TU WIEN Informatics

Pixel Art Restoration

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Florian Wagner
Matrikelnummer 11907095

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Mitwirkung: Dipl.-Ing. Lukas Lipp

Wien, 20. März 2024

Florian Wagner   Michael Wimmer

Pixel Art Restoration

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Florian Wagner
Registration Number 11907095

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Assistance: Dipl.-Ing. Lukas Lipp

Vienna, 20th March, 2024

Florian Wagner	Michael Wimmer

Erklärung zur Verfassung der Arbeit

Florian Wagner

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 20. März 2024

Florian Wagner

Danksagung

Zunächst möchte ich meiner Familie danken, die mir während meines Studiums stets tatkräftig zur Seite stand und mich in allen Belangen unterstützte. Ebenso möchte ich meinen Mitstudierenden meinen Dank aussprechen, die mir sowohl während meines Studiums als auch bei dieser Arbeit stets zur Seite standen.

Ein besonderer Dank gebührt meinem Betreuer Lukas Lipp, der trotz gelegentlicher Verzögerungen meinerseits stets zur Verfügung stand und mich in sämtlichen Aspekten dieser Arbeit ermutigte und unterstützte.

Acknowledgements

First of all, I would like to thank my family, who always stood by my side during my studies and supported me in all matters. I would also like to thank my fellow students, who have always supported me both during my studies and in this thesis.

Special thanks go to my supervisor Lukas Lipp, who, despite occasional delays on my part, was always available and encouraged and supported me in all aspects of this thesis.

Kurzfassung

Spiele, die in der 8- und 16-Bit-Ära entwickelt wurden, verwendeten niedrig aufgelöste Bilder, so genannte Sprites, zur Darstellung der Spielwelten und ihrer Objekte. Dieser Kunststil wird oft als Pixel-Art bezeichnet und hat sich zu einem eigenen Subgenre von Spielen entwickelt, für das auch heute noch neue Spiele veröffentlicht werden. Während moderne Pixel-Art-Spiele die visuelle Wiedergabetreue moderner Flüssigkristallbildschirme (LCD) in ihr Design einbeziehen, sehen Spiele aus der Kathodenstrahlröhren-Ära (CRT) oft schlechter aus, wenn sie auf einem LCD-Monitor angezeigt werden. Das ist verständlich, denn die damaligen Spiele wurden entwickelt, um auf Röhrenfernsehern und -monitoren gut auszusehen, die im Vergleich zu modernen Bildschirmen ganz anders funktionieren.

Seit dem Aufkommen von Emulatoren für Retro-Spielkonsolen wurde viel Mühe darauf verwendet, die Effekte von Röhrenmonitoren bei der Wiedergabe dieser alten Spieldateien auf LCD-Monitoren zu reproduzieren. Dies erfordert oft die Verwendung von Upscaling-Algorithmen, um das Aussehen der niedrig aufgelösten Spiele auf Monitoren mit höherer Auflösung zu verbessern. Upscaling im Allgemeinen, aber auch auf Pixel-Art zugeschnitten, ist immer noch ein ungelöstes Problem und der Schwerpunkt vieler neuerer Veröffentlichungen.

In dieser Arbeit werden wir untersuchen, welche Auswirkungen CRT-Monitore auf Pixel-Art haben und wie man diese Informationen nutzen kann, um die Hochskalierung von Pixel-Art aus der 8- und 16-Bit-Ära anzuleiten, um die visuelle Wiedergabetreue bei der Darstellung auf modernen LCD-Monitoren zu verbessern.

Abstract

Games developed in the 8 and 16-bit era of computing used low-resolution images called sprites for displaying game worlds and their objects. This art style is often referred to as Pixel-Art and evolved into its own subgenre of games with new games still getting released to this day. While modern Pixel-Art games incorporate the visual fidelity of modern Liquid Crystal Display (LCD) monitors into their design, games from the Cathode Ray Tube (CRT) era often look worse when displayed on an LCD monitor. Understandable, since games at that time were designed to look good on CRT TVs and Monitors which function in very different ways compared to modern Displays.

Since the upcoming of retro game console emulators, a lot of effort was put into reproducing the effects of CRT monitors when playing back these old game files on LCD monitors. This often requires the use of upscaling algorithms to improve the look of the low-resolution game assets on higher-resolution monitors. Upscaling in general but also tailored towards Pixel-Art is still an unsolved problem and the focus of many recent publications.

In this work, we will explore what effects CRT monitors have on Pixel-Art and how to use this information to guide the upscaling of 8 and 16-bit era Pixel-Art to improve visual fidelity when displayed on modern LCD monitors.

Contents

CHAPTER 1

Introduction

In this chapter, we present the motivation for this work and discuss the underlying problem of displaying pixel-art as-is on modern displays.

1.1 Motivation

Since the early days of computer games, the visual appearance of the game's content has been of high importance. During these early times of game development, game developers did not have the computational resources of today available to build large-scale worlds with stunning visuals. Rather, they used low-resolution two-dimensional images, typically referred to as sprites, to build the game's world. These sprites were not only low in resolution, but also low in the usage of colors since computer screens during that time had a limited color palette to display, with only 8 bits in total for all three color channels, resulting in 256 colors total. During the 16-bit era, this limit was lifted to 65,536 different colors.

In essence, this meant that game developers had to carefully craft these sprites to get visually appealing results. This art style is usually referred to as pixel art and still receives a lot of attention, especially within the indie games industry. Games designed for Cathode Ray Tube (CRT) TVs often look worse on modern displays because they lose aspects of their original design. Understandably, games at that time were designed to look good on CRT TVs/monitors, which function in a very different way compared to modern displays. Modern pixel art games make use of the high visual fidelity of Liquid Crystal Display (LCD) monitors, making them look more detailed in comparison to their predecessors.

Figure 1.1 demonstrates the distinct viewing experiences of pixel art on a CRT and LCD display. The original image (a) looks dull when viewed on a modern display in comparison to how it looks on a CRT display. Much work and research has been dedicated

to developing filter solutions that replicate the experience of a CRT display, known as CRT filters. These filters simulate the analog processes an image, such as (a), would have undergone on those devices to look like image (b).
Image (c) is an artist's reinterpretation of the pixel art (a), preserving the original pixel art but reflecting the color and form of its CRT version. This interpretation illustrates how the pixel art in (a) would have appeared if it were originally crafted on a modern display. Our goal is to faithfully recreate this form, capturing its color and form as envisioned when it was crafted on a CRT display, and bring it to life on modern display technologies.

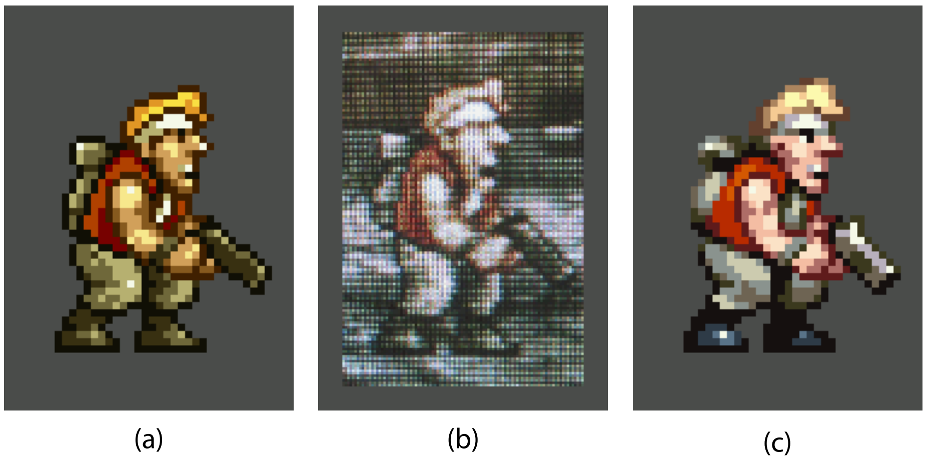


Figure 1.1: A visualization by X user Thomas Feichtmeir [Fei], describing how pixel art should look like, when designed for modern Liquid Crystal Display (LCD) displays.
(a) The raw pixel art.
(b) The original look when viewed on a CRT display.
(c) Handcrafted reinterpretation (by Thomas Feichtmeir [Fei]) of the raw pixel art (a) capturing its intrinsic crt look (b) for modern displays.

1.2 Problem Definition

One of the major reasons for the difference in color and form of pixel art on modern displays is the fundamental difference in how a CRT display operates compared to modern LCD displays. With the rise of retro game console emulators, considerable effort has been dedicated to replicating the effects of CRT monitors when playing back these old game files (ROMs) on LCD monitors. However, methods for generating higher detailed versions (Figure 1.1 c) of the original pixel art (Figure 1.1 a) while capturing its intrinsic CRT look (Figure 1.1 b) without involving manual work by professional artists remain completely unexplored.

In this work, we investigate the feasibility of utilising a neural network to generate a higher quality version, similar to image (c) in Figure 1.1, from its CRT counterpart (b).

CHAPTER 2

Related Work

In this chapter, we delve into greater detail about various types of pixel-art upscalers and downscalers. Additionally, we categorize the upscalers into non-AI and AI sub-categories, respectively. We employ these upscalers, in particular the SRRes network, in our work to re-create, or more accurately, reconstruct pixel art for higher-resolution displays. Finally, we will examine existing CRT filters developed for emulation use and leverage the insights gained from our research to guide the upscaling, using the aforementioned SRRes network, of CRT images into a higher definition version that faithfully represents the pixel art on a modern display

2.1 Pixel Art Upscaler

Pixel art upscalers are algorithms that “upscale” pixelated images or graphics into smoother or higher-quality representations. We categorize these upscalers into two groups: Non-AI, as discussed in sections 2.1.1 and 2.1.2, and AI-powered, which aim to enhance results using Artificial Intelligence techniques.

2.1.1 Non-AI

Various upscaling or upsampling techniques and algorithms exist for images and graphics. One particularly intriguing algorithm for our work is MMPX [MG21], a style-preserving pixel-art magnification filter.

Similar to other upscaling algorithms, the MMPX algorithm establishes a set of rules that determine the upsampling of the input image to a 2x output image. A 2x magnification signifies that the output image is twice the size of the input image.

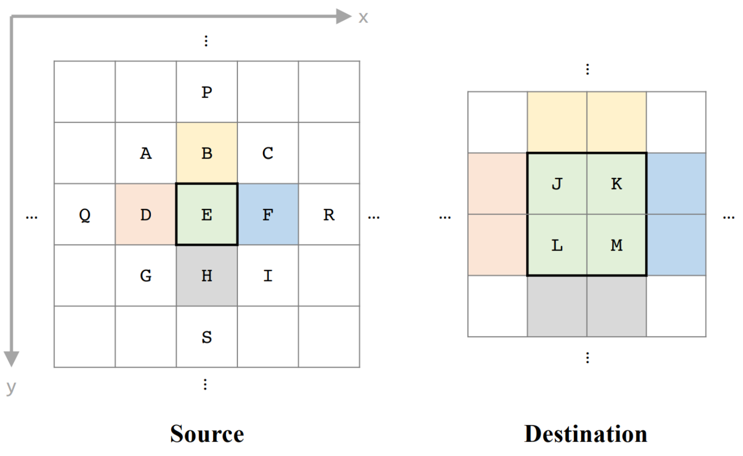


Figure 2.1: The source-to-destination mapping in Mazzoneli notation that MMPX uses is as follows: J, K, L, and M represent the destination pixels, while the 3x3 neighborhood pixels A-I and the diamond points P-S are utilized as input pixels [MG21].

As mentioned the MMPX algorithm works by defining a set of rules such as

- Fallback to Nearest Neighbor
  In case the central source pixel is different than the other pixels in the 3x3 center grid and the 3x3 grid pixels are all different.
- 1:1 Edges
  The 1:1 Edge rule applies for horizontal, vertical or diagonally-conntected runs of pixels, which feature the exact same value.
- Triangle Tips
  The triangle tips is needed because the 1:1 Edge rule flattens the luminance tips of bright triangles which are against a darker background. This rule will correct such errors and restore the corners.
- 2:1 Edges
  The 2:1 Edge rule extends the reach of the 1:1 Edge rule by an additional pixel which has previously been assigned by a different rule.

A different approach to upscaling images is to apply an image vectorization technique to convert them into vector graphics. One of these techniques, specifically tailored for pixel art by [KL11], extracts a smooth, resolution-independent vector representation from the image.

Most algorithms designed to automatically extract vector representations from images are intended for natural images. The fundamental and primary primitive used is a quadratic B-Spline curve. The original image is represented as a lattice graph, where each pixel represents a cell in this graph.

Considering the properties of such a graph, neighboring pixels share an edge, while diagonal neighbors only share a vertex. As a result, the first step in their approach involves reshaping the square pixel cells so that every pair of neighboring pixels along a thin feature corresponds to cells that share an edge [KL11]. To ensure correct identification of connections, several heuristics are used to resolve locally ambiguous diagonal configurations. Three major heuristics are employed to process distinct features of the pixel art image, specifically Curves, Sparse Pixels, and Islands, respectively.

With the graph reshaped, edges can be identified where adjacent pixels exhibit significantly different colors. These edges form the foundation of the visible contours in the vector representation. The remaining edges will lie within smoothly shaded regions. Quadratic B-spline curves are fitted to the identified contour edges, resulting in smooth contours for the vectorized output.

Finally, the curves are optimized to reduce staircasing effects, which can occur due to the low-resolution pixel grid used for quantization and placement of the spline control points.

2.1.2 AI

In recent years, there has been significant development regarding artificial intelligence in combination with image filtering and feature recognition. One such advancement is the ability to upscale images using neural networks, which are nearly indistinguishable from the original [LTH+16].

Super Resolution

Super Resolution (SR) is an umbrella term for a number of techniques that process Generative Adversarial Network (GAN) images into a High Resolution (HR) representation using signal processing [CHQ+22]. In essence, SR is the process of obtaining one or more HR images from one or more LR input images.[NM14].

