppression (Axis Rotated) [10]

4.5 Neural Network

4.5.1 Neural Network performance with software

Convolutional neural networks(CNN) have always been popular and effective in the field of image processing. The basic structure of CNN includes multiple interconnected layers to perform operations like convolution, pooling and activation. The convolutional layers with learnable filter can capture the local feature and spatial relationship of the entire images.

According to the comparison gallery [70], it it obvious that deep learning algorithm performs the best. However, deploying CNNs on Field-Programmable Gate Array(FPGA) for real-time processing brings both advantages and challenges. The following review aims to analyze the pros and cons of using CNN and propose potential solutions to the address the challenges.

4.5.2 Parallel threads and GPU utilization

Using CNNs includes intensive matrix operation, leading to high computational complexity. Research about accelerating the video super resolution on FPGA [64] has shown that 97% of the time is spent on the convolutional computation, and it takes 450s to process a single frame. The article proposes a solution to this problem: applying efficient GPU optimization techniques and memory hierarchy of GPU to parallelize the convolution operation. The convolution operation is defined as an array operation where each output element is the weighted sum of the product of filter values with the corresponding input patch values. The computation at each pixel of the entire image is considered as a thread, as Figure4.15 shown, therefore the overall time of convolutional operation will be divided by the number of the parallel threads. Theoretically, the entire operation can be accelerated to the operation time equivalent to one pixel, but it will then require more than 700,000 threads which will cost too much resource in real projects. In the article [64], the method achieved a speed of 225X by performing parallel implementation of reasonable count of thread.

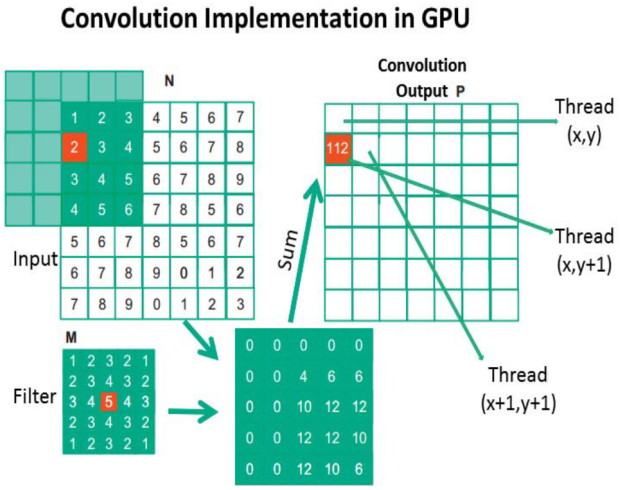


Figure 4.15: Implementation of thread for convolutional operation

Another optimisation to consider is to do with memory management. The hierarchical GPU memory model consists of global memory, constant memory, shared memory and texture memory. Here shared and constant memory has fastest access time and global memory has slowest access time. In the previous method, the input parameters are pre-loaded into global memory. Each thread will access global memory to fetch the parameters and perform the computation. The unnecessary access to memory between convolution and activation can also be cancelled to speed the process faster.

4.5.3 Reducing the computing complexity from the architecture

As the depth of a neural network increases, the performance will be better but also the complexity increase. Thus, finding the balance between complexity and performance is essential to the optimisation. First of all, the format of the input data will affect especially when running the methods on FPGA platform. It is well known that 32-bit floating-point (single precision) data is used in most of the deep learning platforms such as PyTorch [56] and TensorFlow [1]. Therefore converting floating-point data to fixed-point data with optimal bit depth is essential to meet the trade-off between quality and complexity with limited resource.

The common inspiration behind different method is selective patch and reduce the computation over the entire images [17, 38, 40, 60]. One way is to convert the RGB channels to YCbCr channel [71], and only Y channel is used as input for CNNs. The PNSR performance shows that the CNN trained with only Y channel is similar to the CNN trained with RGB channels [17].

To achieve the small use of convolutional filter parameters and line memories, the network incorporates (i) depth-wise separable convolutions, (ii) 1D horizontal convolutions, and (iii) residual connections, each of which is described in detail in the following subsections. As a result, the number of parameters used in proposed network is about 21 times smaller than that of SRCNN-Ex [17], about 4.5 times smaller than that of FSRCNN [18], and 1.56 times smaller than that of FSRCNN-s [18], while maintaining similar PSNR and SSIM performance compared to that of SRCNN-Ex [17]. (i) Depth-wise separable convolutions(DSC) [27] has shown to achieve similar classification performance only with one-ninth of the number of parameters, compared to the cases with conventional non-separable convolutions. The experiment results show as table4.1 below [40]

Method	Bicubic	SRCNN [17]	SRCNN-Ex [17]	FSRCNN [18]	FSRCNN-s [18]
# of Params	-	8k	57k	12k	4k
Bits width	-	32-bit	32-bit	32-bit	32-bit

Table 4.1: Caption

4.5.4 The state-of-art architecture for Super-resolution

Generative Adversarial Networks (GANs) are powerful models in multiple image processing tasks, consisting of two key components: a generator and a discriminator, which are trained simultaneously in an adversarial manner [22]. The generator aims to produce the high-resolution images from the low-resolution images, while the discriminator attempts to distinguish between the generated HR-images and the real HR-images. The GANs model can be applied in lots of image processing tasks, such as text to image synthesis [74],image denoise [8, 72], image to image translation [12], image super-resolution [42] and etc. In the field of super-resolution, faster and deeper convolutional

neural networks have already resulted well in both speed and accuracy [17, 38]. However, the issue of keeping high frequency information and containing the fine texture still exists. GANs have revolutionized the field of by enabling the generation of realistic and visually pleasing high-resolution images, and here SRGAN is introduced. There is two key components of SRGAN which make the model work better at containing the texture of images, the loss function and architecture. The architecture of the model is designed as following figure

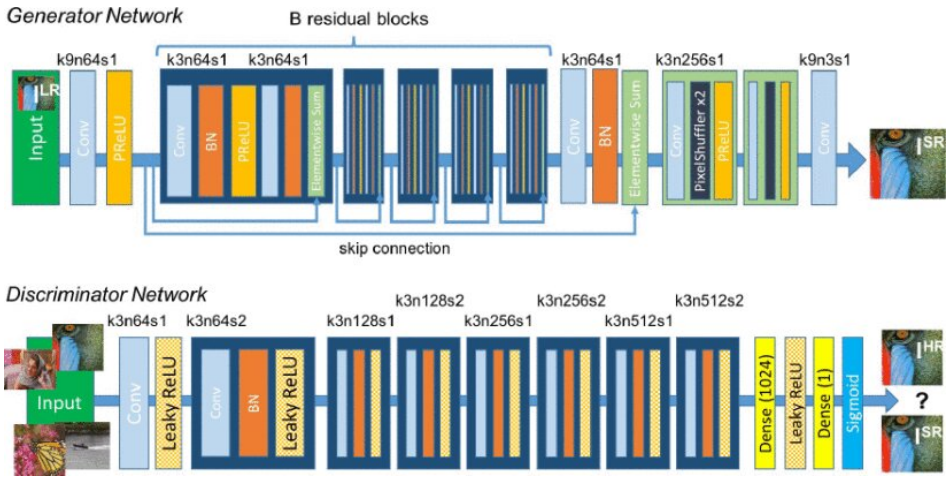


Figure 4.16: The architecture flowchart of SR-GAN with corresponding kernel size (k), number of feature maps (n) and stride (s) indicated for each convolutional layer. [42]

The upper part is the generator component, which will produce the high quality reconstructed images. The core of the network of generator is the residual block layout, which includes two convolutional layers with small 3×3 kernels and 64 feature maps. The residual connection enable the model to keep the information from different dimension, known as the texture. The resolution of input images are also increased by the sub-pixel convolutional layer by Shi et al. [60] The discriminator is defined following the guide summarized by Radford et al. [59] The resulting 512 feature maps are followed by two dense layers and a final sigmoid activation function to obtain a probability for sample classification.

The most common pixel-wise loss function such as mean square error function are difficult to recover the texture as they lead to pixel-wise optimal solution. Typically, the images are over-smooth thus have lower perceptual quality [19, 36, 51]. According to Johnson et al. and Bruna et al. [4, 36] a new loss function is defined based on perceptual similarity:

$$l^{SR} = l_X^{SR} + 10^{-3} l_{GEN}^{SR}$$

The perceptual loss is formulated as the weighted sum of content loss(l_X^{SR}) and adversarial loss($10^{-3} l_{GEN}^{SR}$). The content loss is defined according to the comparison between the result after convolutional layers and the original HR

images. The adversarial loss is defined based on the probabilities of that discriminator regards the re-constructed images as natural images.

4.5.5 Transformer

Transformer has recently gained increasing exposure and usage in image processing field. For up-scaling images, a low-resolution image is required to feed into the transformer model, which learns to generate a corresponding high-resolution image. The transformer contains two sub-layers, namely multi-head self-attention layer and feed-forward layer. The self-attention mechanism operates on an input sequence by computing the dot product of the query and key vectors for each element in the sequence. This yields a matrix of dot product values. The dot product values are then divided by $\sqrt{d_k}$ and passed through a softmax function, resulting in a weight distribution for each query. These weights represent the importance or relevance of each element in the sequence with respect to the corresponding query.

The weighted values are multiplied with the value matrix V, producing the final output of the self-attention mechanism.The equation goes like this,

$$\text{Att}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{4.12}$$

After the self-attention, there comes the feed-forward layer, a transformer decoder responsible for generating the up-scaled image. It takes the output from the encoder as input and generates a sequence of higher-resolution patches. The decoder also includes masked self-attention to ensure that the model considers only the previously generated patches, preventing information leakage from future patches. [67] Those higher-resolution patches are stitched together to form the final up-scaled image after the decoder.

During training, the model is typically provided with pairs of low-resolution and high-resolution images. It learns to minimize the difference between the generated high-resolution image and the ground truth high-resolution image through a loss function, such as mean squared error or perceptual loss. The model's parameters are optimized through back-propagation [73], updating the weights to improve its ability to reconstruct high-resolution details.

Once training is complete, the transformer-based image up-scaling model can take a new low-resolution image as input and generate a high-resolution version with enhanced details, such as sharper edges and more intricate textures.

The transformer decomposes the video sequence into two dimensions, time and space, and then converts this data into self-attention calculations to reduce complexity. The self-attention mechanism allows the model to focus on different parts of the input image when processing each position. It computes the attention weights for each position in the input by comparing it to all other positions. These attention weights indicate the relevance of each position with respect to the others. [77]

Transformer enables the model to capture relationships between different elements of the sequence have a significantly lower number of parameters, regardless of their positions. For CNNs, The number of parameters in a convolutional layer depends on the size of the filters and the number of channels in the input and output. As a result, CNNs can have a large number of parameters, especially in deeper architectures. On the other hand, by using self-attention layer which would enable the transformer to gather information from distant parts of the input sequence without using convolutions or parameter sharing, transformer uses projections to transform the input embeddings into key, query, and value vectors. [9] Those projections, which usually linear projections, have a significantly lower number of parameters. Therefore, transformer has less parameter, less computational complexities with better performance compared with traditional CNN.

Overall, by leveraging the power of self-attention mechanisms, transformers can effectively capture long-range dependencies and learn complex patterns in the image data. This allows them to generate visually appealing upscaled images with improved resolution and enhanced details compared to traditional interpolation-based methods. However, transformer may struggle with capturing spatial information and local patterns that are crucial in certain computer vision tasks. They also require large amounts of data to generalize effectively. Due to their high parameter efficiency, transformers may not perform as well as CNNs on tasks with limited training data too.

Chapter 5

Implementation

Software implementation in this project has utilized the open-source Python library OpenCV [3], Numpy [24], Pillow [13] and sci-kit image [66]. There are other open-source Python library being used in single individual algorithms and in the graphical user interface, which will be detailed and referenced in the relevant section.

5.1 Back-end Development: Algorithms

5.1.1 Interpolation

5.1.1.1 Bicubic Interpolation

We first followed the bicubic method in specific, the convolution method, mentioned in the literature review above.

The function first calculates the dimensions of the new image after scaling, then it performs bicubic interpolation for each pixel in the new image by iterating over its rows and columns using two nested loops.

The weight function is used to compute the weights for cubic interpolation along the x and y axes. This function returns the weights for cubic interpolation based on the distance t from the neighboring pixel. The weights are used to determine the contribution of each neighboring pixel to the final interpolated pixel value.

For each pixel (i,j) in the new image, a nested loop iterates over the 4x4 grid of neighboring pixels in the original image. The pixel values from these neighboring pixels are multiplied by the corresponding weights obtained from cubic weight function and then added to the final pixel value at (i, j) in the new image. We later on apply this bicubic interpolation function to all three RGB channels separately to up-scale a colored image and video.

This direct method performed our desired result. However, as it uses loops and individual pixel calculations, it took much time and was significantly slower than optimized implementations provided by libraries like OpenCV. In OpenCV, as it is a highly optimized C++ code, the cv2.resize() with interpolation method set to cv2.INTERCUBIC

can also perform bicubic interpolation with less cost and much less time, which is demonstrated clearly in the algorithms' testing results on images shown in the design evaluation section. Therefore, the opencv bi-cubic method is the one we choose for final implementation for testing and in the video upscaler software application.

The full code can be found in the project's GitHub repository: https://github.com/jiaqige0612/VideoScaler/blob/main/Algorithms/Bicubic.py. Most relevant code for own implementation is shown below:

```
def bicubic_interpolation(img, scale=2):
height, width = img.shape
new_height = height * scale
new_width = width * scale
new_img = np.zeros((new_height, new_width), dtype=np.float32)

for i in range(new_height):
    for j in range(new_width):
        x = i / scale
        y = j / scale

        # get floor number
        x_floor = int(np.floor(x))
        y_floor = int(np.floor(y))

        # Bicubic interpolation weights
        dx = x - x_floor
        dy = y - y_floor
        wx = np.array([cubic_weight(c) for c in cubic_neighbors(dx)])
        wy = np.array([cubic_weight(c) for c in cubic_neighbors(dy)])

        for k in range(-1, 3):
            for l in range(-1, 3):
                img_x = min(max(x_floor + k, 0), height - 1)
                img_y = min(max(y_floor + l, 0), width - 1)
                new_img[i, j] += img[img_x, img_y] * wx[k + 1] * wy[l + 1]

return new_img
```

```
# weight
def cubic_weight(t):
    if abs(t) <= 1:
        return 1.5 * abs(t)**3 - 2.5 * abs(t)**2 + 1
    # smaller weight away from the target
    elif abs(t) < 2:
        return -0.5 * abs(t)**3 + 2.5 * abs(t)**2 - 4 * abs(t) + 2
    else:
        return 0
#fixed range
def cubic_neighbors(t):
    return [t + 1, t, 1 - t, 2 - t]
```

5.1.1.2 Lanczos Interpolation

The implementation of Lanczos interpolation is done using the `dtcwt.sampling.rescale` function with method set to lanczos in dtcwt library [69], which is an open-source library providing Python implementation of dual-tree complex wavelet transform and associated algorithms including basic interpolation algorithms. The library is further referenced in the Wavelet Based Methods section below. This implementation is chosen instead of OpenCV's `INTER_LANCZOS4`, because this library's lanczos interpolation is used in the implementation of dual tree complex wavelet transform based method, thus it can offer better comparison with the DTCWT method.

5.1.2 Wavelet Based Methods

5.1.2.1 Dual Tree Complex Wavelet Transform (DT-CWT) with Lanczos Interpolation

The implementation of this method is done with the dtcwt library [69], which provides a Python implementation of the 1, 2 and 3-D dual-tree complex wavelet transform and associated algorithms. This open access library permits redistribution and use in source and binary forms:

> "Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
>
> 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
>
> 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. "

The full license for this library is attached in the appendix.

The actual implementation is relatively straightforward. Firstly, the input video frame is decomposed using `dtcwt.compat.dtwavexfm2` function, which performs DT-CWT 2D decomposition, with level 1 and using Antonini 9/7 wavelet, which yield super-resolved images with better visual quality based on testings, compared to, for instance, default near-symmetric 5,7 tap wavelet filters. The low frequency sub-band is then set to upscaled original frame, which is interpolated by the scaling ratio using `dtcwt.sampling.rescale` function; the high frequency sub-bands are interpolated by the scaling ratio using `dtcwt.sampling.rescale_highpass` function. In the original literature review, the upscaling was done with bicubic interpolation, however,lanczos interpolation is chosen here, since lanczos interpolation theoretically have better performance compared to bicubic interpolation; besides, only lanczos interpolation is supported with this library, although bicubic interpolation for the low frequency sub-band can be easily implemented in our own way, it is much harder to implement that on our own for the high frequency sub-bands, which is returned as a pyramid object consisting of tuples storing coefficients of 6 sub-bands together. At the end, the interpolated sub-bands are inverse wavelet transformed to produce the super-resolved frame, using `dtcwt.compat.dtwaveifm2` function. The process can repeated for three channels separately to process colored video frames.

The full code can be found in the project's GitHub repository: https://github.com/jiaqige0612/VideoScaler/blob/main/Algorithms/Wavelet%20Based/DTCWTLanczos.py. The code is also presented below:

```
def scale_waveletlanczos(ds_frame, scale):

    def upscale(im):
        #scale low frequency sub-band
        return dtcwt.sampling.rescale(im, (im.shape[0]*scale, im.shape[1]*scale), 'lanczos')

    def upscale_hf(im):
        #scale high frequency sub-band
        return dtcwt.sampling.rescale_highpass(im, (im.shape[0]*scale, im.shape[1]*scale), 'lanczos')

    trans_result = np.empty((ds_frame.shape[0]*scale, ds_frame.shape[1]*scale, ds_frame.shape[2]))

    for i in range (3):

      # Transform ds_frame
      dsframe_l, dsframe_h = dtcwt.compat.dtwavexfm2(ds_frame[:, :, i], nlevels=1, biort='antonini')

      dsframe_l = upscale(ds_frame[:, :, i])

      dsframe_h_a = []

      for h in dsframe_h:
          print(h.shape)
          dsframe_h_a.append(upscale_hf(h))

      # Inverse transform
      trans_result[:, :, i] = dtcwt.compat.dtwaveifm2(dsframe_l, dsframe_h_a, biort='antonini')

return trans_result
```

5.1.2.2 Wavelet Zero Padding

This method is implemented with PyWavelets [44] [45], which is an open source wavelet transform package for Python.

For each channel of the input frame, the low frequency sub-band is set directly to the original frame, the other three sub-bands are set to Numpy arrays of zeros, with the same dimension as the low frequency sub-band. The low frequency sub-band is multiplied by 2 (by 1.999999999999 in actual implementation to avoid the image exceeding the range of 0-255 and have abnormal colours). All the sub-bands are then assigned to the tuple coeffs2, which is a tuple containing wavelet coefficients in form of `(Approximation, (Horizontal Detail, Vertical Detail, Diagonal Detail))`, whose dimensions are determined using the `pywt.dwt2` function after inputting the original frame. The coefficient

tuple is then transformed to super-resolved frame using the `pywt.idwt2` function.

The discrete wavelet transform is set to level 1. Haar wavelet is used for comparing the output result with the HR frame, since with this implementation, other wavelets produce final image of slightly different dimensions compared to the original one. However, haar wavelet produces images with pixelated texture and the final implementation uses biorthogonal 4.4 wavelet, which yield much finer and more realistic texture.

This implementation upscale videos by 2, but other scaling ratios that are multiples of 2 can be achieved through looping it.

The full code can be found in the project's GitHub repository: https://github.com/jiaqige0612/VideoScaler/blob/main/Algorithms/Wavelet%20Based/Other_Wavelet_Based_Code.ipynb. The most relevant code is presented below:

```
original = ds_frame

# Inverse Wavelet transform
coeffs2 = pywt.dwt2(original[:,:,0], 'haar')
LL, (LH, HL, HH) = coeffs2
trans_result = np.empty((frame.shape[0], frame.shape[1], frame.shape[2]))

original_frame = cv2.resize(ds_frame, None, fx=2, fy=2, interpolation=cv2.INTER_CUBIC)

for i in range(3):
  LL = original_frame[:,:,i]
  LH = np.zeros(LL.shape)
  HL = np.zeros(LL.shape)
  HH = np.zeros(LL.shape)
  coeffs2 = LL*1.999999999999, (LH, HL, HH)
  #inverse wavelet transform
  trans_result[:,:,i]=pywt.idwt2(coeffs2,"haar")
```

5.1.2.3 Discrete Wavelet Transform with Stationary Wavelet Transform

This method is also implemented with PyWavelets [44] [45], which is an open source wavelet transform package for Python.

For each channel of the input frame, the four sub-band coefficients are obtained through DWT implemented with `pywt.dwt2` function. The low frequency sub-band is set to the original frame upscaled by 2 using OpenCv `resize` function with `cv2.INTER_CUBIC`. The original frame is then decomposed using stationary wavelet transform,

using `pywt.swt2`. The resulted SWT high-frequency sub-bands are added to the DWT high-frequency sub-bands upscaled by 2 using OpenCv `resize` function with `cv2.INTER_CUBIC`. The combined high-frequency sub-bands are then further upscaled by 2 using OpenCv `resize` function with `cv2.INTER_CUBIC`.

The low frequency sub-band is then multiplied by 2 (by 1.999999999999 in actual implementation to avoid the image exceeding the range of 0-255 and have abnormal colours). All the sub-bands are then assigned to the tuple coeffs2 containing wavelet coefficients. The coefficient tuple is then transformed to super-resolved frame using the `pywt.idwt2` function.

This implementation has a scaling ratio of 4, but the scaling ratio can be changed to any numbers supported by OpenCv bicubic `resize`.

The discrete wavelet transform is set to level 1. Haar wavelet is used for comparing the output result with the HR frame, since with this implementation, other wavelets produce final image of slightly different dimensions compared to the original one. However, haar wavelet produces images with pixelated texture and the final implementation uses biorthogonal 4.4 wavelet, which yield much finer and more realistic texture.

The full code can be found in the project's GitHub repository: https://github.com/jiaqige0612/VideoScaler/blob/main/Algorithms/Wavelet%20Based/Other_Wavelet_Based_Code.ipynb. The most code is presented below:

```
# Discrete Wavelet transform
trans_result = np.empty((original.shape[0]*4, original.shape[1]*4, original.shape[2]))

original_interpolated = cv2.resize(original, None, fx=2, fy=2, interpolation=cv2.INTER_CUBIC)
for i in range(3):
  coeffs2 = pywt.dwt2(original[:,:,i], 'haar')
  LL, (LH, HL, HH) = coeffs2

  LH = cv2.resize(LH, None, fx=2, fy=2, interpolation=cv2.INTER_CUBIC)
  HL = cv2.resize(HL, None, fx=2, fy=2, interpolation=cv2.INTER_CUBIC)
  HH = cv2.resize(HH, None, fx=2, fy=2, interpolation=cv2.INTER_CUBIC)
  LL = original_interpolated[:,:,i]

 # Stationary Wavelet transform of image, and plot approximation and details
  arr = original[:,:,i]

  level = 0
  titles = ['Approximation', ' Horizontal detail',
            'Vertical detail', 'Diagonal detail']
  for LLs, (LHs, HLs, HHs) in pywt.swt2(arr, 'haar', level=1, start_level=0):
      level += 1

  LHs = LHs + LH
  HLs = HLs + HL
  HHs = HHs + HH

  LHs = cv2.resize(LHs, None, fx=2, fy=2, interpolation=cv2.INTER_CUBIC)
  HLs = cv2.resize(HLs, None, fx=2, fy=2, interpolation=cv2.INTER_CUBIC)
  HHs = cv2.resize(HHs, None, fx=2, fy=2, interpolation=cv2.INTER_CUBIC)

  coeffs2= LL*1.99999999, (LHs, HLs, HHs)
  #inverse wavelet transform
  trans_result[:,:,i]=pywt.idwt2(coeffs2,"haar")
```

5.1.2.4 Discrete Wavelet Transform with NEDI

This method is implemented with PyWavelets [44] [45], which is an open source wavelet transform package for Python; and with "Edges-Directed Interpolation" open source python script [29], which implements NEDI in python.

For each channel of the input frame, the frame is interpolated by NEDI using `EDI_predict` from the script [29], with a scaling ratio of 4. Then the NEDI-interpolated frame is decomposed by DWT using `pywt.dwt2` function. The low frequency sub-band is set to original frame upscaled by 2 using OpenCv `resize` function with `cv2.INTER_CUBIC`.

The low frequency sub-band is then multiplied by 2 (by 1.999999999999 in actual implementation to avoid the image exceeding the range of 0-255 and have abnormal colours). All the sub-bands are then assigned to the tuple coeffs containing wavelet coefficients. The coefficient tuple is then transformed to super-resolved frame using the `pywt.idwt2` function.

This implementation has a scaling ratio of 4, but the scaling ratio can be changed to any numbers supported by OpenCv bicubic `resize`.

The discrete wavelet transform is set to level 1. Haar wavelet is used for comparing the output result with the HR frame, since with this implementation, other wavelets produce final image of slightly different dimensions compared to the original one. However, haar wavelet produces images with pixelated texture and the final implementation uses biorthogonal 4.4 wavelet, which yield much finer and more realistic texture.

The full code can be found in the project's GitHub repository: https://github.com/jiaqige0612/VideoScaler/blob/main/Algorithms/Wavelet%20Based/Other_Wavelet_Based_Code.ipynb. The most relevant code is presented below:

```
# Load image
original = ds_frame
original2x = cv2.resize(original, None, fx=2, fy=2, interpolation=cv2.INTER_CUBIC)
trans_result = np.zeros([np.shape(original)[0]*4, np.shape(original)[1]*4, np.shape(original)[2]])

for i in range (3):
  original_enlarge = EDI_predict(original[:,:,i], 4, 4)

  # Wavelet transform of image

  coeffs = pywt.dwt2(original_enlarge, 'haar')
  LL, (LH, HL, HH) = coeffs

  coeffs = original2x[:,:,i]*1.9999999, (LH, HL, HH)

  #inverse wavelet transform
  trans_result[:,:,i]=pywt.idwt2(coeffs,"haar")
```

5.1.3 Local Structure Estimation

This method encompasses various different approaches, however, the one implemented is designed to be flexible, efficient and highly parralelisable. References to the developers of the original bi-directional algorithm along with detailed explanation are available in Chapter 4. However, for simplicity it can be summarised as follows: the missing HR pixels in the LR image are interpolated using arbitrary coefficients along two directions, after which the same interpolation is carried out on the four nearest neighbours. Since the nearest neighbours are LR pixels, they are available to use in calculation of the error from the chosen interpolation. This error allows the algorithm to weight the directions differently and therefore adapt to the local structure. Our workflow for this method is as follows:

- Implement naive proof of concept in Python [20]
- Experiment with other implementations
- Produce final version of potentially marketable algorithms for integrating with the Python software application
- implement final versions of algorithms in C
- Optimise the implemented C code to maximise speed

Our naive implementations based exactly on the original algorithm allowed us to provide quick results and proof of concept as well as to discover the pitfalls and potential improvements. The original algorithm is highly effective, however it requires exponent operations which can slow down the algorithm significantly. Further, the bidirectional nature of the algorithm causes unappealing artifacts (figure 5.1.3.1). Due to the simplicity of Python, we were able to easily experiment and improve the performance of the algorithm, the results of which are in chapter 7. Our implementations extending the original are:

- a four-directional implementation which removes said artifacts and can be executed more quickly due to the ability to replace the 'edge scaling factor' (described in chapter six) with an operation which removes the smallest error and increases the maximum error.
- A longer range version (interpolating over six pixels instead of four) which is able to achieve better results with marginal performance decrease since the number of memory accesses must be the same due to the use of the same data.

The next step was to implement the algorithms in C which was done by interfacing with the C code from python using the numpy [54] and Ctypes [20] libraries. The GCC [57] compiler optimisation options provided significant speedup (detailed in chapter seven). When combined with our own methods and optimisations in the python interface, eg. minimising memory accesses and allocations by referencing C-type numpy arrays only as well as our

own C optimisations regarding type sizes and mathematical precision, we achieve an astounding speedup from python in the range of 500x.

5.1.3.1 Code

The code is overly long for being included in the main body of this documentation. The Python is attached in the appendix. Both full Python and C code can be found on this project's GitHub Repository: https://github.com/jiaqige0612/VideoScaler/tree/main/Algorithms/LocalStruct.