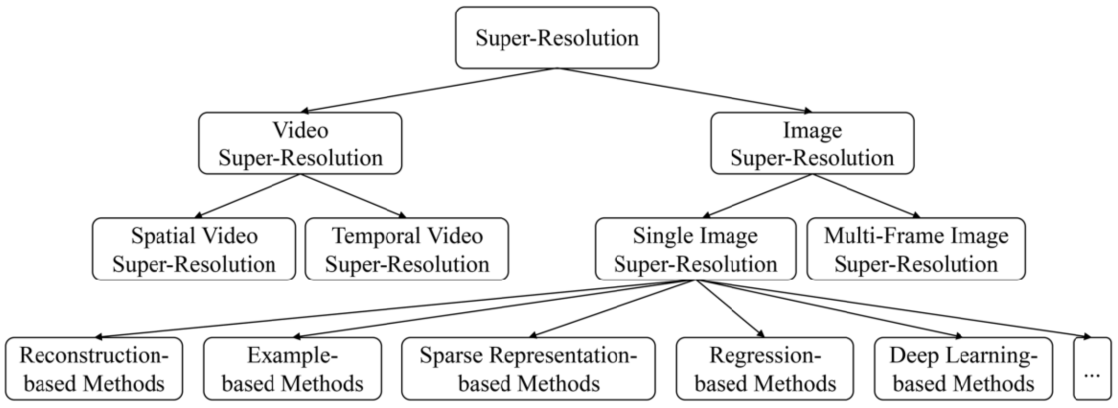

Super-Resolution
Video Super-Resolution
Image Super-Resolution
Spatial Video Super-Resolution
Temporal Video Super-Resolution
Single Image Super-Resolution
Multi-Frame Image Super-Resolution
Reconstruction-based Methods
Example-based Methods
Sparse Representation-based Methods
Regression-based Methods
Deep Learning-based Methods
...

Figure 2.2: An overview of existing super resolution techniques [CHQ+22].

Figure 2.2 gives an overview of the various techniques that exist today. SR is not only able to produce a higher resolution image; these techniques allow for resolutions beyond the limit of typical imaging solutions [CHQ+22]. Concisely, SR algorithms provide details finer than the sampling grid [PETM09]. Therefore, they are used for subsequent image analysis, segmentation, and other image manipulation or recognition tasks [CHQ+22].
Typical software filtering algorithms, such as the MMPX algorithm discussed in 2.1.1, suffer from oversimplifying the Single Image Super-Resolution problem. This oversimplification leads to results with overly smooth textures [LTH+16].
Besides Single Image Super Resolution (SISR), which only focuses on one singular image, Video Super Resolution (VSR) focuses on video sequences. The two main subsets of VSR are spatial VSR, which aims to improve the spatial resolution of a video, and temporal VSR, which, in turn, aims to improve the frame rate of the observed video [CHQ+22].
For our work, we mainly focus on Single Image Super-Resolution (SISR), which is an approach that recovers or estimates a high-resolution (HR) image from just a single low-resolution GAN input.

SRResNet

Image processing and computer vision use neural networks to leverage the difficulty of image processing and recognition tasks. However, typically, the better such networks become, the deeper they will become. In this context, "deepness" refers to the number of layers that make up the neural network.
"Deep learning methods aim at learning feature hierarchies with features from higher levels of the hierarchy formed by the composition of lower level features."[GB10]
The main problem with deep learning methods, and deep neural networks in general, is that the deeper the neural network gets, the more difficult it is to properly train. Once such deep networks start converging, it becomes apparent that with increasing depth, the network's accuracy gets saturated and degrades [HZRS16].

To this end, deep residual learning, was proposed in the form of "ResNet" by He et al. [HZRS16], a residual neural network that leverages the learning and degradation problem by introducing shortcut connections, also known as skip connections, into the network. Shortcut connections are connections that skip one or multiple layers in the network. In the case of "ResNet", these connections perform identity mappings, and their outputs are simply added to the outputs of the stacked layers [HZRS16]. One of the great advantages of identity shortcut connections is that they do not add any extra parameters, nor do they add computational complexity to the overall network [HZRS16].

When considering the underlying mapping $H(x)$ the stacked layer shall fit to another mapping $F(x) = H(x) - x$. The original mapping is then recast to $F(x) + x$. Where x is the identity mapping of the shortcut connection. This formulation can be realized by feedforward neural networks that employ said connections [HZRS16].

To that end, "SRResNet" proposed by Ledig et al. [LTH+16] is a super-resolution residual network, which employs shortcut connections, to achieve a deep neural network

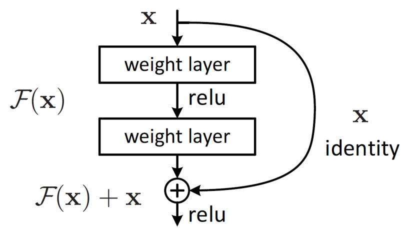


Figure 2.3: A building block for residual learning. Symbolizing how feed forward networks can be created with shortcut connections to realize the formulation $F(x) + x$.[HZRS16]

that allows for high upscaling factors up to 4x. The network consists of three different building blocks that make out the layers of the network. These building blocks are a convolutional block, which performs 2d convolution on the target. A residual block, which introduces skip connections and a sub-pixel convolutional block which is responsible, as the name suggests, for sub-pixel convolution to upscale the image by a given factor. The convolutional block is used in all other modules like the residual block as well as the sub-pixel convolutional block. The convolutional block has an optional activation layer as well as an optional batch normalization layer.

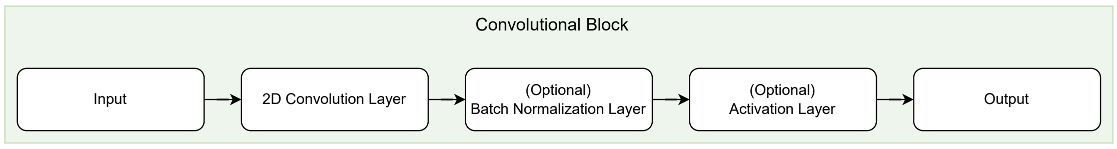

Convolutional Block
Input
2D Convolution Layer
(Optional) Batch Normalization Layer
(Optional) Activation Layer
Output

Figure 2.4: The convolutional block in our implementation. Both the Batch Normalization layer as well as the activation layer are optional. The convolutional block is used throughout our implementation.

The forward propagation for the convolutional block is then calculated by having the 2D convolution and the optional batch normalization and activation layer in sequence, as shown in Figure: 2.4. The residual block used by the network is composed of two convolutional blocks, both having a batch normalization layer, but only the first one has a “PReLU”, a Parametric Rectified Linear Unit, activation layer. The forward propagation for these blocks is then the sequential execution of the convolutional blocks. Since these are residual blocks, a skip connection in the form of the input is added to the output of the second convolution.

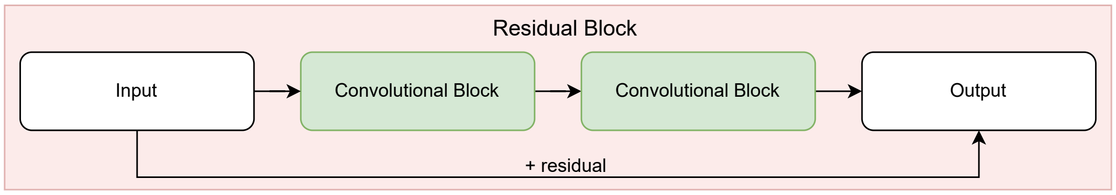


Figure 2.5: Diagram showing the residual block in our implementation. It consists of two convolutional blocks, both having input and output channels of size 64 and a kernel size of 3. Both convolutional blocks have a batch normalization layer and the first block also features a PReLU activation layer.

The sub-pixel convolutional block consists of a 2D convolution, which scales the output channels to the number of input channels times the scaling factor squared, a pixel shuffle, and a PReLU layer. The pixel shuffle is needed to form additional pixels which are used for the sub-pixel upscaling. It upscales each dimension by the scaling factor. It "rearranges elements in a tensor of shape (*,C×r**2 ,H,W) to a tensor of shape (*,C,H×r,W×r), where r is an upscale factor."[Fou] Where * is zero or more batch dimensions.

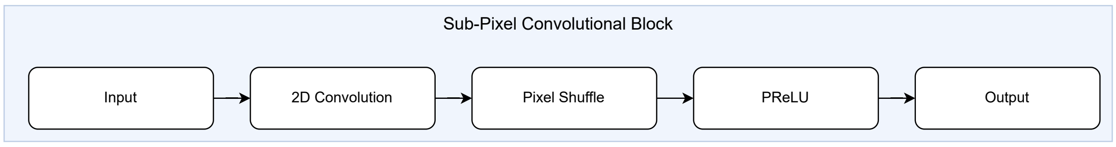


Figure 2.6: This diagram shows how the sub-pixel convolutional block is structured. It features 2D convolution input channels and $64 * scalingSactor^2$ output channels, a pixel shuffle to shuffle the additional channels to form additional pixels, and a PReLU layer.

Finally they all come together and are used to form SRResNet. The first convolutional block transforms the three input channels into 64 output channels with a kernel size of nine and a PReLU activation layer. This output is then stored as a residual, which will be added back later in the pipeline, and is fed to the residual blocks. There are a total of 16 residual blocks in sequence. It further gets propagated into another convolutional block with a batch normalization layer after which the residual is added back to the output and used as input to the sub-pixel convolutional block. The number of sub-pixel convolutional blocks depends on the scaling factor, which is chosen. In our case, the upscaling factor is two, resulting in one sub-pixel convolutional block. This output is then used by the final convolutional block, which transforms the 64 channels back into three. The final convolutional block has a Hyperbolic Tangent activation layer, applying the Hyperbolic Tangent (Tanh) element-wise to its input.

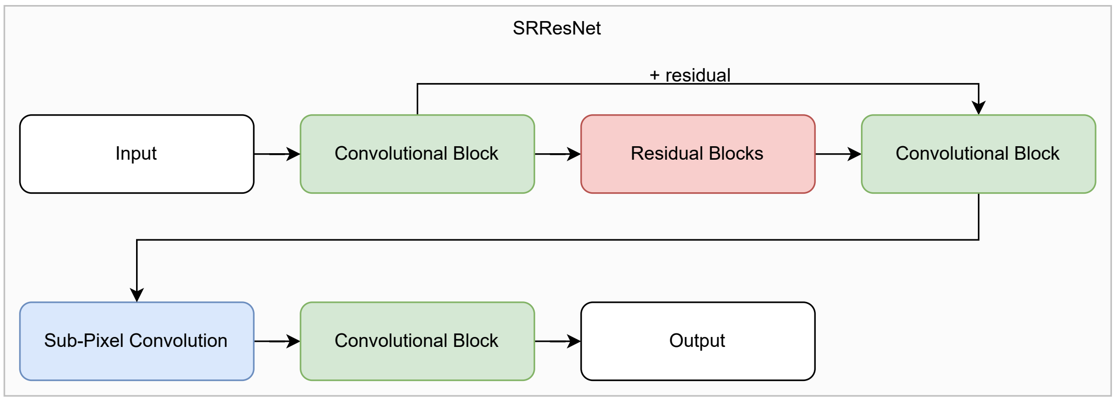


Figure 2.7: This diagram shows how the previously mentioned blocks are used together to form the SRResNet. The residual is formed as a skip connection that “skips” the residual blocks and the second convolutional block.

2.2 Pixelart Downscaler

Pixelart downscalers, as the name suggests, are designed to do the opposite of what we described in 2.1 for pixelart upscalers. Instead of upscaling the image - going from the pixelated graphic into a smooth or even vector representation - they are designed to downscale normal images into pixelated versions of the input image.

2.3 CRT Filter

In recent years, with the introduction of sophisticated emulators such as libretro [Tea24], which provide users with necessary API endpoints, more and more filters have been developed to enhance the user experience. One such enhancement is the implementation and use of CRT effects.
One of the most advanced filters is the CRT-Royale [TT14] filter for libretro’s RetroArch [Tea] emulator frontend. The CRT-Royale filter features a wide variety of effects typical for old CRT monitors and tries to simulate them with lots of customizability such as user settings for Gamma, Contrast, Bloom, Mask, Anti-Aliasing, and more. While it is currently one of the most advanced filters, it is also a very complex filter that requires certain hardware to run efficiently.
Another popular filter, which we will also briefly discuss in Section 4.2.2, is Timothy Lottes’ [Lot14] public domain CRT filter which features different mask styles such as aperture-grille, stretched and non-stretched VGA-style shadow masks, and more. Compared to other filters like the CRT-Royale, it is simpler, both implementation-wise as well as in the effects it provides, but therefore also easier to run on low-end machines.
While there are plenty of filters dedicated to different CRT displays and shadow masks, there are also filters for specific consoles such as the Amiga. One such specific filter is the A2080 [gue22] filter by guest.r, written for the Amiga emulator WinUAE [TT24].

This filter, different from the others, does not provide a lot of options to tweak the shadow masks or various other aspects of the filter, such as anti-aliasing, but focuses on accurately recreating the Amiga look.

It is important to note that there are a lot of different filter implementations freely available on the internet; the ones listed above are just a subset of the most popular and influential filters available.

CHAPTER 3

Background

In order to understand the significance and challenges that come with upscaling pixel art from its CRT representation, while preserving its original aesthetics, and transforming it into a high-definition version, it is crucial to delve into the fundamentals of pixel art and grasp the differences between traditional CRT and modern display technologies, such as a LCD. In this section, we explore the technology and disparities between CRT and LCDs and introduce the variations in upscaling rasterized and vector graphics. Additionally, we discuss the methods for color correction using look-up tables.

3.1 The Pixel

In layman's terms, a pixel can be described as a two-dimensional little square that has further properties like color, transparency, and brightness [Bli05]. We will discuss the principles of a pixel later in 3.3 when we discuss the fundamentals of CRT and LCDs and how they operate to display pixels on the screen.

3.2 Pixel-Art

With the fundamental principle of what a pixel is in mind, pixel art is "an image where each pixel visible on screen is placed intentionally“ [Sil15]. Pixel art itself became the go-to format for game developers, not because of its aesthetics, but because the technical restraints of early computer hardware did not allow a higher fidelity in computer-displayed imagery. Because of this, pixel art is now not only iconic but still widely used due to its many benefits for indie developers. Since pixel art graphics use a reduced color palette, the inherent file size and processing power needed to display those graphics are largely reduced, making it a very convenient art style for young developers. Similarly, because pixel art games have been around since the very early days, they are regarded as iconic in every aspect [Sil15].