5.1.4 Sobel Operator Based Methods

5.1.4.1 Sobel Operator Based

Firstly, a grayscale version of the input frame is produced using `cv2.cvtColor`. Then, the sobel gradients in four directions are calculated using the grayscale image and `cv2.Sobel`, as shown below. The parameters S1 and S2 are also calculated, which are used to determine the edge direction at a pixel.

```
sobel_x = cv2.Sobel(gray_frame, cv2.CV_64F, 1, 0, ksize=3)
sobel_y = cv2.Sobel(gray_frame, cv2.CV_64F, 0, 1, ksize=3)
R1 = np.sqrt((sobel_y*np.sin(0))**2+(sobel_x*np.cos(0))**2)
R2 = np.sqrt((sobel_y*np.sin(np.radians(90)))**2+(sobel_x*np.cos(np.radians(90)))**2)
R3 = np.sqrt((sobel_y*np.sin(np.radians(45)))**2+(sobel_x*np.cos(np.radians(45)))**2)
R4 = np.sqrt((sobel_y*np.sin(np.radians(135)))**2+(sobel_x*np.cos(np.radians(135)))**2)
S1 = R2/R1
S2 = R3/R4
```

The frame is expanded by 2, by creating a Numpy array of zeros and filling the even number pixels with the values corresponding to the pre-expanded frame.

```
def imgExpand(img):
    lenX = len(img[0]) * 2
    lenY = len(img) * 2
    scaled = np.zeros(shape=(lenY, lenX, 3), dtype=np.float32)

    for c in range(0, 3):
        for x in range(0, lenY):
            for y in range(0, lenX):
                if ((x % 2 == 0) and (y % 2 == 0)):
                    scaled[x][y][c] = int(img[int(x / 2)][int(y / 2)][c])

    return scaled
```

The pixels that need to be interpolated are assigned values based on the existence and direction of edge at every original pixel, as determined using the four Sobel gradients and parameters S1 and S2, as well as threshold values a and b.

```
def stage1(img):
    a = 1.6
    b = 40
    for channel in range (3):
      for x in range(2, len(img) - 2):
          for y in range(2, len(img[0]) - 2):
              if ((x % 2 == 0) and (y % 2 == 0)) and (x <= np.shape(img)[0]-2) and (y <= np.shape(img)[1]-2):
                if S1[x//2][y//2] < 1/a and R1[x//2][y//2] >b:
                  img[x+1][y][channel] = (img[x][y][channel]+img[x+2][y][channel])/2
                if S1[x//2][y//2] > a and R2[x//2][y//2] >b:
                  img[x][y-1][channel] = (img[x][y][channel]+img[x][y-2][channel])/2
                if S2[x//2][y//2] > a and R3[x//2][y//2] >b:
                  img[x+1][y-1][channel] = (img[x+2][y][channel]+img[x][y-2][channel])/2
                if S2[x//2][y//2] < 1/a and R4[x//2][y//2] >b:
                  img[x+1][y-1][channel] = (img[x][y][channel]+img[x+2][y-2][channel])/2
    return img
```

The rest of the image is filled with bilinear interpolated pixels, which are calculated using OpenCV `resize`.

```
bilinear_img = cv2.resize(frame, None, fx = 2, fy = 2, interpolation = cv2.INTER_LINEAR)

def stage2(img):
    a = 1.6
    b = 40
    for channel in range (3):
      for x in range(0, len(img)):
          for y in range(0, len(img[0])):
              if img[x][y][channel] == 0:
                img[x][y][channel] = bilinear_img[x][y][channel]
    return img
```

This algorithm is designed for a scaling factor of 2, however, scaling factors of 2's multiples can be achieved through repeating the process.

This algorithm is decided to be implemented with a series of conditionals, since it is most convenient and intuitive way of implementing the algorithm described in the paper. Nonetheless, this intuitive implementation also leaves much room for optimization.

The full code can be found on this project's GitHub Repository: https://github.com/jiaqige0612/VideoScaler/blob/main/Algorithms/SobelBased/SobelBased.ipynb.

5.1.4.2 Sobel Operator Based with Zig-Zag Suppression

This algorithm is developed based on the previous Sobel operator based method.

The grayscale version of the input frame is similarly produced. An edge map is then generated using `cv2.Canny` and the grayscale frame, with the threshold set to 0 and 200. The four Sobel gradients are calculated using the edge map instead of the original frame. Adaptive thresholds for determining the edge direction are then calculated based on the mean and standard deviation of the sobel gradient Numpy arrays.

```
Tx= (np.mean(R1)+np.std(R1))/4
Ty= (np.mean(R2)+np.std(R2))/4
T45= (np.mean(R3)+np.std(R3))/4
T135= (np.mean(R4)+np.std(R4))/4
```

The rest of the code is the same as the previous algorithm, with only the interpolation function being different. The existence and direction of edges are determined using the four Sobel gradients and the new adaptive thresholds, and will only be determined at the edge pixels from the Canny edge detector. Besides the original interpolation, more pixels around the edge pixels are modified based on the edge direction and neighbouring pixels, to suppress zig-zag artefacts.

```
def stage1(img):

    for channel in range(3):
        for x in range(2, len(img) - 2):
            for y in range(2, len(img[0]) - 2):
                if ((x % 2 == 0) and (y % 2 == 0)) and (x <= np.shape(img)[0] - 2) and (y <= np.shape(img)[1] - 2):
                    if sharp[x//2][y//2] == 255:

                      if R1[x // 2][y // 2] >= R2[x // 2][y // 2] and R1[x // 2][y // 2] >= Tx:
                        #x direction
                        #sobel based interpolation

                          img[x + 1][y][channel] = (img[x][y][channel] + img[x + 2][y][channel]) / 2

                        #zig zag suppression

                          weight_x = img[x][y - 1][channel] / (img[x][y - 1][channel] + img[x][y - 2][channel] + 1)
                          img[x][y-1][channel] = weight_x * img[x][y-1][channel] + (1- weight_x) * img[x][y-2][channel]

                          weight_x = img[x][y + 1][channel] / (img[x][y + 1][channel] + img[x][y + 2][channel] + 1)
                          img[x][y+1][channel] = weight_x * img[x][y+1][channel] + (1- weight_x) * img[x][y+2][channel]

                      if R2[x // 2][y // 2] > R1[x // 2][y // 2] and R2[x // 2][y // 2] >= Ty:
                        #y direction
                          img[x][y - 1][channel] = (img[x][y][channel] + img[x][y - 2][channel]) / 2

                          weight_y = img[x-1][y][channel] / (img[x-1][y][channel] + img[x-2][y][channel] + 1)
                          img[x-1][y][channel] = weight_y * img[x-1][y][channel] + (1- weight_y) * img[x-2][y][channel]

                          weight_y = img[x+1][y][channel] / (img[x+1][y][channel] + img[x+2][y][channel] + 1)
                          img[x+1][y][channel] = weight_y * img[x+1][y][channel] + (1- weight_y) * img[x+2][y][channel]

                      if R3[x // 2][y // 2] >= R4[x // 2][y // 2] and R3[x // 2][y // 2] >= T45:
                        #45 degree direction
                          img[x + 1][y - 1][channel] = (img[x + 2][y][channel] + img[x][y - 2][channel]) / 2

                          weight_45 = img[x-1][y - 1][channel] / (img[x-1][y - 1][channel] + img[x-2][y - 2][channel] + 1)
                          img[x-1][y-1][channel] = weight_45 * img[x-1][y-1][channel] + (1- weight_45) * img[x-2][y-2][channel]

                          weight_45 = img[x+1][y + 1][channel] / (img[x+1][y + 1][channel] + img[x+2][y + 2][channel] + 1)
                          img[x+1][y+1][channel] = weight_45 * img[x+1][y+1][channel] + (1- weight_45) * img[x+2][y+2][channel]
                      if R4[x // 2][y // 2] > R3[x // 2][y // 2] and R4[x // 2][y // 2] >= T135:
                        #135 degree direction
                          img[x + 1][y - 1][channel] = (img[x][y][channel] + img[x + 2][y - 2][channel]) / 2

                          weight_135 = img[x-1][y +1][channel] / (img[x-1][y +1][channel] + img[x-2][y + 2][channel] + 1)
                          img[x-1][y+1][channel] = weight_135 * img[x-1][y+1][channel] + (1- weight_135) * img[x-2][y+2][channel]

                          weight_135 = img[x+1][y - 1][channel] / (img[x+1][y - 1][channel] + img[x+2][y - 2][channel] + 1)
                          img[x+1][y-1][channel] = weight_135 * img[x+1][y-1][channel] + (1- weight_135) * img[x+2][y-2][channel]


    return img
```

This algorithm is designed for a scaling factor of 2, however, scaling factors of 2's multiples can be achieved through repeating the process.

This algorithm is also decided to be implemented with a series of conditionals, since it is most convenient and intuitive way of implementing the algorithm described in the paper. Nonetheless, this intuitive implementation also leaves much room for optimization.

The full code can be found on this project's GitHub Repository: https://github.com/jiaqige0612/VideoScaler/blob/main/Algorithms/SobelBased/SobelBased.ipynb.

5.1.5 NEDI (New Edge-Directed Interpolation)

The NEDI algorithm is a resolution enhancement technique which uses the local covariance coefficients calculated by using training windows in the low-resolution image to predict the high-resolution sub-pixel values.

The implementation process of the code can be described as follows:

1. Initialization and Input: The function takes a color image as input and defines certain parameters for the window size (m) and the scale size (s). The window size refers to the size of the pixel neighborhood used for interpolation, and the scale size defines how much the image size will be increased.

2. Edge Detection: The Canny edge detection method is used to identify the edges in the grayscale version of the input image. The output of this operation is a binary image (sharp), where the pixel value is 1 if that location is an edge and 0 otherwise. This binary image is used to determine which pixels are interpolated in later steps.

3. Low-Resolution Pixels Insertion: The image is upsampled by a factor of 2 using bicubic interpolation to get an initial high-resolution image (imgo). However, only the pixels at locations that are not edges are well interpolated, while the others are not.

4. High-Resolution Image Enhancement: The main process of the NEDI algorithm is implemented in three steps that each interpolate a certain type of pixel location: (2i+1,2j+1), (2i+1,2j), and (2i,2j+1). In each step, a loop goes through each pixel in the low-resolution image and checks if the corresponding location in the binary image (sharp) is an edge (value of 1). If it is, the algorithm proceeds with the following steps:

a. Local covariance estimation: For the neighborhood of each pixel, the covariance is estimated among the four nearest known pixels (2*ii-2,2*jj-2),(2*ii+2,2*jj-2),(2*ii+2,2*jj+2), and (2*ii-2,2*jj+2).

b. Covariance-adjusted interpolation: The local covariance is used to adjust the interpolation of the new pixel. This interpolation is based on a linear combination of the four nearest known pixels, with weights given by the inverse of the covariance matrix. These steps are performed for each color channel (k).

After these steps, the function returns the enhanced high-resolution image (imgo). The overall effect of the NEDI algorithm is to increase the resolution of an image by creating new pixel values that are consistent with the existing pixel values and the local edge directions.

In the analysis of the implemented algorithm, it is essential to note that the approach utilized is not a pure

embodiment of the New Edge-Directed Interpolation (NEDI) method. The core modification in the applied technique involves the selective application of NEDI only on the pixels identified as edges, as opposed to the conventional practice of deploying it on every pixel in the frame. This modification is done creatively to reduce the processing time and avoid the NEDI producing overly smooth textures.

This targeted application not only optimizes computational efficiency but also significantly alleviates the burdensome operation time traditionally associated with NEDI. Particularly, the matrix inversion operation, a critical computational step in the NEDI algorithm, is known for its intensive computational costs. Conducting this operation on each pixel indiscriminately results in a substantial increase in computational load, leading to a decelerated process speed.

Moreover, it is pertinent to mention that the interpolation method employed in this code is flexible and can be replaced with alternative strategies, such as bilinear or Lanczos interpolation. While the bicubic interpolation is used in the present scenario, the choice of interpolation method can be adjusted based on the specific requirements of the image processing task or the desired balance between computational efficiency and output resolution quality. This makes our approach versatile and adaptable for diverse image upscaling needs.

The MATLAB function code is overly long for presenting here and is shown in the appendix. It doubles the height and width of the incoming frame or image when it executes.

The full code can be found on this project's GitHub Repository: https://github.com/jiaqige0612/VideoScaler/tree/main/Algorithms/NEDI.

5.1.6 Deep learning ESRGAN

The ESRGAN for Video Super-resolution takes advantage of the open source pre-trained model from Tensorflow hub. Modify the code to satisfy the requirement of capturing the frames from videos and input these images into the model, and then generate the videos from the super-resolution frames. Here is the example code of the programme:

```
os.environ["TFHUB_DOWNLOAD_PROGRESS"] = "True"

SAVED_MODEL_PATH = "https://tfhub.dev/captain-pool/esrgan-tf2/1"

def preprocess_image(image_path):
  """ Loads image from path and preprocesses to make it model ready
      Args:
        image_path: Path to the image file
  """
  hr_image = tf.image.decode_image(tf.io.read_file(image_path))
  # If PNG, remove the alpha channel. The model only supports
```

```
  # images with 3 color channels.
  if hr_image.shape[-1] == 4:
    hr_image = hr_image[...,:-1]
  hr_size = (tf.convert_to_tensor(hr_image.shape[:-1]))
  hr_image = tf.image.crop_to_bounding_box(hr_image, 0, 0, hr_size[0], hr_size[1])
  hr_image = tf.cast(hr_image, tf.float32)
  return tf.expand_dims(hr_image, 0)


model = hub.load(SAVED_MODEL_PATH)

video_path    = 'path to videos'
output_path   = 'path of saving videos'

frame_path    = 'path to the frames captured from videos'
sr_frame_path = 'path to the super-resolution frames'

#frames per second
fps = 30

#make directory for frames as they are temp file
if not os.path.exists(frame_path):
  os.makedirs(frame_path)

if not os.path.exists(sr_frame_path):
  os.makedirs(sr_frame_path)

capture_frames(video_path,frame_path)

# input all the frames into the model
for i in range(0, num(frames)):
    lr_framepath = str(i)+'.png'
    tmp_path = os.path.join(frame_path,lr_framepath)
    tmp_image = preprocess_image(tmp_path)
```

```
    sr_image = model(tmp_image)
    sr_image = tf.squeeze(sr_image)
    sr_frame_name = f"{i}"
    sr_save_path = os.path.join(sr_frame_path,sr_frame_name)
    print('frame '+ str(i) + 'done')
    save_image(sr_image,sr_save_path)

create_video_from_frames(sr_frame_path,output_path,fps)
```

The model involves two two key components: a generator and a discriminator, which are trained simultaneously in an adversarial manner [22]. The generator aims to produce the high-resolution images from the low-resolution images, while the discriminator attempts to distinguish between the generated HR-images and the real HR-images. The GANs model can be applied in lots of image processing tasks, such as text to image synthesis [74],image denoise [8, 72], image to image translation [12], image super-resolution [42] and etc. In the field of super-resolution, faster and deeper convolutional neural networks have already resulted well in both speed and accuracy [17, 38]. However, the issue of keeping high frequency information and containing the fine texture still exists. GANs have revolutionized the field of by enabling the generation of realistic and visually pleasing high-resolution images, and here SRGAN is introduced. There is two key components of SRGAN which make the model work better at containing the texture of images, the loss function and architecture. The architecture of the model is designed as following figure

The upper part is the generator component, which will produce the high quality reconstructed images. The core of the network of generator is the residual block layout, which includes two convolutional layers with small 3×3 kernels and 64 feature maps. The residual connection enable the model to keep the information from different dimension, known as the texture. The resolution of input images are also increased by the sub-pixel convolutional layer by Shi et al. [60] The discriminator is defined following the guide summarized by Radford et al. [59] The resulting 512 feature maps are followed by two dense layers and a final sigmoid activation function to obtain a probability for sample classification.