3.3 CRT and LCD Displays

The Cathode Ray Tube Display and the Liquid Crystal Display are two vastly different technologies that originate from very distinct eras.
The CRT and the LCD are two major technologies of recent times used to render and display images on a screen. The CRT display is a so-called emissive display technology, whereas the LCD is a non-emissive display technology [She03].

3.3.1 The Cathode Ray Tube

The Cathode Ray Tube itself was conceived back in 1879 by William Crookes [Tri86], while the idea of using this technology as a display device was first thought of by Ferdinand Braun in 1886 [Tri86].

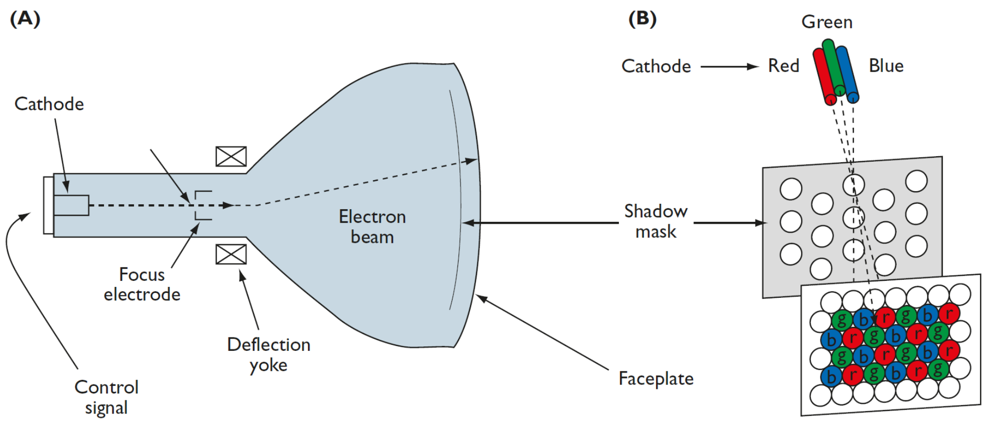


Figure 3.1: The basic principle of a shadow-mask CRT. Image courtesy of [She03]. (A) depicts the basic architecture of the CRT, while (B) shows the relationship between the shadow mask and the cathode rays that are directed onto the phosphor-coated faceplate.

For each primary color – red, green, and blue – there is one cathode, or electron gun, which serves as a source of electrons. The electron beam, for each color respectively, can be modulated by a control signal. Usually, this signal is derived from an input signal originating from an image or video. These beams of electrons are modulated by the input signal, altering the current flowing in the cathodes as well as the electrodes to accelerate, shape, and focus the beam onto the phosphor-coated faceplate [She03].

The electrons from the electron beams of each color pass through a so-called shadow mask. Different kinds of such masks exist from various manufacturers, which together form the pixel structure visible on the screen. We will delve into more detail on this topic below, as it is a vital component of re-creating the look and feel of these old-school displays.

The electrons that pass through the shadow mask excite the red, green, and blue phosphors respectively, creating the pixel color in that manner.

Shadow Masks

Shadow masks essentially determine the pixel structure of the CRT display. They reside between the phosphor-coated faceplate and the inner housing of the Cathode Ray Tube.

The mask's purpose is to ensure that the electron beam for each primary color only excites or activates the phosphors on the screen that are directly assigned to that color's electron beam [Slu98]. To aid in our ability to distinguish individual pixels, regardless of the mask type, a black matrix layer is added [Oak84] [Slu98]. Different screen structures are achieved by placing and distributing either holes or slots over the screen in combination with the aforementioned black matrix layer [Slu98].

A distinct difference between the various mask types lies not only in the layout of the red, green, and blue phosphors but also in the arrangement of the electron guns themselves. The commonly used layouts are the ”Delta Gun,“ ”Inline Gun,“ and ”Trinitron“ configurations, respectively [WC92].

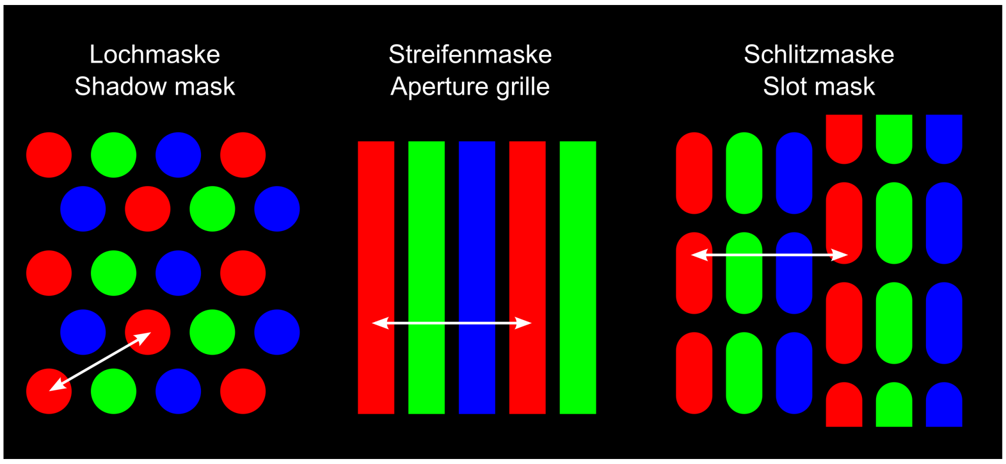


Figure 3.2: Displaying the three most common mask types of CRT displays. Original: ”CRT mask types, ways of measuring dot pitch.“ By Philipp M. Moore. [Phi10]

The dot-triad structure, often referred to as a shadow mask, consists of red, green, and blue phosphors. In the case of the shadow mask, the phosphors are not arranged in stripes but in dots, with a black matrix between the dots [Oak84]. The dot-triad structure employs a Delta Gun, aligning the red, green, and blue electron guns in an equilateral triangle [WC92]. This is why the dotted phosphor arrangement resembles a triangular pattern.

The aperture grille employs three phosphor stripes for red, green, and blue, respectively, in conjunction with a black matrix layer situated between the stripes [Oak84]. The aperture grille was developed alongside a patented electron gun layout known as the Trinitron, introduced by Sony [YOM68]. A major advantage of the aperture-grille mask is its improved transparency, which can enhance brightness by up to 30 percent compared to conventional mask types like the shadow mask.

The Trinitron, a three-beamed single-gun optical system, enables the production of brighter images with superior resolution compared to conventional three-gun systems [YOM68]. While the aperture grille was designed alongside the Trinitron, the Trinitron system is not limited solely to the aperture grille. It can also be used with conventional mask types, such as the shadow mask, resulting in images that are 50 percent brighter with a much sharper appearance [YOM68].

The slot-mask structure is the outcome of rapid advancements in electron optics. Instead of the dotted triangular layout, a striped (slots) arrangement is employed. This alteration in arrangement is attributed to a newly developed electron gun layout known as the Inline Gun [WC92]. As the name suggests, the guns are laid out in line, which explains the change in the mask's structure [WC92].

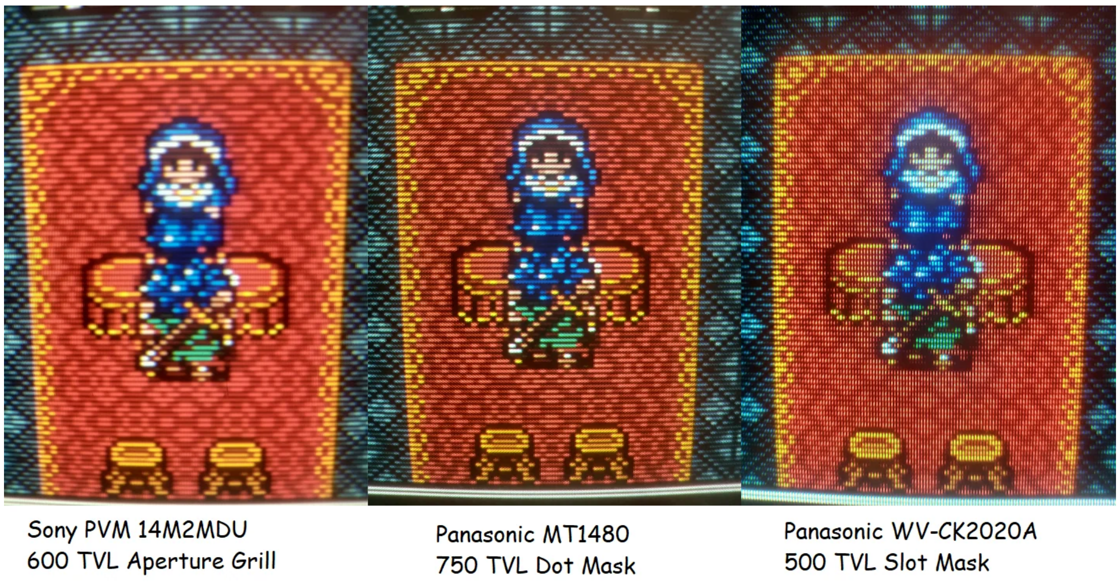


Figure 3.3: Comparison of different mask types (of different TVs) by Reddit user "Reddit_sacks" [Red22].

Pixel-Bleed and Halation

One important side effect of CRT displays is the so-called pixel bleed or halation. A pixel, in the context of a CRT screen, has a spatial profile that extends beyond the borders of the cell [MC92]. While this profile is radially symmetrical and nearly Gaussian, the light output distribution differs when the phosphor is excited by the electron beam [NM92]. One major problem that pixel bleed introduces is the reduction of contrast because darker

parts of the image displayed on the CRT are lighter, and small bright areas are perceived bigger than they are [UVD88, MC92].
If we consider the luminance contribution of any given pixel to its neighbors, the spatial profile of a stationary electron beam can be expressed as a Gaussian function, centered around the pixel location. However, while this Gaussian function holds as a good approximation for the vertical profile, it is less accurate for the horizontal direction, which is the direction the electron beam moves. The interaction of adjacent pixels in the scanning direction of the electron beam varies from CRT to CRT and also features vast nonlinearities on some CRT displays [MC92]. However, Naiman et al. [NM92] show through measurements, that even though there are nonlinearities, the spatial horizontal pattern is still qualitatively Gaussian. The cause of pixel bleed lies in the angles with which the light rays strike the glass-air interface of the display and the reflection and refraction caused. If the light ray is perpendicular to the glass-air interface, it will pass through and exit the display undeflected. In case the angle of incidence of the light ray hitting the glass is oblique, it will be refracted away from the perpendicular ray, towards the tube face. Essentially, the larger the angle of incidence is, the more the ray will bend away from the normal of the glass face. In case the angle of incidence reaches a critical angle, it will no longer be refracted by the glass face but rather reflected back onto the phosphor layer where it excites the phosphors and diffuses again in all directions, bleeding into the neighboring pixels [UVD88]. This effect continues consequently until no perceivable light is reflected. If viewed from just a singular pixel, the effect will form a dim circle around the pixel, hence the name halation, as it appears like a halo around the pixel with Gaussian distribution.

While more effects occur on a CRT display, such as moiré patterns, geometric distortions, screen flickering, and more, these effects and their influence are outside the scope and interest of this work.

3.3.2 Modern Displays

Nowadays, Cathode-Ray-Tube displays have been superseded by Liquid Crystal Displays LCD and even more modern types of display technologies, such as LED and OLED display devices, respectively. These work fundamentally differently from CRT display devices. Since these display technologies are highly complex on their own and have various sub-types, we will focus more on the Liquid Crystal Display to illustrate the general technological differences when compared to an old-school CRT device.

A fundamental distinction between the newer LCD technology and the CRT lies in the fact that the LCD device does not require any form of an electron gun to function. This feature enables the display to be much slimmer in-depth than the CRT.

Figure 3.4 illustrates the basic principles of the layers and technologies employed by an LCD. In particular, this Figure presents the basic architecture of a “twisted nematic”

(TN) LCD. We will now proceed to elaborate on the fundamental components of the TN Liquid Crystal Display, as depicted in Figure: 3.4.

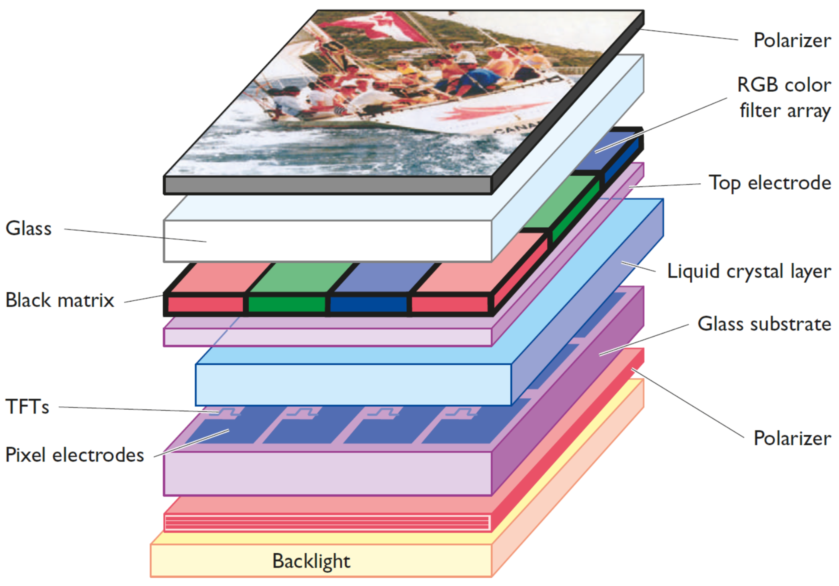


Figure 3.4: Architecture showing the basics of a liquid crystal display. Image courtesy by [She03].

The backlight of the LCD is responsible for illumination. Usually, the backlight consists of a fluorescent lamp, which offers the advantage of high luminous efficiency and the ability to tailor the spectral power distribution as needed [She03] [Kaw02] [Che11].

The incoming light is then plane-polarized by a polarizer. This polarized light passes through the thin-film transistor array, which is used to apply voltage to the Liquid Crystal (LC) layer [She03] [Kaw02] [Che11].

The liquid crystal layer is highly complex and dependent on a variety of material and geometrical parameters. However, the spectral transmission of this layer also relies on the applied voltage, which is a key principle for the display [She03] [Kaw02] [Che11].

The liquid crystal cells polarize the light based on the applied voltage. When no voltage is applied to the liquid crystal cell, the liquid crystal optically rotates the axis of polarization of the incoming light, which originates from the backlight and is linearly polarized. However, applying voltage above a certain threshold to the liquid crystal allows the dielectric anisotropy of the liquid crystal to deform it in a way that destroys the rotated structure of the liquid crystal display and aligns it with the incoming light [She03] [Kaw02] [Che11].

Due to this change in light polarization, the exit polarizer also referred to as the "analyzer," assesses the polarization state of the exiting light. If the light is polarized parallel to the analyzer's polarization, it is transmitted through; however, if the light is polarized perpendicular to the analyzer's polarization, it is extinguished [She03] [Kaw02] [Che11].

3.4 Scaling Graphics

Scaling images, including pixel art for our purposes, is a field that continues to receive considerable attention and development for better and/or faster algorithms. Despite the thorough exploration of this field and the existence of numerous algorithms, scaling pixel art naturally to preserve its appearance remains quite challenging and has not been fully achieved to this day.

Typically, two distinct kinds of graphics undergo upscaling:

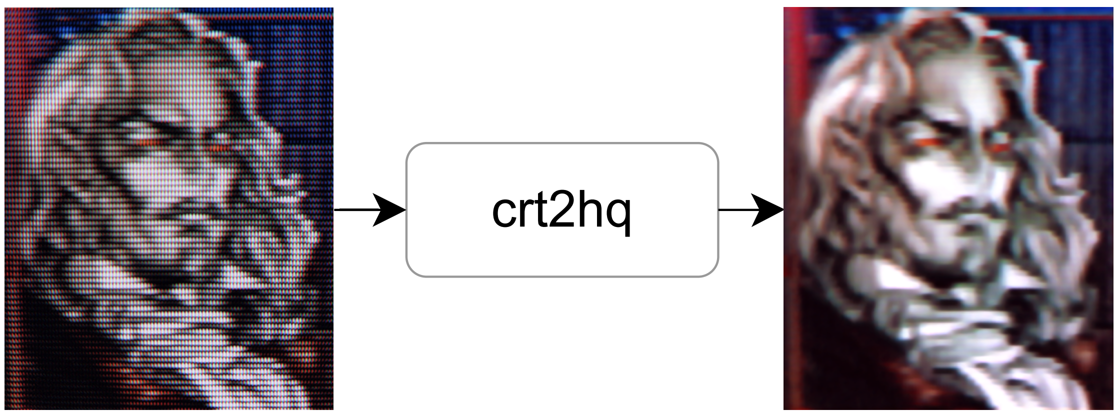
- Rasterized Graphics
  Rasterized graphics are typically images that are defined by a pixel matrix on a 2D image plane. This means that the number of pixels a rasterized graphic or image contains is finite and determined by the size of the image raster.
- Vector Graphics Vector graphics are, as the name suggests, based on vectors that define the position of a given color, rather than a pixel matrix. Because of this characteristic, vector graphics can be scaled more easily than their rasterized counterparts.

3.5 LUT - Look-Up Table

A LUT, also known as a Look-Up Table, is, as the name suggests, a table that, in the context of image processing and color correction, transforms the image colors based on the table. The benefit of such tables is that they can be easily applied and changed if needed. There are different types of tables, ranging from 1D LUTs to 3D LUTs. A 1D LUT can be compared to a gamma curve transformation, whereas a 3D LUT is capable of transforming or mapping the hue, saturation, and brightness, each represented on one axis.

CHAPTER 4

Implementation



We propose “crt2hq”, an approach for upscaling and converting CRT-based images into high-quality versions while still maintaining the intrinsic look and feel of the CRT. The key idea of our approach is to take advantage of a residual neural network, SRResNet as described in 2.1.2, and its upscaling capability in conjunction with a custom-made dataset comprised of two sets of images. One set of the images represents the upscaled target images while the other set consists of the same lower-resolution images but with a custom software CRT effect applied to them. The goal was to then train the network with this dataset to recognize the CRT patterns on any given input image and convert and upscale them into a high-resolution representation. To this end, we implemented seven different CRT effects to mimic as many different display types as possible, which were then used for training the network to not gain any bias towards certain CRT displays that might have occurred if only one CRT-based effect were used. Additionally, we trained a second network which acts as a cleanup filter. The idea here is that not all CRT images might be of the same quality and the CRT pattern might have different scales. Therefore, we trained a second network with faded or blurry CRT patterns. This is to help in two ways. Firstly, it helps clean up the image from CRT marks and allows for a wider range of input images; secondly, it sharpens the image, as we discuss later, the target images used for training the network are still sharp and high-quality representations of the inputs.

4.1 SRResNet Model

Our implementation is written in the Python programming language [pyt]. We chose Python for its versatility and deep integration with existing neural network libraries such as PyTorch, which we utilized for this work. PyTorch enables quick and efficient neural network programming and is described as an "End-To-End Machine Learning Framework" [Fou].

We structured the network into different modules representing the building blocks outlined in 2.1.2. In PyTorch, a "Module" serves as the base class for any network model, including our sub-blocks. Modules can be nested in a tree-like structure and used as attributes inside the class. With this structure in mind, we created four different classes where three implement the building bocks: "ConvolutionalBlock", "SubPixelConvolutionalBlock", and "ResidualBlock", and the fourth being the implemented model itself. Each of these classes only has two methods: the Python built-in "___init___" method, which acts as a constructor to set all parameters used in the class as well as to set up the different layers of the block, and the PyTorch method "forward" which is used to define the forward propagation of the module.
All of the layers and PyTorch equivalent classes mentioned below are part of the PyTorch package "nn" [tor23]. This package encompasses most of the basic PyTorch functionality used to construct graphs and neural network models.
The ConvolutionalBlock class was designed to be versatile, comprising a convolution layer, an optional batch normalization layer, and an optional activation layer. Using a standard Python list, one can append different layers, predefined in the PyTorch library, such as "nn.Conv2d" for 2D convolution or "nn.BatchNorm2d" for batch normalization, to this list. By setting the list as a parameter in the constructor of "nn.Sequential" PyTorch automatically sequences the layers and facilitates easy extension or modification of the layers of this block.
Similarly, the SubPixelConvolutionalBlock is set up using PyTorch equivalents of the outlined blocks from Section 2.1.2, namely "nn.Conv2d", "nn.PixelShuffle" and "nn.PReLU" activation layer. This block is ready to use in just ten lines of code.
Analogue to the other two blocks, the ResidualBlock is constructed using the ConvolutionalBlocks created earlier. However, in this case, the ConvolutionalBlocks are utilized sequentially in the forward method of the ResidualBlock. The initial input is added at the end of both ConvolutionalBlocks as a residual. The structure of the ConvolutionalBlocks within the ResidualBlock follows the specifications outlined in Section 2.1.2. Specifically, the first ConvolutionalBlock includes a batch normalization layer and a PReLU activation layer, while the second block consists solely of a batch normalization layer.

```python
class SRResNet(nn.Module):

    def __init__(self, large_kernel_size=9, small_kernel_size=3, n_channels=64,
    n_blocks=16, scaling_factor=4):
        [...]

    def forward(self, lr_imgs):
        """
        The forward propagation of the SRResNet.

        :param lr_imgs: The LR image to be upscaled. A tensor of size (N, 3, w, h)
        :return: SR images upscaled from the LR input. A tensor of size (N, 3, w *
    scaling factor, h * scaling factor)
        """

        # (N, 3, w, h)
        output = self.conv_block1(lr_imgs)

        # (N, n_channels, w, h)
        residual = output

        # (N, n_channels, w, h)
        output = self.residual_blocks(output)

        # (N, n_channels, w, h)
        output = self.conv_block2(output)

        # (N, n_channels, w, h)
        output = output + residual

        # (N, n_channels, w * scaling factor, h * scaling factor)
        output = self.subpixel_convolutional_blocks(output)

        # (N, 3, w * scaling factor, h * scaling factor)
        sr_imgs = self.conv_block3(output)

        return sr_imgs
```

Figure 4.1: Code shows the forward propagation of the SRResNet and how the different building blocks are used in code, as outlined in the diagram of Figure: 2.7.

4.2 Data Processing

To properly train the network, we had to prepare a dataset first. The training images for our dataset were sourced from the supplements of the paper “Make Your Own Sprites: Aliasing-Aware and Cell-Controllable Pixelization” [WCZ+22]. The dataset features a wide variety of images, including both pixel art and non-pixel art, collected from sources such as the CartoonSet [RBG+17] or the Abstract Scene dataset [ZP13]. This base dataset was then modified and processed to suit our needs. The SRRes network is a super-resolution network, meaning it takes a small input image and a larger target image, and then trains the weights accordingly based on those inputs. For each entry in our training set, we must prepare two groups of images. The first group serves as the input to our model, while the second group represents the desired output, also referred to as the target. To avoid bias towards a single CRT filter, we created a total of seven different filters, resulting in a training-to-target ratio of 7:1. The raw dataset consisted of 10,969 images with resolutions ranging from 256x256 to 512x512. With seven different CRT masks, this resulted in a total of 76,783 training images in our dataset. Similarly, we have 10,969 target images ranging from a resolution of 512x512 to 1024x1024, as well as

1024x1024 to 2048x2048 for our second network respectively. Neither the training nor the input images have an alpha layer. Therefore, our implementation does not utilize the alpha layer for either training or testing purposes.

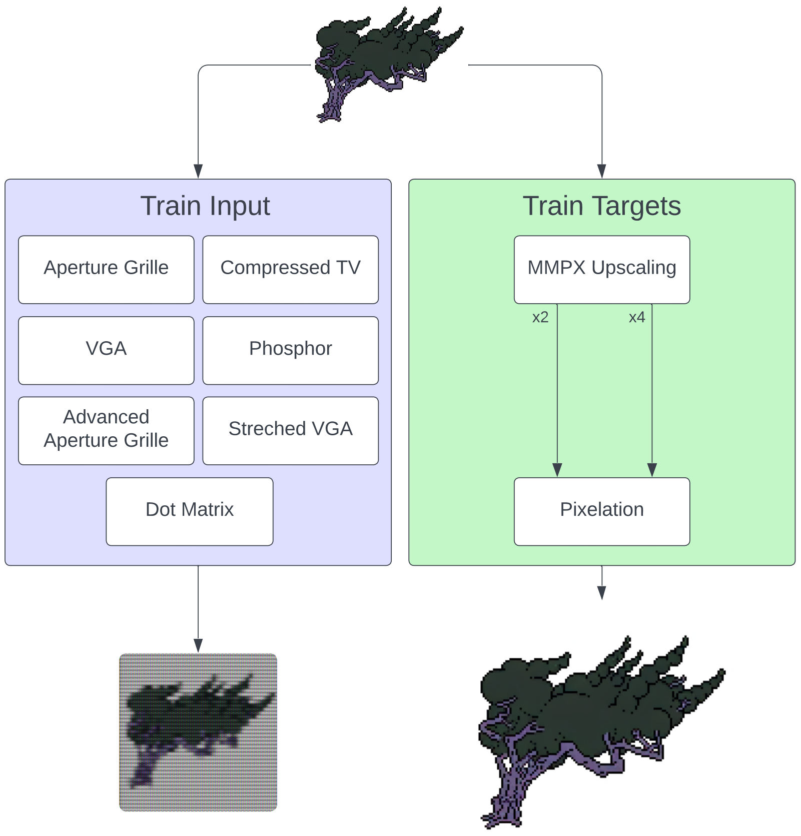


Figure 4.2: The figure illustrates how our dataset was modified from the base dataset. The training target undergoes upscaling and then pixelation to ensure the elimination of upscaling artifacts. The training inputs are generated by processing the base image with seven different CRT filters each.

It should be noted that the following explanations of our training procedure, as well as the input and target images, represent the best results we have achieved. We attempted several different training procedures with input and target images of varying sizes, and we applied the pixelation effect at different stages. However, none yielded results that came close to what we present below. In particular, we tried to downscale the input images by a factor of two and pixelate them afterward, instead of upscaling and pixelating the target images. The rest of the training pipeline remained the same, including the application of our CRT filter. While this approach yielded better color accuracy overall, it didn't process the CRT patterns as efficiently and accurately, leading us to pursue the following solution for input and training images.

4.2.1 Training Targets

For the training targets, we had two options: either downsampling, in conjunction with pixelation, using the method from Wu et al. [WCZ$^+$22] to reduce the original images by a factor of two and pixelating them, while keeping the target images the same size as before, or upscaling the target images by a factor of two and leaving the training input images unchanged in size.

In this work, we opted for the latter option. Given that our base dataset primarily comprised images sized 256x256 and 512x512, where a significant portion already consisted of pixel art or pixelated images, we decided to upscale the target images by a factor of two to enhance the quality with higher-resolution images. Our dataset consisted of 4235 512x512 images and 6734 256x256 images, respectively. The 256x256 images were of pixel art nature.

To achieve the upscaling, we utilized the MMPX upscaling algorithm, as described in 2.1.1, and then passed the upscaled images through the pixelation network by Wu et al. [WCZ$^+$22]. While it may seem counterproductive initially, given that our primary objective is to detect the CRT pattern and upscale the pixel art image into a higher resolution representation where the pixel art style may not be desired, our intention was not to completely repixelize the target images but rather to address upscaling artifacts like misplaced pixels at the edges.

The effectiveness of this approach depends significantly on the image resolution. The pixelation network is primarily designed for lower-resolution images, where detecting and repixelizing graphics is more straightforward. Given that our images are larger than the preferred resolution, the repixelization aspect is almost nullified, while the network effectively cleans up the image by addressing upscaling artifacts on the pixel art boundaries. Additionally, due to the image size, the network even smooths out the pixelation of the upscaled image, minimizing the pixel art effect for our target images. However, employing the pixelation network results in a slight color loss compared to the original image. It should be noted, however, that employing the pixelation network generally has minimal effect due to our large image resolutions. While we did include it in our final dataset creation, omitting it would have yielded the same results.

4.2.2 Training Inputs

As previously discussed, for each of the training targets, there are seven different input images. Each of the seven images has a different CRT mask representing different types of displays, respectively. To achieve this, we implemented the filter as a post-processing effect which is directly applied to the image. This filter was written in the OpenGL Shading Language (GLSL) using OpenGL as the graphics API. We chose OpenGL for this task due to its simplicity and efficiency in creating a small filter application for preprocessing tasks like this one.