The most common pixel-wise loss function such as mean square error function are difficult to recover the texture as they lead to pixel-wise optimal solution. Typically, the images are over-smooth thus have lower perceptual quality [19, 36, 51]. According to Johnson et al. and Bruna et al. [4, 36] a new loss function is defined based on perceptual similarity:

$$l^{SR} = l_X^{SR} + 10^{-3} l_{GEN}^{SR}$$

The perceptual loss is formulated as the weighted sum of content loss(l_X^{SR}) and adversarial loss($10^{-3} l_{GEN}^{SR}$). The content loss is defined according to the comparison between the result after convolutional layers and the original HR

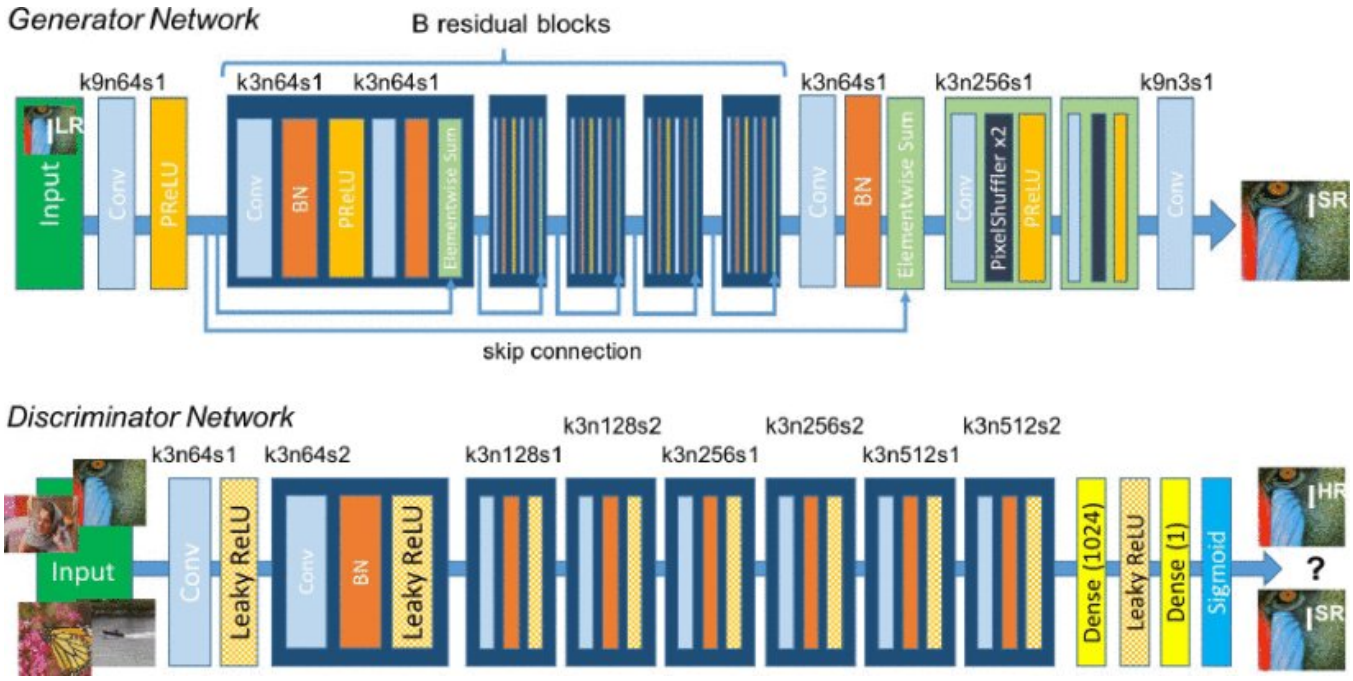


Figure 5.1: SRGAN Architecture

images. The adversarial loss is defined based on the probabilities of that discriminator regards the re-constructed images as natural images.

The full code can be found on this project's GitHub Repository: https://github.com/jiaqige0612/VideoScaler/tree/main/Algorithms/ESRGAN.

5.2 Front-end Development: Graphical User Interface (Jiaqi)

Our innovation was also evident in our GUI design, which was specifically created for code testing and demonstration purposes. With this user-friendly interface, we could simplify the testing process, making it accessible not just for our team, but potentially for our client as well. It allows users to easily utilize the implemented video upscaling algorithms to super-resolve videos with scaling ratios of their choices. The GUI design showcased our creativity and ingenuity by integrating a human-centric design perspective into a highly technical project. The GUI was created in Python using open-source library PySimpleGUI [58]. The license is attached in the appendix.

The GUI has a configuration shown in figure 5.2.

The GUI consists of two columns. The left hand side column was designed to present information about the original video and contain elements that allow users to adjust upscaling algorithm parameters. The right hand side column was designed to present the upscaled video frame. This structure was designed considering that most people

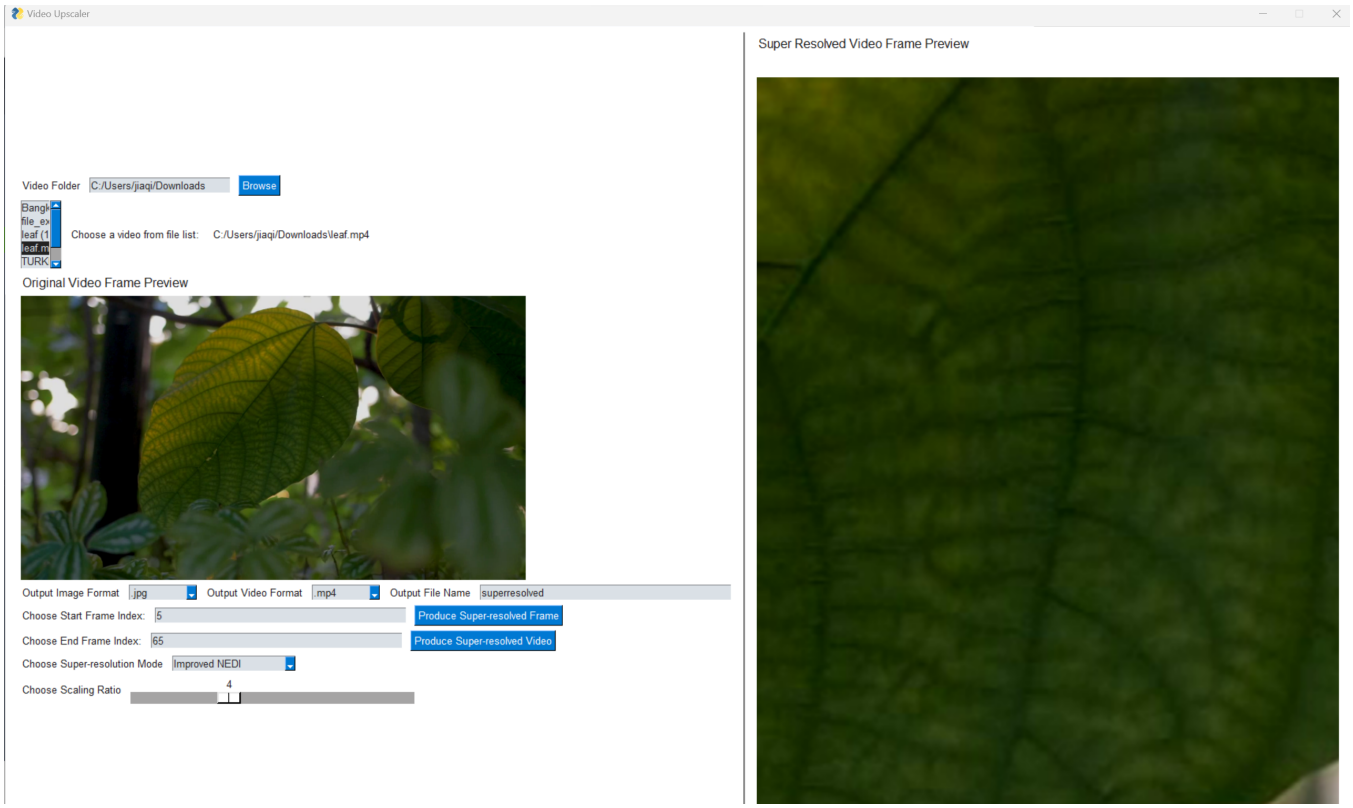


Figure 5.2: GUI Configuration

read and look from left to right, so it is more logical and natural to have pre-processing information on the left and post-processing result on the right.

On the left hand side, the upper section was designed for user to choose folder path and select video to be upscaled from a list box, as well as a text box displaying the chosen video file name. The middle area was designed for displaying a preview of the frame corresponding to the start frame index of the selected video. The bottom section was designed for users to adjust parameters for the upscaling algorithms and the properties of output file. The order of these three sections was also designed for the most logical and natural experience of the users, who are most likely to read from top to bottom and would like to select file first, then have a preview, finally decide upscaling settings and start super-resolution.

The adjustable parameters include start frame index; end frame index if the user chooses to produce super-resolved video instead of frame by clicking the button on the right; super-resolution mode, which can be chosen from the five selected video upscaling algorithms; and scaling ratio, which can be adjusted between 2x, 4x, 6x and 8x. The output file can be changed to any name that the user input to the text box; its format can be changed between JPG, PNG and BMP for images, between MP4, AVI and MOV for videos, through the drop-down lists. All the adjustable

settings have default values, which are shown below:

- Start Frame Index: 0
- End Frame Index: 100
- Super-resolution Mode: Bicubic
- Scaling Ratio: 2x
- Output Image Format: PNG
- Output Image Default Name: savedImage
- Output Video Format: AVI
- Output Video Default Name: savedVideo

The default values were designed to make sure that the software application will not crash or have unexpected behaviours when users forget to set up something.

The full code can be found on this project's GitHub Repository: https://github.com/jiaqige0612/VideoScaler/blob/main/GUI/mainfinal.py.

5.3 Integrating Algorithms with GUI

The 5 chosen algorithms, which are bicubic, DTCWT Lanczos, local structure etsimation, improved NEDI and ESRGAN are integrated with the GUI to produce the final application software. The bicubic, DTCWT Lanczos and local structure estimation can all be readily imported and integrated with the front-end; the image data is passed between the back-end and front-end in form of a 3 channel Numpy array, which are produced using OpenCV; to output a video, every frame within the range defined by start and end frame index is upscaled by the chosen method and outputs an image named using its frame index, the images are converted at the end to video using OpenCV's `VideoWriter` object in order of frame index and deleted. Naming by frame index ensures that the order of frames in the output video is correct. This is arguably not the most efficient method of producing video using the upscaling algorithms, but is the most reliable one among all the implementation being contrived and attempted and is thus adopted as the final implementation.

The improved NEDI is implemented in MATLAB [31]. It was attempted to replicate the algorithm in Python, nonetheless, the results are not optimal. Therefore, it was decided to use the MATLAB Engine API to allow the Python application to call MATLAB as a computational engine [30]. The input frame read by the front-end is saved as a PNG image, which is then read by the MATLAB Program. The output frame is saved by MATLAB program

as another PNG image and is read by the front-end. Other than that, it is intergated with the front-end the same way as the other three methods described above.

The ESRGAN utilizes mostly TensorFlow and process data in terms of tensors, which are not directly compatible with the front-end and other four algorithms. Thus, it is treated separately. For outputting image, it was designed to read the image as Numpy array from the front-end, then it is converted to a 4-dimensional tensor; the output result from the ESRGAN is saved in a temporary file, which is then read again by the front-end and saved with user-chosen name and format. For outputting video, the ESRGAN module was designed to read the video directly from the chosen path, then directly save the super-resolved video to the user-chosen path and with the chosen format, without much communication with the front-end in the process.

The full code of the developed video upscaling software application can be found on this project's GitHub repository https://github.com/jiaqige0612/VideoScaler/tree/main/Final%20Application%20Software.