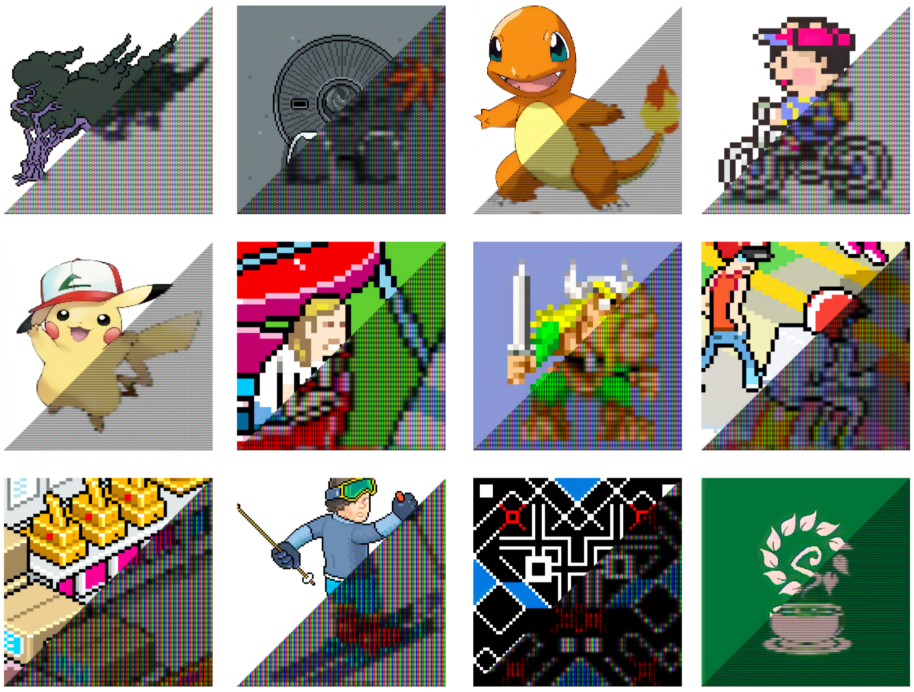


Figure 4.3: Images from our dataset which exhibit different kinds of CRT filters, in various sizes, with not all adhering strictly to the pixel-art aesthetics. It is important to feature various kinds of images with different colors and contrasts to better train the model.

The masks were chosen based on our research, as extensively discussed in Section 3.3.1. The chosen masks encompass a range of display types that were prevalent during the era of the first pixel art games. Several of the implemented shaders (b) to (e) in Figure: 4.4 were originally published to the public domain by Timothy Lottes [Lot14] and modified for our purposes. The phosphor mask, (f) in the same Figure, was originally crafted by “libretro”[Tea24] user “guest.r”[gue19] and also modified for our purposes.

The shader was crafted to allow easy modification of various mask features, such as brightness, mask scale, or scanline density, enabling high customizability for fast and efficient image creation. To calculate the mask for a given input, several steps are necessary. Since we are dealing with a fragment shader, the mask and scanline are calculated and applied per fragment shader invocation. The initial step involves computing the mask. What remains consistent across all seven masks is how the invocation index of the fragment is utilized. We leverage the invocation index to determine the color component of the mask. This position is then scaled by an input parameter called “i_maskSize”, allowing for adjustments to the mask size for differently sized images or artistic purposes. Once we determine the scaled position, we normalize it to the number of intervals, which

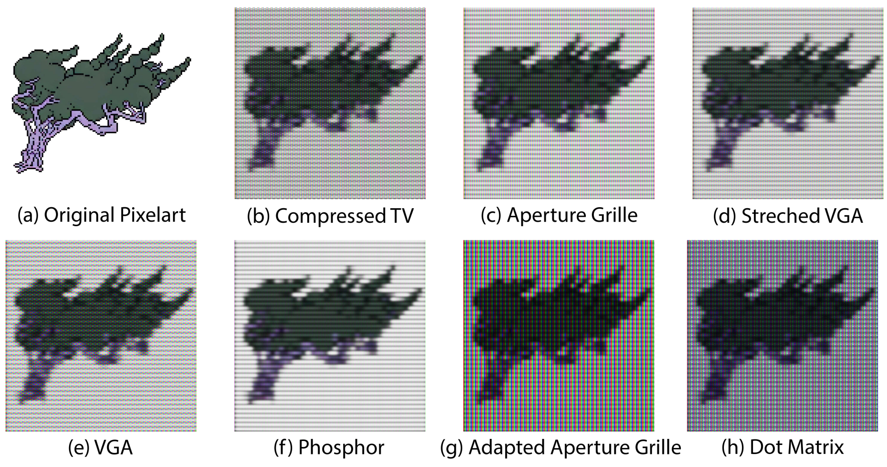


Figure 4.4: This Figure shows the different kinds of CRT masks we have implemented for the training dataset. The aperture grille version (c) was already present in Figure: 4.3.
(a) The raw pixel art without any effects applied.
(b) A compressed TV CRT mask.
(c) An aperture grille CRT mask without a black matrix.
(d) Stretched version of a CRT screen where the image was transmitted over VGA.
(e) Non-stretched version of a CRT screen where the image was transmitted over VGA.
(f) A CRT mask that uses just the raw Phosphors as colors.
(g) An adapted Aperture grille mask featuring a black matrix as described in 3.3.1.
(h) A Dot-Matrix CRT mask with an included black matrix.

is predetermined by the type of mask. An interval represents the different mask color components. For example, the Aperture-Grille mask shown in Figure 4.5 has an interval of three because the red, green, and blue components are adjacent to each other. If one were to include a black matrix between each color component, the interval would become six, as each color component is followed by one black component.
By taking the fractional part, we normalize the index from zero to one based on the number of intervals. This allows for the layout of the mask such that depending on the position (now between zero and one), either the red, green, or blue channel of the mask is active or inactive. This can be further expanded by utilizing not only the x-component of the position but also the y-component, enabling the creation of complex masks with minimal effort.

In our implementation, we use predefined light and dark values for the brightness of the different color channels of the mask, depending on whether they are active or not. However, this can be enhanced by sampling the image at the given position to reflect the intensity and color of the image to which the effect is applied.
Once the mask color is determined for the current fragment, the next step is to calculate the scanline intensity. The scanlines are computed by utilizing the current fragment

invocation index, similar to the mask calculation. A 1D Gaussian blur is applied to the pixels to the left and right of the current fragment, with a 3-tap, 5-tap, and 3-tap Gaussian blur being applied, respectively. The term “3-tap, analogous to "5-tap," indicates that the filter kernel has three (or five for the 5-tap kernel) non-zero coefficients, respectively. A Gaussian blur is a common image processing technique used to reduce image noise and detail by averaging the pixel values in the vicinity of each pixel. This averaging is weighted by a Gaussian function to give more importance to nearby pixels.
In a 1D Gaussian blur filter, the kernel is typically centered around the current pixel and weights the neighboring pixels according to a Gaussian distribution. The weight of the individual blurred results is determined based on the distance of the current pixel and the invocation index. Finally, the calculated scanline value, a three-component color vector, is multiplied with the previously calculated mask to form the final output.
It should be noted that while the various parameters allow for a lot of customization for each mask type as well as more artistic freedom, for creating our dataset specifically, all values were left at their default settings, as we will discuss later in Section 4.4.2.

4.3 Training Procedure

For our approach, we trained two separate networks that work together as one. Both networks were trained with the same parameters: 2×10^6 iterations and a learning rate of 1×10^{-5}. The content loss criterion used was MSE-Loss (mean squared error or squared L2 norm) when comparing the Super Resolution (SR) image with the target image. Since we minimize this loss, and the comparison is between the SR and the target image, we change the parameters of the network in such a way that when we re-input the Low Resolution (LR) image again, it will create an SR image that is closer to the target image than it was before. The optimizer used was Adam [KB17], a stochastic gradient-based optimizer.
The batch size for each trained model was set to 16. Regarding the model parameters, a kernel size of nine was chosen for the first and last convolutions, which transform the inputs and outputs, respectively. The size of the kernels used for all in-between convolutions such as in the residual and subpixel convolutional blocks, was set to three. The number of channels was selected as 64 for the in-between input and output channels, such as the residual and subpixel convolutional blocks. The model consists of a total of 16 residual blocks.
Additionally, both networks were trained with two different datasets, each dependent on the other. Firstly, the network was trained using the CRT images as training input and the two times upscaled, non-CRT version as the target. Secondly, another SRResNet was trained, but this time, the outputs of the first network were used as the training input and a four-times upscaled, non-CRT version was used as the target. This means two differently scaled target images were used: one 2x and one 4x.

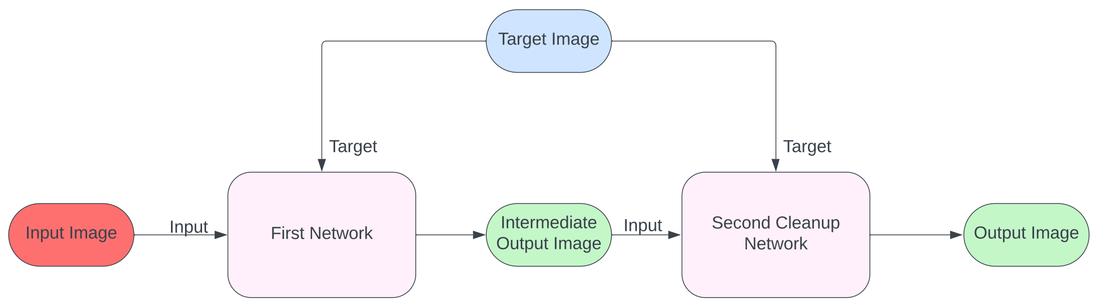


Figure 4.6: The pipeline illustration depicts the training process where both our networks utilize the intermediate output of the first network as the input image for the second network. Throughout this process, the same target image is employed for training both networks.

For the second network, the input data comprised the output of the first network, which was run with the same training dataset. While this isn't a typical approach, the second network aimed to create a clean-up network to remove smears, incorrect pixels, and residuals that may still be visible from CRT filters.
Consistently, the training procedure for both networks was the same. For each iteration, the input was cropped into random chunks of size 24x24. The target images underwent a similar cropping process, albeit with size adjustments based on the scaling factor, which in our case is two. This ensured that the target images maintained proportionality and consistency with the input images, facilitating accurate training of the network. By aligning the dimensions of the target images with those of the input images, we ensured that the network learned to produce output that corresponded correctly to the desired upscaled versions of the input images.
The cropping of images offers several benefits for training our network. Firstly, it enables data augmentation, effectively diversifying our dataset by introducing variations in image content. Secondly, it enhances training efficiency by utilizing more patches of smaller images, allowing the network to learn from a wider range of data samples. Additionally, cropping helps standardize image sizes during training, ensuring consistency across the dataset. Overall, these benefits contribute to improving the robustness and effectiveness of our network during the training process.

4.4 Parameters and Workflow

In our implementation, we prioritized simplicity and user-friendliness as overarching goals. This guiding principle was upheld throughout the development process, reflected in both the resultant application, crt2hq, and the software CRT filter used in generating our dataset. As a result, both crt2hq and the software CRT feature very few parameters that can and need to be tweaked. All parameters are supplied as command-line parameters, also known as command-line arguments or command-line options. These parameters are provided to the program when it is executed in a command-line interface (CLI) or terminal. This design choice enhances ease of use and allows users to interact with the program efficiently.

4.4.1 crt2hq

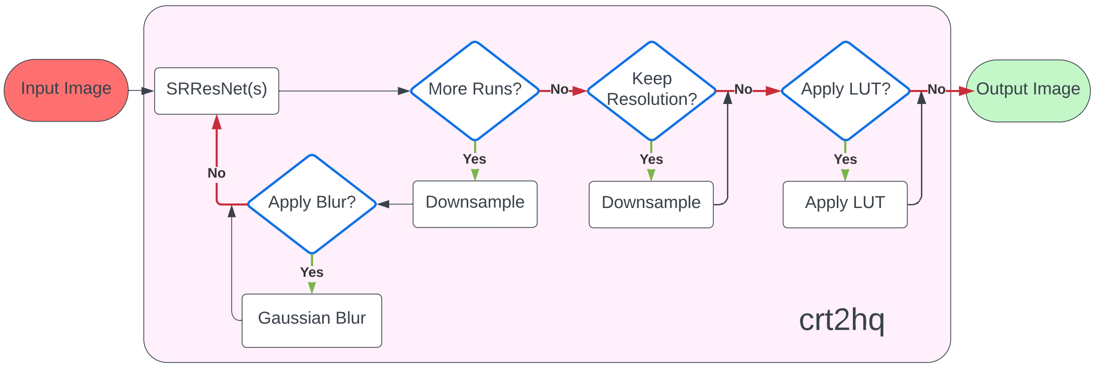


Figure 4.7: Pipeline image demonstrating how our approach, crt2hq, works.

Upon evaluating our trained model, it became apparent that merely inputting an image into the network did not yield the desired outcome. Consequently, we modified the program to run the input image multiple times through the trained model. However, this approach sometimes resulted in over-sharpened images or harsh artifacts, particularly with certain types of CRT images. To address this issue, we introduced a slight blur applied after each iteration the image is sent through the network. This blur has become a vital point for our approach as we will discuss later in Section: 5.4.

Parameter	**Description**	**Type**
-i	The input file or directory.	String
-o	The output file or directory.	String
-r	The number of runs (iterations) for the network.	Integer
-b	The blur radius for in-between blur. Helps smooth the output. 0 = Disable.	Float
-s	If set, skips the first crt filtering step. (Useful for crt images on the internet.)	Boolean
-k	If true, it keeps the same size as the original image.	Boolean
-l	The path to the HALD (LUT) image used for color correction.	String

Table 4.1: Showcasing the input parameters of our program, crt2hq.

The most notable parameters include “-r” which determines the number of times the image is fed through the network, and the parameter “-b” which sets the blur radius of the applied blur between each run. Additionally, “-l” enables the application of LUTs.

This feature was implemented because our network may experience color or brightness loss at times. Utilizing a LUT image allows us to address these shortcomings. Depending on the input image used, the number of runs and the blur radius can significantly impact the resulting image.

4.4.2 CRT Filter

As previously mentioned, we have developed a highly versatile shader that incorporates seven different CRT mask types. To streamline the process of creating and testing CRT input images for our dataset, we have enabled the easy adjustment of shader parameters. Although the filter can warp the image to simulate the appearance of a curved CRT

Parameter	**Default Value**	**Description**
i_scale	14	The scale of the overal image. Used to apply the proper amount of scanlines to the image.
i_hardScan	-15	The sharpness of the scanlines.
i_hardPix	-1.5	The sharpness scale used for the gaussian blurs.
i_maskLight	1.5	The brightness of the pixel when light shines through the mask.
i_maskDark	0.5	The brightness of the pixel when the mask covers the current pixel.
i_maskSize	3	The size of the mask itself.
i_warpX	32	Warping in X direction. Used to recreate a CRT screen more faithfully.
i_warpY	64	Warping in Y direction. Used to recreate a CRT screen more faithfully.
i_maskType	1	The mask type to use. Value between 1-7.

Table 4.2: The parameters used by our CRT shader and their default values have been used when generating the training dataset.

display, we did not utilize this feature for our purposes. However, it can be enabled if required. This decision was made because our images are relatively small, and adding curvature to them would offer minimal to no benefit for training the model.
The default values provided in Table: 4.2 were used as parameters when creating our dataset. The only parameter that changed during training was the “i_maskType” parameter, which tells the shader which mask type to apply.

```glsl
// Aperture-grille without black matrix
vec3 mask_ApertureGrille(vec2 position) {
  //scale the current fragment position by using
  //an input (uniform) i_maskSize
  vec2 pos = floor(position / i_maskSize);

  //the number of intervals
  //this lets us choose how many sub-pixels we want to have
  float num_intervals = 3.0;

  //we take the fractional part of the x coordinate
  //this allows us to map 0-1 to one fragment
  pos.x = fract(pos.x / num_intervals);

  //the mask variable will be the return value.
  //here its initialized with just dark
  vec3 mask = vec3(i_maskDark, i_maskDark, i_maskDark);

  //i_maskLight = the brightness of the pixel when light shines through the mask
  //i_maskDark = the brightness of the pixel when the mask covers the current pixel

  //now we can map the x position to thirds which will be our RGB
  //Aperture-Grille mask
  //this same principle can be extended into 2D with the y axis
  if(pos.x < 0.333)  {mask.r = i_maskLight; }
  else if(pos.x < 0.666) { mask.g = i_maskLight; }
  else { mask.b = i_maskLight; }
  return mask;
}

// Calculate the scanline intensity at the given position
vec3 scanline(vec2 position) {
  //calculate the blur colors of neighboring pixel
  vec3 a = gauss3Tap(position, -1.0);
  vec3 b = gauss5Tap(position, 0.0);
  vec3 c = gauss3Tap(position, 1.0);

  //get the weights of these pixels
  float wa = scanWeight(position, -1.0);
  float wb = scanWeight(position, 0.0);
  float wc = scanWeight(position, 1.0);

  return (a * wa + b * wb + c * wc);
}

// Return scanline weight.
float scanWeight(vec2 position, float off) {
  return exp2(dist(position).y + off, i_hardScan * i_hardScan);
}

// Calculates the distance of the emulated pixel to the neares texel
vec2 dist(vec2 position) {
  //Calculate the resolution using the input variable i_scale, iResolution.xy is the resolution of
     the input image
  vec2 res = iResolution.xy / i_scale;
  position = position * res;
  return -((position - floor(position)) - vec2(0.5));
}
```

Figure 4.5: This code snippet illustrates how the aperture-grille mask intensity and color are calculated based on the fragment invocation index. The position is normalized to determine which color component of the mask should be active at the given fragment invocation index. Finally, the function returns a 3D vector representing the mask color. Additionally it depicts how the scanline is calculated by using different Gaussian blurs together.

CHAPTER 5

Evaluation

In this chapter, we evaluate and analyze our proposed approach of refining CRT based images into a higher definition version, by using a reference CRT images sourced from the internet, as well as a software CRT image created using our custom CRT filter implementation, which was used to generate our training dataset, outlined in Section 4.2. We compare the results against each other and assess how well the CRT images are filtered, including both the reference images and our own implementation. Additionally, we evaluate the resulting images based on the accuracy of the CRT image processing and how they compare to the original pixel art. We also analyze why some CRT images, both sourced and with our filter, might not look as expected, and identify the sources of these discrepancies.

5.1 Experimental Setup

As briefly stated, for our experiments, we sourced several reference CRT images as well as the original pixel art from the internet. We then analyzed the reference images and created our own CRT image using our software CRT filter, which was applied to the original pixel art.
These images were then processed using our proposed approach, crt2hq. Both processing runs were executed with the same parameters per image, respectively. However, the first iteration did not use a Look-Up Table (LUT) color correction at the end, whereas the second run applied a custom LUT. This LUT was created based on the results of the first run, to evaluate whether it is possible to accurately recreate colors as well. This already suggests some shortcomings or color loss in our approach, which we will discuss in more detail in Section 5.2.

5.1.1 Test Dataset

We chose three different games that stem from the SNES and PlayStation 1 era. These games were originally developed and played on CRT displays, making them ideal for our purposes. The games in question are:

- Castlevania: Symphony of the Night.
  Developed by "Konami"[Kon] for the Playstation 1.
- Final Fantasy VI.
  Developed by "Squaresoft", nowadays known as "Square Enix"[SQU] for the SNES.
- Chrono Trigger.
  Developed by "Squaresoft", nowadays known as "Square Enix"[SQU] for the SNES.

From each game respectively, we sourced one CRT image, which was captured using a professional camera displayed on an equally good-quality display. We then sourced the corresponding original pixel art or sprite sheet and applied our own software CRT effect. Every software CRT filter used was tuned to look as closely as possible to the reference image. Figure: 5.1 shows the original, the reference CRT, as well as our CRT-filtered image.

The reference image of Castlevania: Symphony of the Night and Chrono Trigger were originally captured by X user "CRTPixels" [CRT21], the Final Fantasy VI image used was captured by X user "Ruuupu"[Ruu21]. Table: 5.1 shows the values with which we created our CRT replicas of the reference image. To understand the differences in values and what they do individually, refer to Section: 4.4.2.

Image	**i_scale**	**i_hardScan**	**i_hardPix**	**i_maskLight**	**i_maskDark**	**i_maskSize**	**i_maskType**
Castlevania	13	-10	-1.1	3	1.5	3	6
Final Fantasy	14	-15	-1.5	1.5	0.4	3	6
Chrono Trigger	26	-10	-1.0	1.7	0.5	3	2

Table 5.1: The parameters used when we created our CRT images were based on the reference images.

Because the original pixel art images are naturally very small in size, we applied a similar process as we did for our training targets, outlined in Section: 4.2.1, where we upscaled and pixelated the images again. We did this mainly because our sourced reference images, captured by a camera, did not adhere strictly to pixel boundaries and were naturally larger. We attempted to match the sizes and actual crop of the images as closely as possible to ensure a fair comparison. Only the Chrono Trigger image has a slight discrepancy in size due to upscaling and cropping of the original pixel art.

All images used were of the Portable Network Graphics (PNG) file format as it is one of the most commonly used formats and offers lossless compression of the images. However, all of our references as well as original pixel art images had different widths and heights.

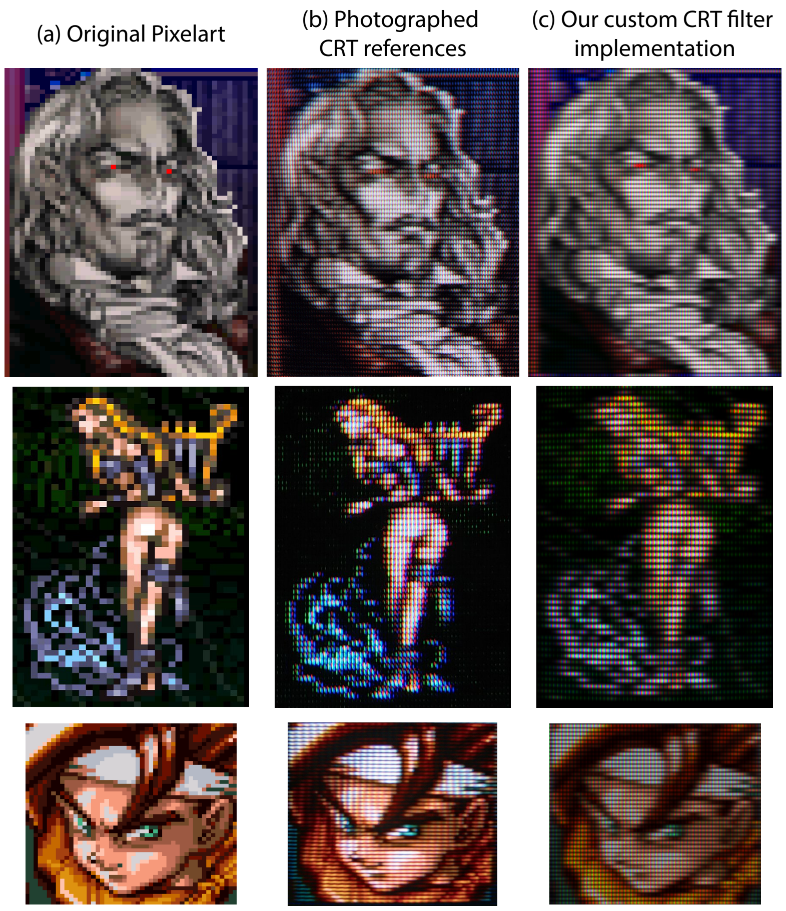

(a) Original Pixelart
(b) Photographed CRT references
(c) Our custom CRT filter implementation

Figure 5.1: The test images which were used to evaluate our approach.
(a) The raw pixel art.
(b) Photographed CRT references. The first and the third images were captured by X user “CRTPixels” [CRT21]. The second image was captured by X user “Ruuupu”[Ruu21].
(c) Our recreation of the reference CRT image by using our custom software CRT filter, which was used to generate the training dataset, on the original pixel art.

This is important to note as we will later discuss how the image size has a significant effect on both the runtime of our approach and the RAM consumption in Section: 5.3.

5.2 Results

Figure: 5.2 showcases our results. To properly compare and evaluate our results, we used the original pixel art image, upscaled it by a factor of two using both bilinear and the hq2x algorithm, and compared those results with our approach, processing both the reference and our own CRT filtered image. The images (e) to (g), enclosed within the highlighted red box, showcase our results using crt2hq on the input image in (a).

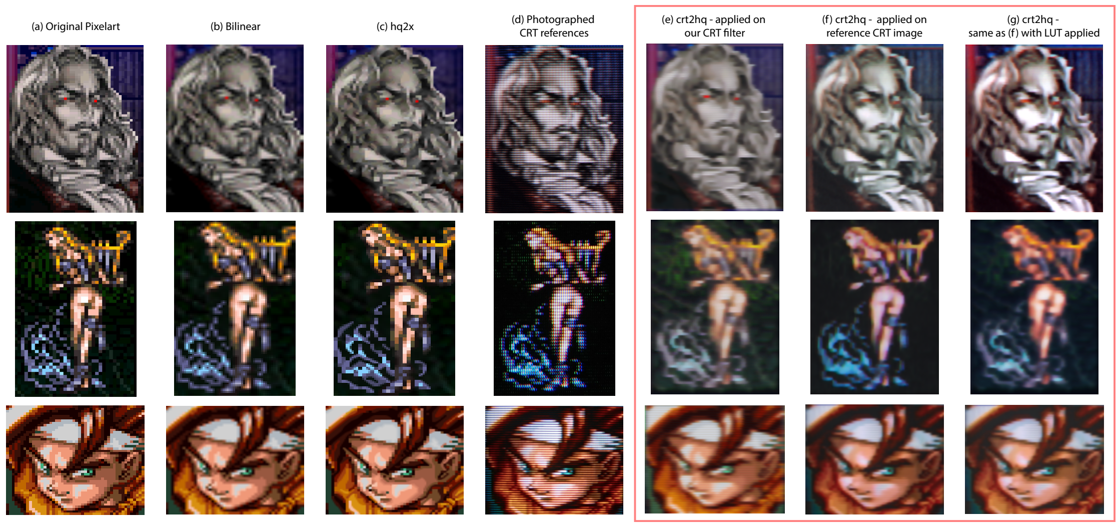

(a) Original Pixelart
(b) Bilinear
(c) hq2x
(d) Photographed CRT references
(e) crt2hq - applied on our CRT filter
(f) crt2hq - applied on reference CRT image
(g) crt2hq - same as (f) with LUT applied

Figure 5.2: (a) The raw pixel art.
(b) A two times upscaled version of the raw pixel art using a standard Bilinear filter.
(c) The hq2x upscale algorithm applied on the raw pixel art.
(d) Photographed CRT reference image.
(e) Our result using our recreated CRT images from Figure: 5.1 (c) as input.
(f) Our result with the photographed CRT reference image as input.
(g) Our result with the photographed CRT reference image as input with a custom LUT applied to resemble the original colors.

5.2.1 Discussion of Color Loss and Correction

With the right parameters supplied, our approach delivers satisfactory results in terms of upscaling and maintaining the CRT style while simultaneously removing the CRT pattern successfully. However, when we compare the images in rows (e) and (f) of Figure: 5.2 to their original pixel art, it becomes clear that the resulting images become somewhat faded in color and contrast. The color loss can be traced back to our SRResNet model. To address this, as previously stated in Section: 5.1, a LUT was applied for color correction at the end of the process in images of column (g) using the reference CRT image.

Image	Runs	Blur	Skip
Castlevania (Our CRT)	3	2.2	Set
Castlevania (Ref. CRT)	3	2.0	Set
Final Fantasy IV (Our CRT)	4	2.1	Set
Final Fantasy IV (Ref. CRT)	4	2.0	Set
Chrono Trigger (Our CRT)	3	4.0	Set
Chrono Trigger (Ref. CRT)	3	4.0	Set

Table 5.2: The parameters used for crt2hq when processing the test dataset.