Chapter 6

Final Built Design Specifications

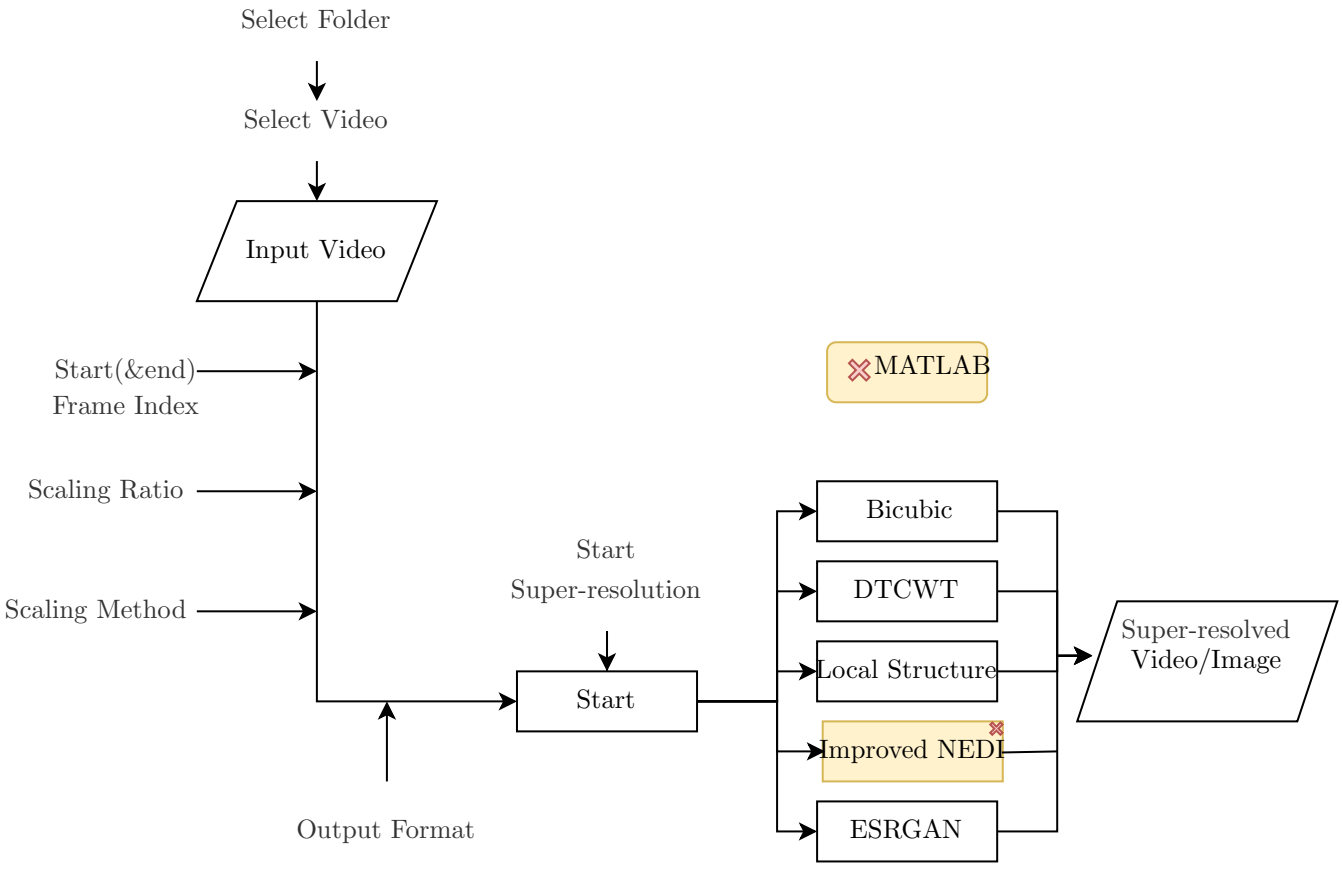


Figure 6.1: Flowchart for the Video Upscaler Application Software

6.1 Functions and Features

The edge adaptive video up-scaler is an application software that upscales videos with 5 encapsulated super-resolution algorithm, with varied cost and performance. The 5 algorithms are bicubic interpolation, local structure, DTCWT, Improved NEDI and ESRGAN. The software can support up-scaling of 2x, 4x, 6x and 8x. The supported input and output video formats include MP4, AVI and MOV; supported output image formats include PNG, JPG and BMP.

In the user interface, the software first takes the chosen video in the direct folder user picked and read in. There is an section, in which the user can choose desired start and end video frame index, the scaling ratio and pick one of the five super-resolution methods which will be performed on the video, as well as the output formats. All the parameters have default values that prevent the application from crashing if the user does not input anything. After the user press "Produce Super-resolved Video" or "Produce Super-resolved Frame", a reminder window would pop up to notify the user that the video is being processed. The frame data is passed from the front-end to the back-end algorithm modules; the upscaled frame data is passed back to the front-end, for outputting directly or converting to a video with other frames. After the upscaling process is completed, a reminder window would pop up, indicating the directory in which super-resolved video or image has been saved and showing a preview of the result. The process flowchart for the video upscaler application software is shown in figure 6.1.

6.2 System Requirements

Using the edge adaptive video upscaler application software requires a local installation of Python and MATLAB. The application is designed for running on Windows.

6.3 Intended Users

Our project focuses on video and image upscaling and we have implemented various methods into a user interface. Intel is the client that starts the whole project and is considered as the main target. Despite our supported company, the video upscaler software application is also designed for individuals or professionals involved in media production, content creation, or digital imaging tasks. For example, video, image editors or graphic designers may find our product available as they often encounter the need to upscale content for various purposes, such as adapting lower-resolution footage to higher-resolution formats, enhancing image quality or improving the resolution and quality of graphics, illustrations, and other visual elements. Photographer or even general user who have interest and wish to improve the quality of their personal photos and videos can also get what they need with the help of our product. The range of varied cost and performance of the encapsulated upscaling algorithms, with the flexibility provided by the several adjustable settings, should satisfy the needs of both professional and individual users.

We do our best to create a good user experience as our first priority, ensuring ease of use, accessibility, and user-friendly design in the GUI to accommodate both professional users and those with limited technical expertise.

6.4 Evaluation Metrics and Performance considerations

There are several common performance measurement scales that are used to evaluate the quality and effectiveness of up-scaling algorithms and of the application software. In this project, PSNR(Peak Signal-to-Noise Ratio) and SSIM(Structural Similarity Index) metrics are used.

PSNR measures the ratio between the maximum possible power of a signal (the original image) and the power of the noise (the difference between the original and up-scaled images). While SSIM compares the structural information of the original and up-scaled images. It takes brightness, contrast, and structural similarity into account and ranges from -1 to 1, with 1 indicating identical images. The higher these two scales are usually indicates the better of image similarities and performance the method is. However, it is possible to have algorithms which achieve high scores on objective metrics but still produce visually unsatisfactory results.

Other scales such as execution time, as well as estimated number of ALM (Adaptive Logic Module) needed for implementing the algorithms on FPGA are also taken into consideration.

Chapter 7

Design Evaluation

7.1 Testing Results on Images

Since all the algorithms implemented are single-frame based, the evaluation is carried out on images first, which is used to determine the algorithms being encapsulated in the application software, according to the performance of each algorithm.

The evaluation on Images is based on three different aspects of each algorithms, including the image quality, execution time. The quality of the images is defined as the similarity between the produced super-resolution images and the original high-resolution images. There are two major methods of measuring it, the peak signal noise ratio(PSNR) and the structural similarity index measure(SSIM), which represent the quality of the images better than usual method like mean square error(MSE).

Here is the pseudo code of loading the images produced by the different algorithms and comparing with the original one. The score is proportional to the quality of the images.

```
from skimage.metrics import structural _similareity
# loading the image into array
SR_image = image.read(super-resolution image path)
HR_image = image.read(original high-resolution image path)
#call the function imported to calculate SSIM
(score, diff) = structural_similarity(SR_image, HR_image, sigma=1.5, data_range=255)
SSIM = score

#calculate PSNR
```

```
mse = mean((SR_image-HR_image)**2)
PSNR =  20*log10(255/sqrt(mse))
```

The images are chosen from DIV2K [2] to test all the algorithms implemented in this project. The following tables 7.1, 7.2 and 7.3 show results of each algorithms. Evaluation and analysis in this documentation is carried out primarily using one of the tested images with enlarged details to observe the difference at the edges.

Method	PSNR(dB)	SSIM(%)	Software Execution Time(s)
Bicubic	31.57	92.84	**0.303**
Lanczos	31.69	92.97	16.876
Wavelet Zero Padding	28.16	86.67	0.523
DWT with SWT	21.63	60.61	1.264
DWT with NEDI	30.76	91.07	740.177
DT-CWT with Lanczos	30.39	90.38	49.279
NEDI	30.26	94.78	136.104
NEDI with Canny edge and Bicubic Interpolation	31.04	**95.49**	3.331
Local Structure	26.16	88.89	0.712
Local Structure (YCbCr)	26.33	90.52	0.432
Sobel Based Interpolation	28.84	90.69	53.561
Sobel Based Interpolation with Zig-Zag Suppression	27.16	89.17	65.04
ESRGAN	**32.95**	91.10	9.437

Table 7.1: Results of Implemented Algorithms (Image 0882)

Method	PSNR(dB)	SSIM(%)	Software Execution Time(s)
Bicubic	27.82	79.22	**0.225**
Lanczos	27.95	79.81	17.113
Wavelet Zero Padding	27.18	76.30	0.541
DWT with SWT	21.396	47.38	0.805
DWT with NEDI	27.36	76.79	737.822
DT-CWT with Lanczos	27.14	75.75	64.881
NEDI	26.30	75.30	129.88
NEDI with Canny edge and Bicubic Interpolation	27.33	79.96	6.317
Local Structure	24.21	68.18	0.342
Local Structure (YCbCr)	27.55	73.34	0.223
Sobel Based Interpolation	25.95	72.05	56.145
Sobel Based Interpolation with Zig-Zag Suppression	24.18	65.60	92.612
ESRGAN	**28.67**	**83.17**	10.30

Table 7.2: Results of Implemented Algorithms (Image 0801)

The input to the algorithms in this testing are low resolution images obtained using Matlab imresize function with default settings and a downscale factor of 4. The output results are compared with original high resolution images to calculate PSNR and SSIM as mentioned before. The execution time is also measured to give an indication of the computational cost. All the testings have been done on Google Colab.

Figure 7.1 is the image 0882 from dataset DIV2K [2] and one of the image tested. Other detailed cropped images in figure 7.2, figure 7.3, figure 7.4 and figure 7.5 showcase the performance of each algorithm.

As shown in figure 7.2 and table 7.1, the bicubic interpolation and Lanczos interpolation have achieved relatively

Figure 7.1: Original High Resolution Image 0882 from DIV2K

((a)) Original

((b)) Bicubic Interpolation

((c)) Lanczos Interpolation

Figure 7.2: Original, Bicubic and Lanczos Interpolation

Method	PSNR(dB)	SSIM(%)	Software Execution Time(s)
Bicubic	31.87	89.27	**0.22**
Lanczos	32.01	89.56	16.776
Wavelet Zero Padding	29.11	81.96	0.436
DWT with SWT	23.49	57.07	0.922
DWT with NEDI	31.22	87.49	735.126
DT-CWT with Lanczos	30.94	86.83	51.66
NEDI	30.51	93.89	71.43
NEDI with Canny edge and Bicubic Interpolation	31.38	**94.89**	4.455
Local Structure	27.03	83.18	0.378
Local Structure (YCbCr)	29.83	92.32	0.188
Sobel Based Interpolation	29.45	85.57	57.287
Sobel Based Interpolation with Zig-Zag Suppression	29.50	85.72	56.387
ESRGAN	**32.95**	90.31	8.656

Table 7.3: Results of Implemented Algorithms (Image 0815)

high values of PSNR and SSIM, which are respectively 31.57 dB, 92.84% and 31.69 dB, 92.97%. The performance of these two algorithms are very similar to each other, with the bicubic being much faster in software execution time and the lanczos being slight better in terms of performance. Despite having higher PSNR than all implemented algorithms except the ESRGAN and having higher SSIM than all implemented algorithms except NEDI and NEDI with Canny edge detection and bicubic interpolation, the visual quality of images upscaled by these two interpolation algorithms is not very good, with apparently blurred edges and textures.

For the wavelet based methods, the results recorded in table 7.1 and shown in figure 7.3 are only the results obtained using the most basic wavelet, which is the haar wavelet for DWT and biorthogonal 1.3 wavelet for DT-CWT. Since wavelet transform with other wavelets produce images with slightly different dimensions compared to the original one, making it impossible to make meaningful comparison with the original without extra resizing. To better evaluate the performance of wavelet based algorithms, super-resolved images have also been produced using the same methods but with CDF 9/7 wavelet for DWT, also known as biorthogonal 4, 4 wavelet, which allows for better image reconstruction than other conventional wavelets [41]; and with Antonini 9/7 wavelet for DT-CWT. The execution time for implementation with CDF 9/7 and Antonini 9/7 wavelet is approximately the same and omitted for presentation in this review.

As a result of the simple wavelet used, the output images shown in 7.3 have clear pixelated texture and slightly lower PSNR and SSIM even when compared with bicubic and lanczos interpolation, as shown in table 7.1. This is particularly the case for the wavelet zero padding method, as shown in figure 7.3. The DWT with SWT method also produces unsatisfactory upscaled image, which have blurred edges and textures. Nonetheless, for DWT with NEDI and DTCWT with Lanczos, it is clear from the output images shown in 7.3 and 7.4 that they produce more distinct and sharper edges compared to bicubic and lanczos interpolation. The textures, except the pixelated appearance due to usage of very basic wavelets, are maintained well and realistic, unlike the NEDI that over-smooths the edges and results in drawing-like textures. These methods also do not produce zig-zag or fuzzy artefacts like the sobel based

method and the sobel based method with zig-zag suppression. Moreover, the image super-resolved by wavelet zero padding with CDF 9/7 wavelet have much less pixelated texture and better visual quality compared to wavelet zero padding result with haar wavelet, which illustrates the impact of exact wavelet used on the algorithm performance.

The execution time in software for wavelet-based methods is longer than bicubic and lanczos interpolation, but comparable to all other adaptive interpolation methods. In the process of executing these algorithms, significantly much more time is taken for the interpolation of sub-bands, rather than the wavelet transforms, partcularly for the case of DWT with NEDI. Nonetheless, when implemented in FPGA, the wavelet transform may takes a much longer time. Further investigation should be made to evaluate wavelet-based methods, when implementing with different wavelets and in FPGA.