5.2.2 Impact of Parameters

To achieve satisfactory results as shown in the above Figure, it is necessary to adjust the provided parameters as outlined in Section: 4.4.1. Different parameters had to be supplied for each image processed. Additionally, due to the number of runs that our approach takes and the slight blur introduced in between each of the processing steps, a blur is visible in the final results. Looking at Figure: 5.2 shows which parameters were used for each of the visible result images in Figure: 5.2. Specifically, the number of runs varies only slightly.

5.2.3 Analysis of Specific Examples

Through extensive testing on the impact of the number of runs as well as the blur radius, the best results are achievable using run values between three and four and blur values between two and four. We discuss and analyze the effects of the parameters used later in Section: 5.4. Overall, our approach preserves the look and feel gained from the CRT quite well, despite the slight color discrepancy compared to the original pixel art. This is particularly evident in the eyes of Dracula in Figure: 5.2 where the long stretched eyes are preserved on our upscaled result.

5.2.4 Challenges and Limitations

While our test images achieve good results, this cannot be said for all input images in general. Similarly, not all input parameters work equally well. If images are very dark or have a certain peak color from the CRT mask, for instance, a high red portion, it is difficult for our approach to produce satisfactory results. While it can achieve a decent job, there may be some detail loss and smearing, especially if the input image is comparatively small. To further investigate why our approach struggles with those examples, we examine the reference image (c) from Figure: 5.4 and compare it with our own CRT implementation to indicate why our approach struggles with these images.

As mentioned previously, CRT displays were tuned by the manufacturer, as well as by the home user. When comparing (b) to (d) in Figure: 5.3, it becomes apparent that the brightness and the contrast between pixels are largely increased in the reference CRT image of (b). Similarly, in our CRT implementation, there is no bias towards any of the

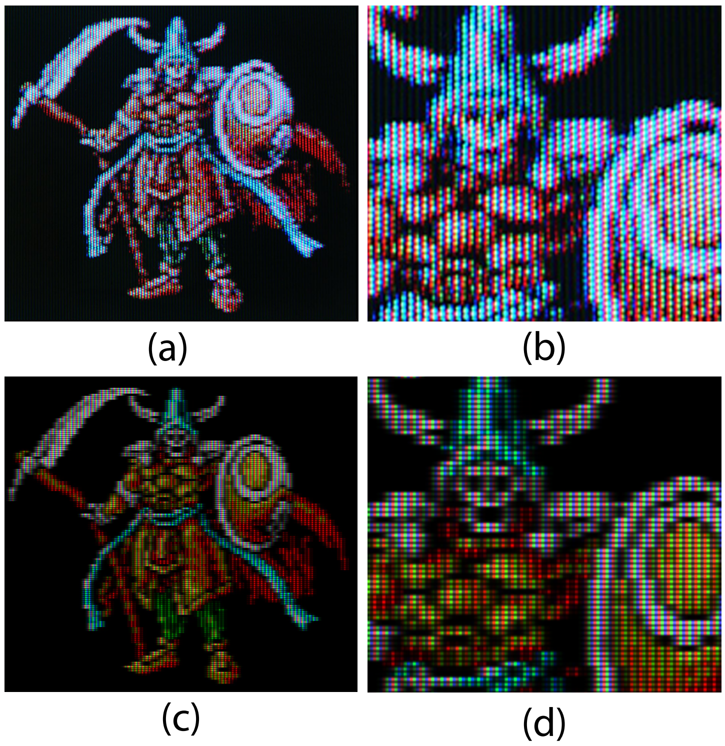


Figure 5.3: This figure demonstrates the difference between reference CRT images and our own software CRT filter.
Image (a) corresponds to the reference image (c) in Figure: 5.4, and image (b) provides a zoomed-in version to clearly show the CRT pattern.
Image (c) depicts our recreation using our custom software CRT filter, while image (d) offers a zoomed-in view for closer examination.

three color channels. Every color is treated with the same amount of brightness for the resulting CRT mask. Compared to the reference image our CRT filter is muted and has less contrast. Because our network was trained with images where the frequency between the red, green, and blue channels is not as high as with the reference image shown in (b), our network struggles to properly retain the colors and shapes. Because there is a bias towards blue in the reference image, the red channel becomes very visible with harsh edges compared to the other two colors, which mix better together. Due to this mixing, red edges become very visible when using such an image with our approach. Analogously, this is the case for image (a) in Figure: 5.4 as well.

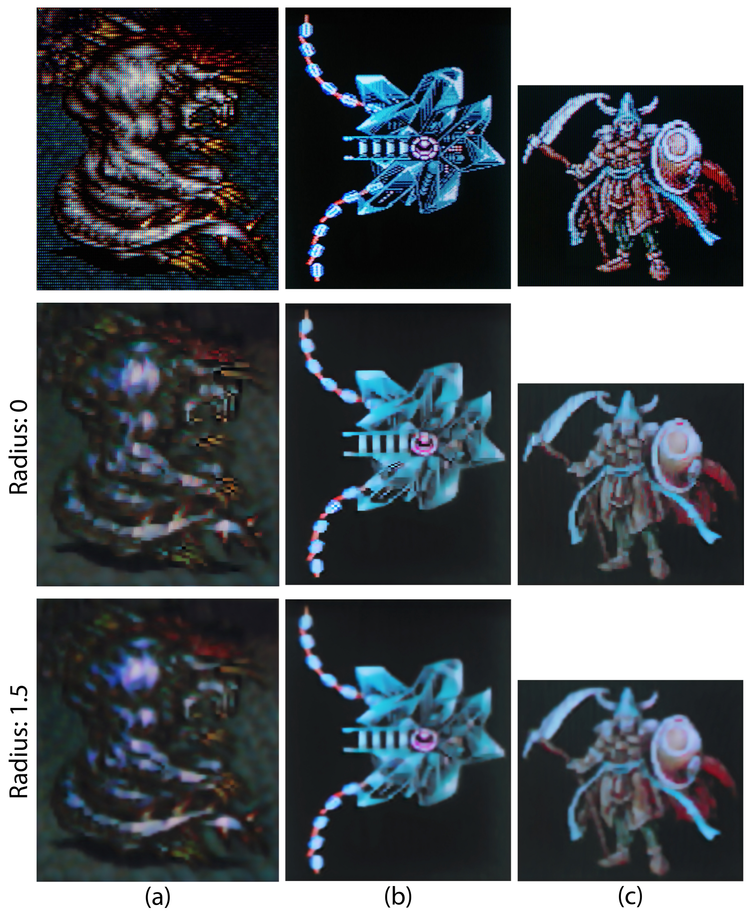


Figure 5.4: Three example images by X user [Ruu21]. The top row is the reference CRT image. The second row is the output of our approach with no blur applied. The bottom row is the output from our approach with a blur radius of 1.5.
It effectively illustrates the challenges our approach encounters with dark or high-contrast images, as well as CRT images biased towards a specific color channel, such as red.

5.3 Memory and Runtime Analysis

We conducted all image tests and benchmarks on a system running Windows 11 and using Python version 3.9, utilizing an AMD Ryzen 9 7950X processor, 128 gigabytes RAM, and an NVidia RTX 2080 graphics card. To maintain consistency in timings and measurements, all processes were executed on the same machine. Each image underwent individual processing in its dedicated process, rather than batch processing, to prevent residual memory allocations from influencing subsequent image analyses. To assess memory consumption and runtime performance, we tracked the average metrics for each image across 25 iterations. The choice of 25 iterations was determined through experimentation to identify a stable plateau in the average metrics. Both processing time and RAM usage exhibit a strong dependence on the number of pixels in the input image, as observed in the graph in Figure: 5.5 and the raw data in Figure: 5.3. As anticipated, both metrics follow a largely linear trend with increasing pixel count.

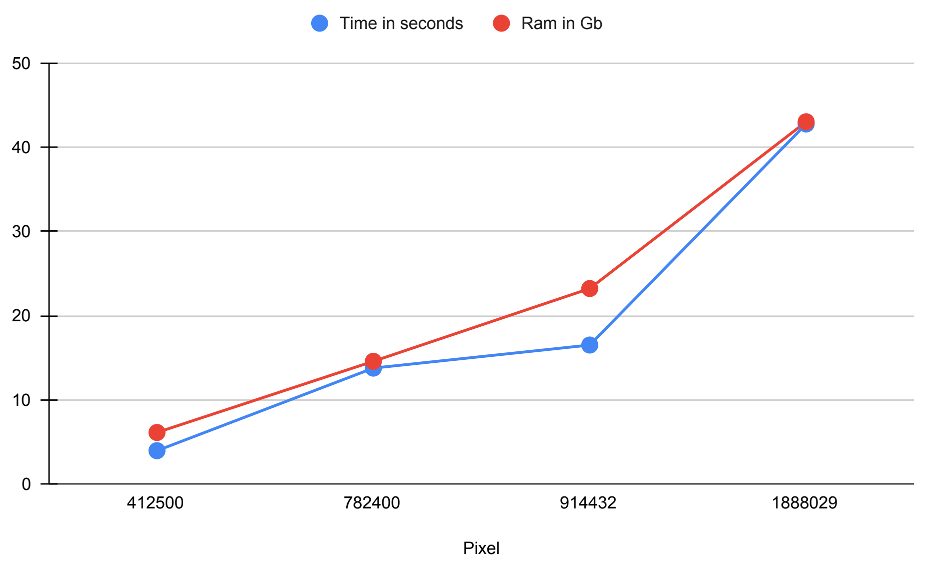


Figure 5.5: This figure illustrates the timings and RAM consumption (Figure: 5.3) relative to the number of pixels, with values sorted in ascending order.

Our approach exhibits inefficiencies in both time and memory compared to conventional upscaling algorithms like bilinear filtering or the hq2x algorithm. The prolonged runtime can be attributed to several factors. Firstly, initializing the neural network entails loading checkpoints and initializing PyTorch, which contributes to the overall processing time. Additionally, neural networks inherently demand significant computational resources, necessitating suitable hardware for efficient operation. Similarly, memory efficiency dimin-

Image	**Avg. Real Time**	**Avg. CPU Time**	**RAM**	**Pixels**
Castlevania (Our CRT)	13.29	11.33	14.5983	782400
Castlevania (Ref. CRT)	13.79	11.73	14.5362	782400
Final Fantasy IV (Our CRT)	3.95	9.83	6.3578	412500
Final Fantasy IV (Ref. CRT)	3.98	9.84	6.3581	412500
Chrono Trigger (Our CRT)	42.79	14.16	43.0527	1888029
Chrono Trigger (Ref. CRT)	16.53	11.32	23.2436	914432

Table 5.3: To display the timings and average RAM usage of our test images, we measured the time units in seconds and RAM units in gigabytes. All images were processed using identical settings, and the number of iterations was set to 25.

ishes linearly with increasing pixel count in the input resolution. As the input resolution rises, our approach's overall memory consumption escalates accordingly. Throughout testing, we encountered challenges related to both RAM and VRAM on our graphics card on images with larger resolutions than our test images, as PyTorch attempted to allocate more memory than available, leading to crashes. Our testing images pushed the boundaries of our hardware capabilities, rendering this approach challenging to execute on standard systems. Optimization is imperative and remains a focal point for future iterations of our approach.

5.4 Parameter Analysis and Their Impact on Output Quality

As previously highlighted, our approach features a relatively small set of input parameters. Among these, the most pivotal parameters include the number of runs and the blur radius applied between successive iterations. Through empirical testing, as discussed in Section 5.2, we determined that an optimal number of runs falls within the range of three to four, while an effective blur radius typically ranges from two to four.

Figure 5.6 vividly illustrates the importance of these parameters in achieving favorable outcomes. Their careful selection significantly influences the quality of the final result.

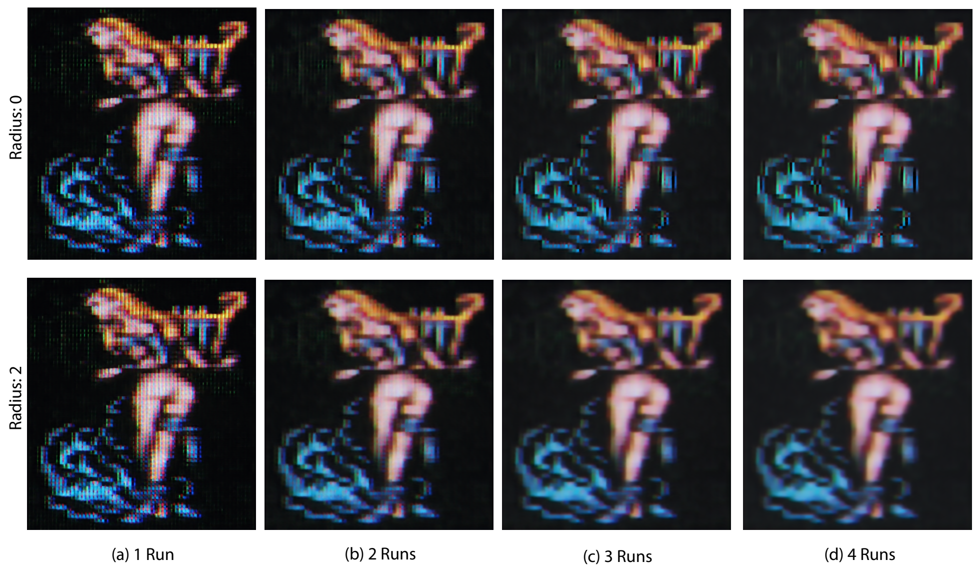


Figure 5.6: Filtering the reference CRT input from Figure: 5.1 (b) with different number of runs as well as different blur radii. The first row shows a radius of zero, whereas the second one has a radius of two.

When the blur radius is relatively small or zero, the refinement process reaches a plateau where further improvement becomes minimal. However, introducing a slight blur radius enhances our model's ability to filter the image and remove unwanted edges and colors

more effectively. As depicted in Figure 5.7, this principle is not absolute.
Image (a) demonstrates the image without any applied blur, where clear red lines are visible at the top, and the overall appearance exhibits roughness with harsh edges. Conversely, applying a small blur radius of 1.5, as shown in the image (b), effectively clears these edges and enhances the overall suitability of the resulting image for our purposes.