((a)) Wavelet Zero Padding ((b)) DWT with SWT ((c)) DWT with NEDI ((d)) DTCWT with Lanczos

Figure 7.3: Wavelet Based Methods with Haar and Biorthogonal 1.3 Wavelet

((a)) Wavelet Zero Padding ((b)) DWT with SWT ((c)) DWT with NEDI ((d)) DTCWT with Lanczos

Figure 7.4: Wavelet Based Methods with CDF 9/7 and Antonini 9/7 wavelet

The Sobel based method takes a relatively little amount of time to execute in software, while the local structure execution takes way too long. As shown in figure 7.5, the Sobel based method does preserve the edges relatively well, yet it produces a lot of zig-zag artefacts along the edges, resulting in relatively low PSNR and SSIM. Using zig-zag suppression method can indeed reduce zig-zag effects and make the output image smoother, however, the

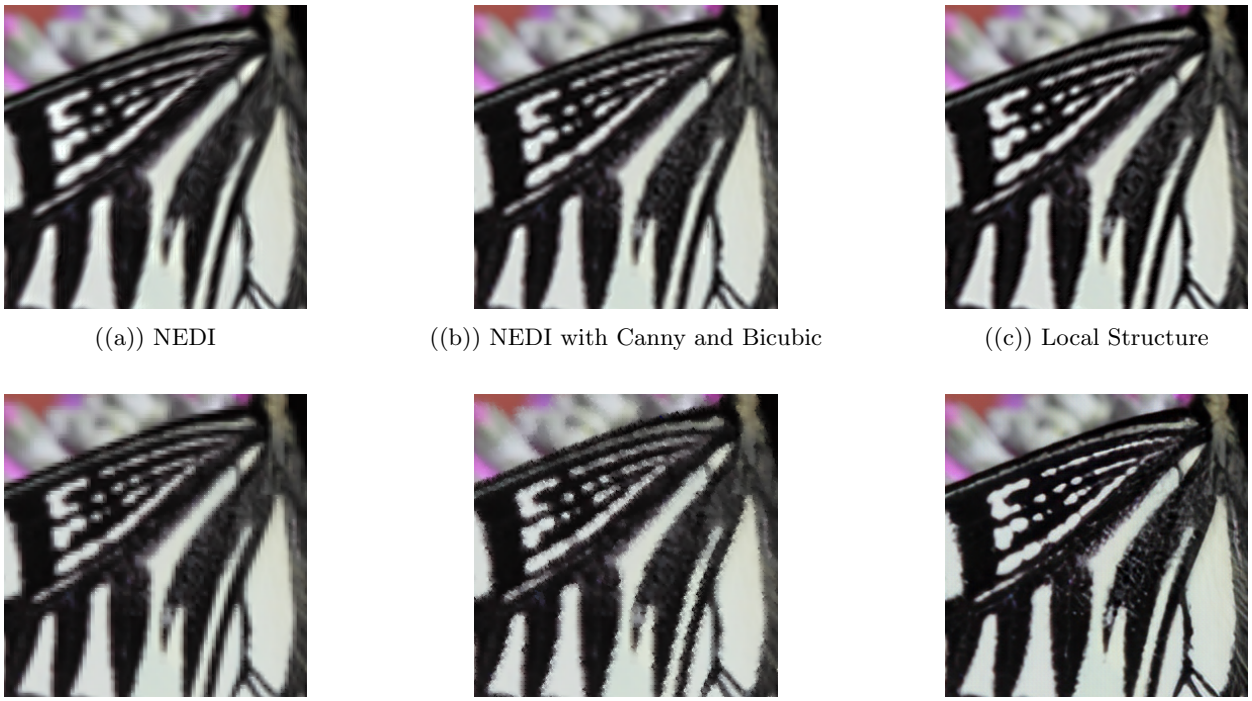
output images can be over-smoothed, resulting in blurry and unrealistic textures, also causing lower PSNR and SSIM. Changing and picking the suitable canny edge threshold in this combined process could ideally improve the results, yet a method for determining the optimum threshold should be developed.

The NEDI method shown in figure 7.5, by virtue of its edge-directed approach, prioritizes the preservation of high-frequency details such as edges and textures during the upscaling process. When applied to video upscaling, it tends to provide a visually pleasing output with sharp edges and detailed textures. By changing the window size in the NEDI code, we can get slightly different results. Smaller the window size, stronger the contrast at the edges; larger the window size, smoother the edges. However NEDI algorithm requires quite intense computation due to the inverse matrix computation applied on each pixel. To solve this problem NEDI with Canny and Bicubic is implemented. NEDI algorithm is only applied onto pixels detected as edges by Canny edge detection, while the rest of the input image is resized with bicubic interpolation. In this way a balance of edge quality and computation speed is achieved.

As the Figure 7.5 shows, the quality of image super-resolved by ESRGAN is superior compared to other algorithms implemented, with accurate and sharp edge as reflected from the highest PSNR and visual quality. However, similar to all other algorithms' results, the texture of the pollen at the centre of the image is still missing. Besides, the model takes about 10 seconds to process the image; it also takes an enormous amount of time to train the model. It needs further optimisation and utilization of parallel computing for acceleration, particularly when implemented on FPGA.

The Local structure estimation method produces poor quantitative but good qualitative results, as shown in table 7.1 and figure 7.5. The poor quantitative performance can be attributed to the fact that quantitative metrics such as PSNR and SSIM do not take into account things such as outliers which can skew the result significantly, particularly PSNR. Furthermore, note that in figure 7.6, where we detail the differences between V1 and V2 of the algorithm, it is clear that V1 does produce some displeasing zig-zag-like artifacts which are not visible in V2. Despite this, due to simplicity and the fact that these artifacts are only visible at an extremely low resolution we implement V1 as our primary local structure estimation method. Beyond this, when evaluating the method on the luma component of the YCrBr colourspace, the algorithm performs significantly better and faster. This is likely due to the fact that most of the significant information in the image is contained within this component and therefore by resolving only this component, errors are minimised.

Based on the evaluation of the testing results on images, considering a balance between computational cost and performance, five super-resolution algorithms have been selected to be encapsulated into the upscaler application software. The 5 algorithms are bicubic interpolation, local structure, DTCWT Lanczos, NEDI with Canny edge detection and bicubic interpolation and ESRGAN.



((a)) NEDI ((b)) NEDI with Canny and Bicubic ((c)) Local Structure

((d)) Sobel Based ((e)) Sobel Based Zig-Zag Suppression ((f)) ESRGAN

Figure 7.5: Adaptation Based on Local Information Interpolation and ESRGAN

((a)) Original texture ((b)) V1 ((c)) V2

Figure 7.6: Comparison between V1 and V2 Local Structure Estimation

7.2 Testing Results on Videos

The procedure of testing and evaluating algorithms' performance on videos is similar to the testing on images; it is essentially applying the same code to every frame captured from the videos. We choose a video of 2 seconds which includes 60 frames in total. The low-resolution video, which is down-sampled by 4 from the original using Pillow's bicubic resize, will be input into each of the chosen algorithms using the software application, the 4x upscaled output will be compared with the original high-resolution video frame by frame. The average PSNR, SSIM and the execution time per frame over the video will be calculated, using the same code as the testing on images. The number of adaptive logic modules (ALM) needed to implement the algorithms on FPGA is also estimated to show the computational cost.

Figure 7.7: One Random Frame of Original High Resolution London Video (1920x1080)

Figure 7.8: One Random Frame of Original High Resolution Las Vegas Video (3840x2160)

Algorithms	Avg PSNR(dB)	Avg SSIM(%)	Avg Exe Time per Frame (s)	Estimated ALM Usage
Bicubic	26.17	63.98	**0.0018**	**10,000**
Local Structure	23.36	52.75	0.185	25,000
DTCWT Lanczos	25.84	60.70	30.57	50,000
Improved NEDI	25.83	62.31	8.868	25,000
ESRGAN	**27.8539**	**83.03**	0.997	200,000

Table 7.4: Evaluation Based on Video Testing (London Video) (1920x1080)

Algorithms	Avg PSNR(dB)	Avg SSIM(%)	Avg Exe Time per Frame (s)	Estimated ALM Usage
Bicubic	27.26	78.61	**0.0050**	**10,000**
Local Structure	-	-	-	25,000
DTCWT Lanczos	26.83	76.07	121.60	50,000
Improved NEDI	26.75	77.04	19.75	25,000
ESRGAN	**28.2529**	**86.46**	4.205	200,000

Table 7.5: Evaluation Based on Video Testing (Las Vegas Video) (3840x2160)

As shown from table 7.4, when upscaling video instead of images, all the algorithms' performance reduce significantly. When comparing with the testing results for image 0882 from DIV2K data set, the PSNR is lowered by 4.61 dB on average for all the chosen algorithms. The least decrease is 2.80 dB for local structure based method, and the greatest decrease is 5.40 dB for bicubic interpolation. The SSIM is lowered by 27.19% on average for all the chosen algorithms. The least decrease is 8.07% for ESRGAN, and the greatest decrease is 36.14% for local structure based method. This decrease in PSNR and SSIM, on one hand, is caused by the inevitable existence of blurred frames in videos due to motion, which makes it difficult for edge adaptive algorithms to explicitly or implicitly detect and correctly preserve edges. On the other hand, the original HR video used only has 1080p resolution, which makes the 4x downscaled LR video unable to preserve enough edges and textures, for upscaling algorithms to work with. This is proven in the testing with the second video shown in figure 7.8, which has a higher resolution and consequently all the algorithms have considerably better performance in terms of PSNR, SSIM and visual quality, with an inevitably longer execution time. (The testing with local structure is omitted due to the overly long processing time with the higher resolution video.)

The execution time per frame for video is considerably lowered compared to the execution time for single image. The execution time is lowered by 167.6 seconds on average for all the chosen algorithms. This reduction could be caused by the usage of different platform, since the testing on video is done on local desktop through the application software whereas the testing on images is done on Google Colab; this reduction could also be caused by the fact that memory allocation is needed less frequently when processing videos compared to images, due to the repeated usage of same functions and variables, as well as due to the temporal coherence of video frames that causes repetition of data representing the frames. For improved NEDI, the execution time increases by 5.53 seconds. This is caused because in image testing, the improved NEDI is tested directly in MATLAB; in video testing, it involves extra stages

of reading and writing images and communicating between Python application and MATLAB program.

Overall, it is ESRGAN performing the best on upscaling videos and the execution time will be much shorter than working on single image because that the model will learn the feature of the frames, so that the similar frames can be calculated faster. However, ESRGAN has the higher computational cost (ALM usage) of all algorithms, when implemented in FPGA.

Since this testing is done using the software application, the testing results also confirm the proper operation and functionality of the developed video upscaler software application. Both the image testing and video testing results can be found on this project's GitHub repository: https://github.com/jiaqige0612/VideoScaler/tree/main/Test%20Results.

Chapter 8

Sustainability Report

8.1 Environmental Sustainability

The nature of a software toolkit is that it does not require the use of external hardware. However there is the energy consumption cost. This cost can vary significantly between machines and whatever hardware accelerators are available. However, our implementation provides the user with a range of algorithms each with a different balance between energy consumption and effectiveness. Furthermore, our algorithms are either optimised already or we have implemented them in C and optimised them maximally in order to make them as efficient as possible. By doing this we minimise the power consumption of the device running our algorithms thereby minimising both the environmental impact and running cost of our product.

8.2 Social and Economic Sustainability

We consider our customers: our core interface runs on python which can run natively on a wide array of machines. Moreover, while our C implementations must be recompiled on whatever machine they are run, they are able to be compiled by a range of compilers: GCC and LLVM. By supporting multiple compilers, we decrease the potential that our users will not have the required software to run our toolkit. By having an easy to use system which has few requirements, we provide the utility of our product to as many people as possible. This allows our product to benefit people in different social classes and from different backgrounds and provide them equal opportunity of processing their videos and enjoying better viewing experience, reducing technology disparities and promoting social equity. Additionally, the customer base of our software application is also expanded in this way, which could potentially provide us with more resources and funds for continuously developing our product and exploring further in this area.

Finally, we believe that our interaction with Imperial students, potential students and faculty during "demon-

stration day" was able to educate about and encourage participation in the engineering profession and contribute to the development of future skilled workforce. Our demonstration also enhances the overall image of our profession and Imperial College.

Chapter 9

Ethical Considerations and Consequences

With any engineering project, one must consider ethical implications, especially when one is associated with large organisations such as Intel and Imperial College London. Engineering Ethics concerns with the micro-ethics of moral conduct in daily affairs of engineering and macro-ethics of collective interests of the engineering profession in society [53]. Micro-ethical considerations surrounding this project are relatively few due to the small team size, none the less, it was ensured that the team, regardless of their role was encouraged to contribute equally and be heard equally by the rest of the team by implementing various modes of communication and having periods where all team members would review each others work.

The macro-ethical considerations of this project are more significant. Due to Intel's large consumer base, and their as well as Imperial College's international recognition make this project potentially impactful to many people. To that end, we make the following considerations in accordance with [53].

- Pay attention to the sustainability triple bottom line: economic, social and environmental
- Involve with community to enhance social welfare
- Engage with and respect stakeholders [14]

Aside from environmental and economic considerations, which are described in details in the sustainability report section, we must consider legal implications surrounding intellectual property. While we do use several tools in the development of our project they are all available under an open source license (made available in the appendix). By doing this we minimise the potential for legal action against ourselves, our stakeholders and our customers. Furthermore, by referencing all tools used (chapter 10) we ensure that all contributors receive appropriate credit and that intellectual integrity is upheld.

Chapter 10

Conclusion and Future Work

With super-resolution being a highly active area of research, there are many different algorithms with many different advantages to consider. In our toolkit we implement a broad variety of algorithms which have a broad range of performance and efficiency. From our results, we place the ESRGAN as our most effective algorithm. This can be attributed to many things such as it being a learning based algorithm which means there is unlikely to be any common displeasing artifacts if the dataset it was trained on is large enough.

However, our goal is to appeal to all consumers in one toolkit. We therefore implement several non-learning approaches. Our range of discrete wavelength transform approaches provide very visually pleasing results despite some quantitative performance lag. Learning and wavelet approaches produce excellent results but due to their large computational requirements are likely more suited to industrial clients in markets where quality is paramount such as satellite imaging or security.

Beyond visual quality we have aimed for performance. To do this we have implemented the Local structure estimation method and NEDI with various implementations. These algorithms provide good results at a very high speed and are suited to more commercial video scaling applications such as integrated video scaling in TVs.

Finally, for the most basic tasks, such as enlarging the video screen on a device with low processing power, we provide the incredibly fast bicubic interpolation method which produces acceptable results with minimal computational requirements.

We conclude by summarising our implementation

- Easy to use, graphical user interface allowing users of all backgrounds to access our software.
- Several algorithms providing our users with a range of options to balance speed and performance.
- Each of our algorithms has been optimised as much as possible in terms of speed to minimise the carbon footprint and maximise throughput.

- We provide our users with the ability to super-resolve a live feed in MatLab.

Our toolkit focuses on ease of use and broad scope of applicability. Our GUI is based on ease of use and we ensure that minimal software is required to use our toolkit. The scope of our product ranges from fast scaling of old movies on live TV with simpler but extremely fast algorithms, to enhancing the feed from security cameras in real-time. The toolkit we provide allows the user to transition their video to the future, no matter what technology they have available to them, what resolution they're transitioning from or what resolution they're transitioning to.

To that end, in the future, we aim to implement our algorithms on FPGAs to make video super-resolution as easy as connecting an HDMI cable and pressing start on the PixelPerfect GUI. Furthermore, as AI and hardware acceleration for it evolve, in the near future, we will have a working implementation of a multi-frame super-resolution algorithm which, in terms of the HR image quality will be unprecedented. Meanwhile, we also aim to research and implement more super-resolution algorithms exemplified by bandelet-based algorithms and neural networks with different architectures.

Chapter 11

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] Eirikur Agustsson and Radu Timofte. Ntire 2017 challenge on single image super-resolution: Dataset and study. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017.

[3] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[4] Joan Bruna, Pablo Sprechmann, and Yann LeCun. Super-resolution with deep convolutional sufficient statistics. *arXiv preprint arXiv:1511.05666*, 2015.

[5] John Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698, 1986.

[6] S. Carrato, G. Ramponi, and S. Marsi. A simple edge-sensitive image interpolation filter. In *Proceedings of 3rd IEEE International Conference on Image Processing*, volume 3, pages 711–714 vol.3, 1996.

[7] Hong Chang, Dit-Yan Yeung, and Yimin Xiong. Super-resolution through neighbor embedding. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, pages I–I. IEEE, 2004.

[8] Jingwen Chen, Jiawei Chen, Hongyang Chao, and Ming Yang. Image blind denoising with generative adversarial network based noise modeling. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3155–3164, 2018.

[9] Kai Chen, Chao Liu, and Yongsheng Ou. Channel attention based network for lidar super-resolution. In *2021 China Automation Congress (CAC)*, pages 5458–5463, 2021.

[10] Wanli Chen and Hongjian Shi. An edge based adaptive interpolation algorithm for image scaling. 2018.

[11] Wanli Chen, Ya Jun Yu, and Hongjian Shi. An improvement of edge-adaptive image scaling algorithm based on sobel operator. In *2017 4th International Conference on Information Science and Control Engineering (ICISCE)*, pages 183–186, 2017.

[12] Yunjey Choi, Minje Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, and Jaegul Choo. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8789–8797, 2018.

[13] Alex Clark. Pillow (pil fork) documentation, 2015.

[14] Stewart Clegg and Carl Rhodes. *Management ethics: contemporary contexts*. Routledge, 2006.

[15] Hasan Demirel and Gholamreza Anbarjafari. Satellite image resolution enhancement using complex wavelet transform. *IEEE Geoscience and Remote Sensing Letters*, 7(1):123–126, 2010.

[16] Hasan Demirel and Gholamreza Anbarjafari. Image resolution enhancement by using discrete and stationary wavelet decomposition. *IEEE Transactions on Image Processing*, 20(5):1458–1460, 2011.

[17] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(2):295–307, 2016.

[18] Chao Dong, Chen Change Loy, and Xiaoou Tang. Accelerating the super-resolution convolutional neural network. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part II 14*, pages 391–407. Springer, 2016.

[19] Alexey Dosovitskiy and Thomas Brox. Generating images with perceptual similarity metrics based on deep networks. *Advances in neural information processing systems*, 29, 2016.

[20] Python Software Foundation. Python documentation. https://docs.python.org/3/.

[21] RW Gerchberg. Super-resolution through error energy reduction. *Optica Acta: International Journal of Optics*, 21(9):709–720, 1974.

[22] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[23] Ernst Adolph Guillemin. *The mathematics of circuit analysis*. John Wiley New York, 1949.

[24] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[25] J. L. Harris. Diffraction and resolving power∗. *J. Opt. Soc. Am.*, 54(7):931–936, Jul 1964.

[26] Forrest Hoffman. An introduction to fourier theory. *Extraído el*, 2, 1997.

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

[28] Chia-Chun Hsu, Jian-Jiun Ding, and Yih-Cherng Lee. Efficient edge-oriented based image interpolation algorithm for non-integer scaling factor. In *2017 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pages 1156–1159, 2017.

[29] Leying Hu. New edge-directed interpolation. https://github.com/Kirstihly/Edge-Directed_Interpolation/, 2020. Online; accessed 05 06 2023.

[30] The MathWorks Inc. Matlab engine api for python (r2023a), 2023.

[31] The MathWorks Inc. Matlab (r2023a), 2023.

[32] M. Irani and S. Peleg. Super resolution from image sequences. In *[1990] Proceedings. 10th International Conference on Pattern Recognition*, volume ii, pages 115–120 vol.2, 1990.

[33] Michal Irani and Shmuel Peleg. Motion analysis for image enhancement: Resolution, occlusion, and transparency. *Journal of visual communication and image representation*, 4(4):324–335, 1993.

[34] Sara Izadpanahi and Hasan Demirel. Multi-frame super resolution using edge directed interpolation and complex wavelet transform. In *IET Conference on Image Processing (IPR 2012)*, pages 1–5, 2012.

[35] Pilla Jagadeesh and Jayanthi Pragatheeswaran. Image resolution enhancement based on edge directed interpolation using dual tree — complex wavelet transform. In *2011 International Conference on Recent Trends in Information Technology (ICRTIT)*, pages 759–763, 2011.

[36] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 694–711, Cham, 2016. Springer International Publishing.

[37] Robert Keys. Cubic convolution interpolation for digital image processing. *IEEE transactions on acoustics, speech, and signal processing*, 29(6):1153–1160, 1981.

[38] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Accurate image super-resolution using very deep convolutional networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1646–1654, 2016.

[39] Jun-Yong Kim, Rae-Hong Park, and Seungjoon Yang. Super-resolution using pocs-based reconstruction with artifact reduction constraints. In *Visual Communications and Image Processing 2005*, volume 5960, pages 1810–1818. SPIE, 2005.

[40] Yongwoo Kim, Jae-Seok Choi, and Munchurl Kim. A real-time convolutional neural network for super-resolution on fpga with applications to 4k uhd 60 fps video services. *IEEE Transactions on Circuits and Systems for Video Technology*, 29(8):2521–2534, 2019.

[41] K Kishore Kumar, Thadigotla Venkata Subbareddy, K Sugapriya, and V Elamaran. Resolution enhancement using dwt and swt by fusion techniques with watermarking. In *2014 IEEE International Conference on Computational Intelligence and Computing Research*, pages 1–5, 2014.

[42] 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.

[43] Chang-Ming Lee, Chien-Jung Lee, Chia-Yung Hsieh, and Wen-Nung Lie. Super-resolution reconstruction of video sequences based on wavelet-domain spatial and temporal processing. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pages 194–197, 2012.

[44] Gregory Lee, Ralf Gommers, Kai Wohlfahrt, Filip Wasilewski, Aaron O'Leary, Holger Nahrstaedt, Alexandre Sauvé, Ankit Agrawal, Daniel M. Pelt, Helder Oliveira, Thomas Arildsen, Christian Clauss, Frank Yu, Matthew Brett, Michel Pelletier, SylvainLan, Daniele Tricoli, Saket Choudhary, asnt, Arfon Smith, 0-tree, Balint Reczey, Corey Goldberg, Daniel Goertzen, Dawid Laszuk, ElConno, Jacopo Antonello, Jakub Mandula, jakirkham, and Jonathan Dan. Pywavelets/pywt: v1.4.1, September 2022.

[45] Gregory R. Lee, Ralf Gommers, Filip Waselewski, Kai Wohlfahrt, and Aaron O8217;Leary. Pywavelets: A python package for wavelet analysis. *Journal of Open Source Software*, 4(36):1237, 2019.

[46] Xin Li and Michael T Orchard. Edge-directed prediction for lossless compression of natural images. *IEEE Transactions on image processing*, 10(6):813–817, 2001.

[47] Xin Li and M.T. Orchard. New edge-directed interpolation. *IEEE Transactions on Image Processing*, 10(10):1521–1527, 2001.

[48] Zhi-Song Liu, Wan-Chi Siu, and Jun-Jie Huang. Image super-resolution via hybrid nedi and wavelet-based scheme. In *2015 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*, pages 1131–1136, 2015.

[49] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60:91–110, 2004.

[50] Bruce D Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *IJCAI'81: 7th international joint conference on Artificial intelligence*, volume 2, pages 674–679, 1981.

[51] Michael Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. *arXiv preprint arXiv:1511.05440*, 2015.

[52] Yasutaka Matsuo and Shinichi Sakaida. Super-resolution for 2k/8k television using wavelet-based image registration. In *2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 378–382, 2017.

[53] Ananda D Moonasingha. Ethics, and social sustainability of engineering projects.

[54] Travis Oliphant. Numpy. https://numpy.org/.

[55] Jagyanseni Panda and Sukadev Meher. A new residual image sharpening scheme for image up-sampling. In *2022 8th International Conference on Signal Processing and Communication (ICSC)*, pages 244–249, 2022.

[56] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[57] GNU project. Gcc. https://gcc.gnu.org/.

[58] PySimpleGUI. Pysimplegui. https://github.com/PySimpleGUI/PySimpleGUI, 2018.

[59] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[60] 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. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1874–1883, 2016.

[61] Heng Su, Ying Wu, and Jie Zhou. Super-resolution without dense flow. *IEEE Transactions on Image Processing*, 21(4):1782–1795, 2012.

[62] Hang Sun, Shengfu Dong, Xiaodong Xie, Meng Li, Xiaofeng Huang, and Wen Gao. A novel hardware-based uhd video up-scaler based on local structure estimation. Springer.

[63] Shen-Chuan Tai, Jiun-Jie Huang, and Peng-Yu Chen. A super-resolution algorithm using linear regression based on image self-similarity. In *2016 International Symposium on Computer, Consumer and Control (IS3C)*, pages 275–278. IEEE, 2016.

[64] K Chaitanya Pavan Tanay, Srikanth Khanna, V Chandrasekaran, and P K Baruah. Fast video super resolution using deep convolutional networks. pages 1–6, 2017.

[65] Roger Y Tsai and Thomas S Huang. Multiframe image restoration and registration. *Multiframe image restoration and registration*, 1:317–339, 1984.

[66] Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu, and the scikit-image contributors. scikit-image: image processing in python. *PeerJ*, 2:e453, jun 2014.

[67] Haiyong Wang and Kai Jiang. Research on image super-resolution reconstruction based on transformer. In *2021 IEEE International Conference on Artificial Intelligence and Industrial Design (AIID)*, pages 226–230, 2021.

[68] Xun Wang. Interpolation and sharpening for image upsampling. In *2022 2nd International Conference on Computer Graphics, Image and Virtualization (ICCGIV)*, pages 73–77, 2022.

[69] Rich Wareham, Fergal Cotter, scf32, Timothy Roberts, Henry Gomersall, and Ghislain Antony Vaillant. rjw57/dtcwt: Version 0.12.0, September 2017.

[70] Wikipedia contributors. Comparison gallery of image scaling algorithms — Wikipedia, the free encyclopedia, 2023. [Online; accessed 28-June-2023].

[71] Wikipedia contributors. Ycbcr — Wikipedia, the free encyclopedia, 2023. [Online; accessed 23-May-2023].

[72] Qingsong Yang, Pingkun Yan, Yanbo Zhang, Hengyong Yu, Yongyi Shi, Xuanqin Mou, Mannudeep K. Kalra, Yi Zhang, Ling Sun, and Ge Wang. Low-dose ct image denoising using a generative adversarial network with wasserstein distance and perceptual loss. *IEEE Transactions on Medical Imaging*, 37(6):1348–1357, 2018.

[73] Chongjun Ye, Lingyu Yan, Yucheng Zhang, Jun Zhan, Jie Yang, and Junfang Wang. A super-resolution method of remote sensing image using transformers. In *2021 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, volume 2, pages 905–910, 2021.

[74] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 5908–5916, 2017.

[75] WenYi Zhao and Harpreet S Sawhney. Is super-resolution with optical flow feasible? In *Computer Vision ECCV 2002: 7th European Conference on Computer Vision Copenhagen, Denmark, May 28–31, 2002 Proceedings, Part I 7*, pages 599–613. Springer, 2002.

[76] H. Zheng, A. Bouzerdoum, and S. L. Phung. Wavelet based nonlocal-means super-resolution for video sequences. In *2010 IEEE International Conference on Image Processing*, pages 2817–2820, 2010.

[77] Minyan Zheng, Jianping Luo, and Wenming Cao. Video super-resolution based on spatial-temporal transformer. In *2021 IEEE 7th International Conference on Cloud Computing and Intelligent Systems (CCIS)*, pages 403–407, 2021.