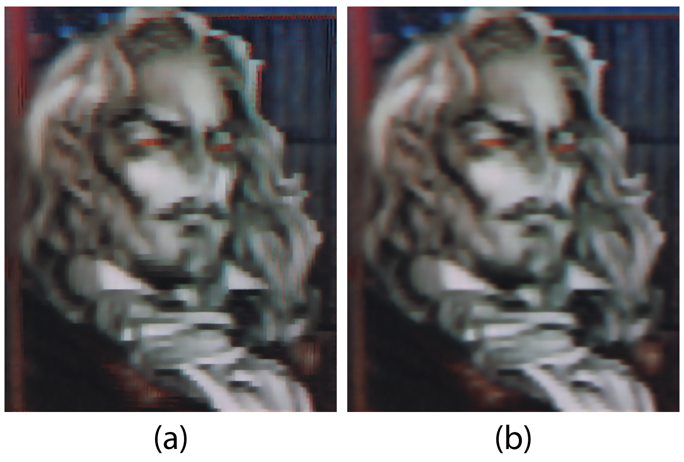


Figure 5.7: Demonstrates the effect a small amount of blur has on the output compared to no blur.
(a) The reference CRT image from Figure: 5.1 upscaled with our approach without using any blur.
(b) Our approach applied on the reference CRT image with a blur radius of 1.5.

The reference CRT images were processed without enabling the first network, where the parameter "–skip" is set. This decision was made due to differences in both color and brightness between the reference CRT images and our own CRT filter used for training the model. The first network tends to over-brighten the output excessively. While it can effectively pre-clean the CRT effect to a significant extent, it exhibits a bias on our dataset. As discussed in Section 5.2, our CRT filter is relatively muted in color and brightness compared to CRT images sourced from the internet.

5.5 Dataset and Training Time Analysis

As outlined and discussed extensively in Sections 4.2.2 and 4.2.1, both the input and target image creation processes were extensive endeavors. The input images comprised a total of 76783 images derived from a ground truth dataset of 10969 images. The CRT filtering process itself was relatively fast, averaging 0.03 seconds per image per CRT mask. Despite its efficiency, processing the entire dataset of input images took approximately 0.63 hours.

In order to create the input images for our second network, we applied our trained network to all 76,783 input images, which took an average of 0.499 seconds per image. This totals to approximately 10,642 hours of processing time. The most time-consuming aspect was the creation of target images. This substantial time consumption is primarily due to the size of our images. Working with both 256x256 and 512x512 images, and with two networks, necessitates target images of 512x512 and 1024x1024 for the first network, and 1024x1024 and 2048x2048 for the second cleanup network. Our dataset consisted of 4235 512x512 images and 6734 256x256 images, respectively. Considering this, we had to upscale and apply the pixelation network to the images twice: once for the first network with a scaling factor of two, and a second time with a scaling factor of four, due to our two-network setup where the first network's output serves as the input for the second network.
Upscaling the 512x512 images to 1024x1024 and 2048x2048 took 1,272 seconds and 5.693 seconds respectively, totaling 8,193 hours. Similarly, the upscaling of the 256x256 images to 512x512 and 1024x1024 took 0,349 seconds and 1,085 seconds respectively, totaling 2,682 hours. Thus, the upscaling of the images alone took an estimated 10,875 hours. Additionally, the pixelation of the 512x512 images took an average of 0,649 seconds, the 1024x1024 images took 2.34 seconds, and the 2048x2048 took 20,997 seconds, amounting to a total pixelation time of roughly 30,266 hours.
In total, creating the target images took an estimated 41,141 hours to produce. Training

Step	**Time in hours**
Training Input (CRT Filtering)	~11,272
Target Input (Upscale + Pixelate)	~41,141
SRResNet Training (Rough CRT)	~10
SRResNet Training (Cleanup)	~10
Total	**72,413**

Table 5.4: Time evaluation of creating the dataset and training the two SRResNet passes. Time was measured in hours.

the networks proved to be more efficient on our hardware compared to the upscaling and filtering steps outlined earlier, as PyTorch is highly optimized for training tasks. As detailed in our implementation in Section 4.3, both networks underwent similar training procedures, comprising a total of 209 epochs each. Each epoch required approximately two to three minutes to process, resulting in roughly ten hours for each network's training cycle. In total, the dataset creation and model training processes amounted to an estimated 72,413 hours. This duration can certainly be optimized by streamlining the workflow, implementing multi-threading where applicable, and utilizing newer hardware for model training.

CHAPTER 6

Conclusion and Future Work

In this thesis, we discussed the challenges of reproducing the distinctive look and feel of pixel art, originally crafted for CRT screens, on modern LCD displays and explored the reasons behind this difficulty. Although the recreation of the CRT effect using software filters remains an area still open to further exploration due to the intricate nature of CRT displays and diverse manufacturing architectures, our research demonstrates the utility of employing these filters for our purpose. Combined with neural networks such as the SRResNet, they offer promising avenues for future development to enhance the pixel art images into their high-definition version. With that said, there are still improvements to be made using different kinds of neural networks such as a Generative Adversarial Network (GAN) that employs adversarial learning that could enhance the results, such as making sharper, less blurry, images.
The training images used in our approach are obtained from two datasets, containing a mix of pixel art and non-pixel art images. Ideally, for the best training outcome, our targets should consist only of high-definition versions of the input images. However, we only had one set of images to create our training and target image pairs, so we experimented with various input and target resolutions and applied pixelization at different stages. We achieved the best training results by upscaling the target images and keeping the input images at their initial resolution. Nevertheless, since this is not an ideal solution, we need to conduct further investigation to refine the training pair in future iterations of our work. Regarding the creation of our input images, all CRT mask parameters were pre-defined and remained similar across all different mask types. One improvement could be calibrating the parameters per image or randomizing them within acceptable parameter bounds and further developing the filters to achieve higher accuracy and a more robust representation of the software CRT result. However, instead of relying on single images on which a CRT filter is then applied, utilizing existing emulation solutions to capture in-game footage and screenshots, or screen capture both with and without a CRT filter applied, could be a valid alternative to generate a comprehensive dataset.

With that in mind, no publicly available CRT dataset currently exists. Generating the dataset using software-based CRT filters also has limitations and shortcomings because of the intricate nature of CRT displays and the multitude of different effects, such as halation which can only be simulated. Creating a dataset based on real CRT displays, instead of using a software-based solution, could be an interesting avenue for future work.

In conclusion, in this work, we presented an approach to addressing the challenge of generating higher-definition versions of CRT images for modern displays. By exploring the feasibility of training and utilising a neural network, specifically the SR network SRResNet, for this purpose, we have demonstrated the potential to bridge the gap between the original pixel art viewed on CRT displays and its high-definition version on modern LCD screens. Our method not only preserves the original look and feel of the CRT representation but also offers insights into the effects of different CRT masks on the training process and final results. Additionally, we have identified several areas for improvement and future research directions to further enhance the efficiency and quality of this approach.

List of Figures

List of Tables

Acronyms

API Application Programming Interface. 9, 23

CRT Cathode Ray Tube. xiii, 1–3, 9, 11, 14, 15, 19, 31, 34, 43–45

GAN Generative Adversarial Network. 5, 6, 43

GLSL OpenGL Shading Language. 23

HR High Resolution. 5

LC Liquid Crystal. 16

LCD Liquid Crystal Display. xiii, 1, 2, 11, 15, 43–45

LR Low Resolution. 5, 26

LUT Look-Up Table. 17, 31

PNG Portable Network Graphics. 32

RAM Random Access Memory. 33

SISR Single Image Super Resolution. 6

SR Super Resolution. 5, 26, 44

VSR Video Super Resolution. 6

Bibliography

[Bli05] James F Blinn. What is a pixel? *IEEE computer graphics and applications*, 25(5):82–87, 2005.

[Che11] Robert H Chen. *Liquid crystal displays: fundamental physics and technology.* John Wiley & Sons, 2011.

[CHQ$^+$22] Honggang Chen, Xiaohai He, Linbo Qing, Yuanyuan Wu, Chao Ren, Ray E Sheriff, and Ce Zhu. Real-world single image super-resolution: A brief review. *Information Fusion*, 79:124–145, 2022.

[CRT21] CRTpixels. Castlevania: Symphony of the night (1997, konami) - ps1 sharp pixels vs. ps1 composite via sony kv-13m51, 2021. [Online at https://twitter.com/CRTpixels/status/1408451743214616587 accessed April 27, 2023].

[Fei] Thomas Feichtmeir. Thomas feichtmeir (@cyangmou) · x. Online at https://twitter.com/cyangmou Mar. 13, 2024.

[Fou] The Linux Foundation. Features. [Online at https://pytorch.org/features/ accessed Jan. 11, 2024].

[GB10] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[gue19] guest.r. Crt - guest - dr.venom, Apr 2019. Online at https://github.com/libretro/glsl-shaders/tree/master/crt/shaders/guest/ accessed Jan. 12, 2024.

[gue22] guest.r. A2080, Nov 2022. Online at https://github.com/guestrr/WinUAE-Shaders/blob/master/CRT-A2080-HiRes-SmartRes-Interlace.fx accessed Mar. 7, 2024.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[Kaw02] Hirohisa Kawamoto. The history of liquid-crystal displays. *Proceedings of the IEEE*, 90(4):460–500, 2002.

[KB17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[KL11] Johannes Kopf and Dani Lischinski. Depixelizing pixel art. In *ACM SIGGRAPH 2011 papers*, pages 1–8. 2011.

[Kon] Konami. Konami group corporation. Online at https://www.konami.com/en/ accessed Jan. 12, 2024.

[Lot14] Timothy Lottes. Fixingpixelartgrille, Aug 2014. Online at https://www.shadertoy.com/view/MsjXzh/ accessed Jan. 12, 2024.

[LTH$^+$16] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew P. Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network. *CoRR*, abs/1609.04802, 2016.

[MC92] Blair MacIntyre and William B Cowan. A practical approach to calculating luminance contrast on a crt. *ACM Transactions on Graphics (TOG)*, 11(4):336–347, 1992.

[MG21] Morgan McGuire and Mara Gagiu. Mmpx style-preserving pixel-art magnification. *Journal of Computer Graphics Techniques Vol*, 10(2), 2021.

[NM92] Avi C Naiman and Walter Makous. Spatial nonlinearities of gray-scale crt pixels. In *Human Vision, Visual Processing, and Digital Display III*, volume 1666, pages 41–56. SPIE, 1992.

[NM14] Kamal Nasrollahi and Thomas Moeslund. Super-resolution: A comprehensive survey. *Machine Vision and Applications*, 25:1423–1468, 08 2014.

[Oak84] K.R. Oakey. Some consequences of the shadow-mask crt dot structure. *Displays*, 5(3):143–148, 1984.

[PETM09] Matan Protter, Michael Elad, Hiroyuki Takeda, and Peyman Milanfar. Generalizing the nonlocal-means to super-resolution reconstruction. *IEEE Transactions on Image Processing*, 18(1):36–51, 2009.

[Phi10] Philipp M. Moore. Crt mask types, ways of measuring dot pitch, 2010. [Online at https://de.m.wikipedia.org/wiki/Datei:CRT_mask_types_ende.svg accessed May 1, 2023].

[pyt] Python. Online at https://www.python.org/ accessed Jan. 12, 2024.

[RBG$^+$17] Amelie Royer, Konstantinos Bousmalis, Stephan Gouws, Fred Bertsch, Inbar Mosseri, Forrester Cole, and Kevin Murphy. XGAN: unsupervised image-to-image translation for many-to-many mappings. *CoRR*, abs/1711.05139, 2017.

[Red22] Reddit_sacks. Mask battle! slot vs dot vs aperture grill, 2022. [Online at https://www.reddit.com/r/crtgaming/comments/vol9yl/mask_battle_slot_vs_dot_vs_aperture_grill accessed May 5, 2023].

[Ruu21] @ruuupu1 Ruuupu. ... pic.twitter.com/q79id6rbgm, Jan 2021. [Online at twitter.com/ruuupu1/status/1348589603763077127 accessed Jan. 11, 2024].

[She03] Steven K Shevell. *The science of color*. Elsevier, 2003.

[Sil15] Daniel Silber. *Pixel art for game developers*. CRC Press, 2015.

[Slu98] AAS Sluyterman. 13.3: A theoretical analysis and empirical evaluation of the effects of crt mask structure on character readability. In *SID Symposium Digest of Technical Papers*, volume 29, pages 179–182. Wiley Online Library, 1998.

[SQU] Square enix limited. Online at https://www.squareenix-games.com/en_GB/home accessed Jan. 12, 2024.

[Tea] Retroarch Team. retroarch. Online at https://www.retroarch.com/index.php accessed Mar. 7, 2024.

[Tea24] Libretro Team, Feb 2024. Online at https://www.libretro.com/ accessed Mar. 7, 2024.

[tor23] torch.nn, 2023. Online at https://pytorch.org/docs/stable/nn.html accessed Feb. 4, 2024.

[Tri86] J Robert Trimmier. Color display technology—an overview. *SAE Transactions*, pages 732–739, 1986.

[TT14] TroggleMonkey and Themaister, Aug 2014. Online at https://github.com/libretro/common-shaders/tree/master/crt/shaders/crt-royale accessed Mar. 7, 2024.

[TT24] Toni and Toni. Winuae, Feb 2024. Online at https://www.winuae.net/accessed Mar. 7, 2024.

[UVD88] Frank B Uphoff, NAVAL AIR DEVELOPMENT CENTER WARMINSTER PA AIR VEHICLE, and CREW SYSTEMS TECHNOLOGY DEPT. Cathode ray tube displays. Technical report, 1988.

[WC92] T. R. H. Wheeler and M. G. Clark. *CRT Technology*, pages 221–256. Springer US, Boston, MA, 1992.

[WCZ$^+$22] Zongwei Wu, Liangyu Chai, Nanxuan Zhao, Bailin Deng, Yongtuo Liu, Qiang Wen, Junle Wang, and Shengfeng He. Make your own sprites: Aliasing-aware and cell-controllable pixelization. *ACM Trans. Graph.*, 41(6), nov 2022.

[YOM68] Susumu Yoshida, Akio Ohkoshi, and Senri Miyaoka. The" trinitron"-a new color tube. *IEEE Transactions on Broadcast and Television Receivers*, 14(2):19–27, 1968.

[ZP13] C. Lawrence Zitnick and Devi Parikh. Bringing semantics into focus using visual abstraction. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3009–3016, 2013.