Appendix A

Code and Licenses

A.1 Local Structure Based Methods: Naive Python implementation with C implementations available on GitHub.

```
def imgPad(img):

    lenY = len(img)
    lenX = len(img[0])

    padded = np.empty(shape = (lenY+10,lenX+10,3), dtype = int)
    padded[5:lenY+5,5:lenX+5,:] = img

    lenPY = lenY+10
    lenPX = lenX+10

    for i in range(5,0,-1):
      for x in range(i,lenPX-i):
        padded[i-1,x,:] = padded[i,x,:]
      for x in range(i,lenPX-i):
        padded[lenPY-i,x,:] = padded[lenPY-i-1,x,:]
      for y in range(i,lenPY-i):
        padded[y,i-1,:] = padded[y,i,:]
      for y in range(i,lenPY-i):
        padded[y,lenPX-i,:] = padded[y,lenPX-i-1,:]
```



```
      padded[i-1,i-1,:] = padded[i,i,:]
      padded[i-1,lenPX-i,:] = padded[i,lenPX-i-1,:]
      padded[lenPY-i,i-1,:] = padded[lenPY-i-1,i,:]
      padded[lenPY-i,lenPX-i,:] = padded[lenPY-i-1,lenPX-i-1,:]


    return padded
```

```
def imgExpand(img):

  lenY = len(img)*2
  lenX = len(img[0])*2

  scaled = np.zeros(shape = (lenY,lenX,3), dtype = int)

  print(img.shape)

  for c in range(0,3):
    for y in range(0,lenY):
      for x in range(0,lenX):
        if((x%2 == 0) and (y%2 == 0)):

          scaled[y][x][c] = int(img[int(y/2)][int(x/2)][c])

  return scaled


def stage1weights(img, x, y, esf):

    coefs = [-0.125, 0.625, 0.625, -0.125]


    interpX45 =  coefs[0]*img[y-3][x-3] + coefs[1]*img[y-1][x-1] + coefs[2]*img[y+1][x+1] + coefs[3]*img[y+3][x+3]
    interpX135 = coefs[0]*img[y+3][x-3] + coefs[1]*img[y+1][x-1] + coefs[2]*img[y-1][x+1] + coefs[3]*img[y-3][x+3]

    interpA45 =  (coefs[0]*img[y-3][x-3]) + (coefs[1]*img[y-1][x-1]) + (coefs[2]*img[y+3][x+3]) + (coefs[3]*img[y+5][x+5])
    interpA135 = (coefs[0]*img[y-3][x+5]) + (coefs[1]*img[y-1][x+3]) + (coefs[2]*img[y+3][x-1]) + (coefs[3]*img[y+5][x-3])

    interpB45 =  (coefs[0]*img[y+3][x+5]) + (coefs[1]*img[y+1][x+3]) + (coefs[2]*img[y-3][x-1]) + (coefs[3]*img[y-5][x-3])
    interpB135 = (coefs[0]*img[y-5][x+5]) + (coefs[1]*img[y-3][x+3]) + (coefs[2]*img[y+1][x-1]) + (coefs[3]*img[y+3][x-3])

    interpC45 =  (coefs[0]*img[y+3][x+3]) + (coefs[1]*img[y+1][x+1]) + (coefs[2]*img[y-3][x-3]) + (coefs[3]*img[y-5][x-5])
    interpC135 = (coefs[0]*img[y-5][x+3]) +  (coefs[1]*img[y-3][x+1]) + (coefs[2]*img[y+1][x-3]) + (coefs[3]*img[y+3][x-5])

    interpD45 =  (coefs[0]*img[y+5][x+3]) + (coefs[1]*img[y+3][x+1]) + (coefs[2]*img[y-1][x-3]) + (coefs[3]*img[y-3][x-5])
    interpD135 = (coefs[0]*img[y+5][x-5]) + (coefs[1]*img[y+3][x-3]) + (coefs[2]*img[y-1][x+1]) + (coefs[3]*img[y-3][x+3])

    e45 = abs(interpA45-img[y+1][x+1]) + abs(interpB45 - img[y-1][x+1]) + abs(interpC45 - img[y-1][x-1]) + abs(interpD45 - img[y+1][x-1])
    e135 = abs(interpA135-img[y+1][x+1]) + abs(interpB135 - img[y-1][x+1]) + abs(interpC135 - img[y-1][x-1]) + abs(interpD135 - img[y+1][x-1])

    esf45 = pow(e45, esf)
    esf135 = pow(e135, esf)

    w45 = (esf135 + 0.001) / (esf135+esf45+0.002)
    w135 = 1 - w45

    #print(interpX45, interpX135)

    return w45*interpX45 + w135*interpX135
```

```
def stage2weights(img, x, y, esf):

  coefs = [-0.125, 0.625, 0.625, -0.125]

  interpX0 =  (coefs[0]*img[y-0][x+3]) + (coefs[1]*img[y-0][x+1]) + (coefs[2]*img[y-0][x-1]) + (coefs[3]*img[y-0][x-3])
  interpX90 = (coefs[0]*img[y+3][x-0]) + (coefs[1]*img[y+1][x-0]) + (coefs[2]*img[y-1][x-0]) + (coefs[3]*img[y-3][x-0])

  interpA0 = (coefs[0]*img[y-0][x-3]) + (coefs[1]*img[y-0][x-1]) + (coefs[2]*img[y-0][x+3]) + (coefs[3]*img[y-0][x+5])
  interpA90 = (coefs[0]*img[y+4][x+1]) + (coefs[1]*img[y+2][x+1]) + (coefs[2]*img[y-2][x+1]) + (coefs[3]*img[y-4][x+1])

  interpB0 = (coefs[0]*img[y-1][x-4]) + (coefs[1]*img[y-1][x-2]) + (coefs[2]*img[y-1][x+2]) + (coefs[3]*img[y-1][x+4])
  interpB90 = (coefs[0]*img[y+3][x-0]) + (coefs[1]*img[y+1][x-0]) + (coefs[2]*img[y-3][x-0]) + (coefs[3]*img[y-5][x-0])

  interpC0 = (coefs[0]*img[y-0][x-5]) + (coefs[1]*img[y-0][x-3]) + (coefs[2]*img[y-0][x+1]) + (coefs[3]*img[y-0][x+3])
  interpC90 = (coefs[0]*img[y+4][x-1]) + (coefs[1]*img[y+2][x-1]) + (coefs[2]*img[y-2][x-1]) + (coefs[3]*img[y-4][x-1])

  interpD0 = (coefs[0]*img[y+1][x-4]) + (coefs[1]*img[y+1][x-2]) + (coefs[2]*img[y+1][x+2]) + (coefs[3]*img[y+1][x+4])
  interpD90 = (coefs[0]*img[y+5][x-0]) + (coefs[1]*img[y+3][x-0]) + (coefs[2]*img[y-1][x-0]) + (coefs[3]*img[y-3][x-0])

  e0 = abs(interpA0-img[y+1][x+1]) + abs(interpB0 - img[y-1][x+1]) + abs(interpC0 - img[y-1][x-1]) + abs(interpD0 - img[y+1][x-1])
  e90 = abs(interpA90-img[y+1][x+1]) + abs(interpB90 - img[y-1][x+1]) + abs(interpC90 - img[y-1][x-1]) + abs(interpD90 - img[y+1][x-1])

  esf0 = pow(e0, esf)
  esf90 = pow(e90, esf)

  w0 = (esf0 + 0.001) / ((esf90+esf0) + 0.002)
  w90 = 1 - w0

  return w0*interpX0 + w90*interpX90


def stage1(img, esf):
  for y in range(5,len(img)-5,2):
    for x in range(5,len(img[0])-5,2):
      img[y][x] = stage1weights(img, x, y, esf)

  return img


def stage2(img, esf):
  for y in range(10, len(img)-10,1):
    if(y%2 == 0):
      p = 11
    else:
      p = 10
    for x in range(p, len(img[0])-10, 2):
      img[y][x] = stage2weights(img, x, y, esf)

  return img
```

```
def SRwLocalStructEst(img, esf = 2):

  padded = imgPad(img)
  scaled = imgExpand(padded)

  stage1(scaled[:,:,0], esf)
  stage1(scaled[:,:,1], esf)
  stage1(scaled[:,:,2], esf)

  stage2(scaled[:,:,0], esf)
  stage2(scaled[:,:,1], esf)
  stage2(scaled[:,:,2], esf)

  return scaled[10:len(scaled)-10,10:len(scaled[0])-10,:]


def LocalStruct(img, scale):
    if scale % 2 != 0:
        print("Scale not supported by this mode, change to bicubic mode")
        return cv2.resize(img, None, fx=scale, fy=scale, interpolation=cv2.INTER_CUBIC)
    else:
        print("Local Structure")
        final_image = img
        for i in range (scale//2):
            final_image = SRwLocalStructEst(final_image, esf=3)

        return final_image
```

A.2 NEDI and Improved NEDI Code

```
function imgo = fcn(frame)

m = 5; % window size
s = 2; % scale size
```

```
dims = size(frame);
rows = dims(1);
cols = dims(2);
targetSize = [2*rows, 2*cols];


% Canny edge detection
img=im2double(frame);

I=rgb2gray(img);

sharp = edge(I,'canny', 0.3);


% insert low-resolution pixels
imgo = imresize(img, targetSize, 'bicubic');


y = double(zeros(m^2, 1)); % pixels in the window
C = double(zeros(m^2, 4)); % interpolation neighbours of each pixel in the window


%% step 1  reconstruct the points with the form of (2*i+1,2*j+1)
for k=1:3  % 3 colors
    for i=4:rows-4
        for j=4:cols-4
                if(sharp(i,j)==1)
                temp=1;
                for ii=(i+1-ceil(m/2)):(i+floor(m/2))
                    for jj=j+1-ceil(m/2):j+floor(m/2)
                        y(temp)=imgo(2*ii,2*jj,k);
                        C(temp,1)=imgo(2*ii-2,2*jj-2,k);
                        C(temp,2)=imgo(2*ii+2,2*jj-2,k);
```

```
                    C(temp,3)=imgo(2*ii+2,2*jj+2,k);
                    C(temp,4)=imgo(2*ii-2,2*jj+2,k);
                    temp=temp+1;
                end
            end
            alpha = pinv(C' * C) * C' * y;
            imgo(2*i+1,2*j+1,k)=(alpha(1)*imgo(2*i,2*j,k)+alpha(2)*imgo(2*i+2,2*j,k)...
                +alpha(3)*imgo(2*i+2,2*j+2,k)+alpha(4)*imgo(2*i,2*j+2,k));

        end
    end
end
end


%% step 2 reconstructed the points with the forms of (2*i+1,2*j)
for k=1:3  % 3 colors
    for i=4:rows-4
        for j=4:cols-4
            if(sharp(i,j)==1)
            temp=1;
            for ii=(i+1-ceil(m/2)):(i+floor(m/2))
                for jj=j+1-ceil(m/2):j+floor(m/2)
                    y(temp)=imgo(2*ii+1,2*jj-1,k);
                    C(temp,1)=imgo(2*ii-1,2*jj-1,k);
                    C(temp,2)=imgo(2*ii+1,2*jj-3,k);
                    C(temp,3)=imgo(2*ii+3,2*jj-1,k);
                    C(temp,4)=imgo(2*ii+1,2*jj+1,k);
                    temp=temp+1;
                end
            end
            alpha = pinv(C' * C) * C' * y;
            imgo(2*i+1,2*j,k)=(alpha(1)*imgo(2*i,2*j,k)+alpha(2)*imgo(2*i+1,2*j-1,k)...
                +alpha(3)*imgo(2*i+2,2*j,k)+alpha(4)*imgo(2*i+1,2*j+1,k));
```

```
            end
        end
    end
end

%% step 3 reconstructed the points with the forms of (2*i,2*j+1)

for k=1:3  % 3 colors
    for i=4:rows-4
        for j=4:cols-4
                if(sharp(i,j)==1)
                temp=1;
                for ii=(i+1-ceil(m/2)):(i+floor(m/2))
                    for jj=j+1-ceil(m/2):j+floor(m/2)
                        y(temp)=imgo(2*ii+1,2*jj-1,k);
                        C(temp,1)=imgo(2*ii-1,2*jj-1,k);
                        C(temp,2)=imgo(2*ii+1,2*jj-3,k);
                        C(temp,3)=imgo(2*ii+3,2*jj-1,k);
                        C(temp,4)=imgo(2*ii+1,2*jj+1,k);
                        temp=temp+1;
                    end
                end
                alpha = pinv(C' * C) * C' * y;
                imgo(2*i,2*j+1,k)=(alpha(1)*imgo(2*i-1,2*j+1,k)+alpha(2)*imgo(2*i,2*j,k)...
                    +alpha(3)*imgo(2*i+1,2*j+1,k)+alpha(4)*imgo(2*i,2*j+2,k));
            end
        end
    end
end
```

A.3 Full license for GNU tools

GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require,

such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the

terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non–Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to

copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must

suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall

be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your

license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer–to–peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and

propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant

patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license

you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any

author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

A.4 Full license for numpy library

A.5 Full License for OpenCV Library

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or
Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices
stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works
that You distribute, all copyright, patent, trademark, and
attribution notices from the Source form of the Work,
excluding those notices that do not pertain to any part of
the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its
distribution, then any Derivative Works that You distribute must
include a readable copy of the attribution notices contained
within such NOTICE file, excluding those notices that do not
pertain to any part of the Derivative Works, in at least one
of the following places: within a NOTICE text file distributed
as part of the Derivative Works; within the Source form or
documentation, if provided along with the Derivative Works; or,
within a display generated by the Derivative Works, if and
wherever such third-party notices normally appear. The contents
of the NOTICE file are for informational purposes only and
do not modify the License. You may add Your own attribution
notices within Derivative Works that You distribute, alongside
or as an addendum to the NOTICE text from the Work, provided
that such additional attribution notices cannot be construed
as modifying the License.

You may add Your own copyright statement to Your modifications and
may provide additional or different license terms and conditions
for use, reproduction, or distribution of Your modifications, or
for any such Derivative Works as a whole, provided Your use,
reproduction, and distribution of the Work otherwise complies with

the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all

other commercial damages or losses), even if such Contributor
has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
the Work or Derivative Works thereof, You may choose to offer,
and charge a fee for, acceptance of support, warranty, indemnity,
or other liability obligations and/or rights consistent with this
License. However, in accepting such obligations, You may act only
on Your own behalf and on Your sole responsibility, not on behalf
of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability
incurred by, or claims asserted against, such Contributor by reason
of your accepting any such warranty or additional liability.

A.6 Full License for dtcwt Library

This licence applies to any parts of this library which are novel in comparison to the original DTCWT MATLAB toolbox written by Nick Kingsbury and Cian Shaffrey. See the ORIGINAL_README.txt file for details on any further restrictions of use.

Copyright (c) 2013, Rich Wareham <rjw57@cantab.net>
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND

ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

A.7 Full License for PySimpleGUI

GNU LESSER GENERAL PUBLIC LICENSE
Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <https://fsf.org/>
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

0. Additional Definitions.

As used herein, "this License" refers to version 3 of the GNU Lesser General Public License, and the "GNU GPL" refers to version 3 of the GNU General Public License.

"The Library" refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An "Application" is any work that makes use of an interface provided

by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A "Combined Work" is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the "Linked Version".

The "Minimal Corresponding Source" for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The "Corresponding Application Code" for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the

function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or

b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.

b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.

b) Accompany the Combined Work with a copy of the GNU GPL and this license document.

c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.

d) Do one of the following:

0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.

1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.

e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation

Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.

b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser

General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy's public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.