Winter 12-2023

An Investigation of Match for Lossless Video Compression

Brittany Sullivan-Reicks
University of Nebraska-Lincoln, sullivan.brittany@huskers.unl.edu

AN INVESTIGATION OF MATCH FOR LOSSLESS VIDEO COMPRESSION

by

Brittany Sullivan-Reicks

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Master of Science

Major: Electrical Engineering

Under the Supervision of Professor Khalid Sayood

Lincoln, Nebraska

December, 2023

AN INVESTIGATION OF MATCH FOR LOSSLESS VIDEO COMPRESSION

Brittany Sullivan-Reicks, M.S.

University of Nebraska, 2023

Adviser: Khalid Sayood

A new lossless video compression technique, Match, is investigated. Match uses the similarity between the frames of a video or the slices of medical images to find a prediction for the current pixel. A portion of the previous frame is searched to find a matching context, which is the pixels surrounding the current pixel, within some distance centered on the current location. The best distance to use for each dataset is found experimentally. The matching context refers to the neighborhood of w, nw, n, and ne, where the pixel in the previous frame with the closest matching context becomes the prediction. w, nw, n, and ne stand for west, northwest, north, and north-east respectively. Using these directions, w is the pixel to the left of the current one, nw is the pixel to the left and up one row, n is the pixel directly above the current one and ne refers to the pixel up one row and to the right one column. From the prediction, the error is then calculated, remapped and encoded using adaptive arithmetic encoding. Match's resulting compression ratio is then compared to that of CALIC's, where the larger the compression ratio the more efficient the method. CALIC is a context-bases adaptive lossless image compression technique that is regarded as one of the best lossless image compression techniques. Match was evaluated for twenty-two video datasets of varying resolutions as well as 65 C.T. scans and 17 M.R.I. scans. Some common differences amongst videos are resolution and frame rate. Therefore, Match was used to compress four videos with varying resolution to see how Match is affected by resolution and Match was examined on one dataset that had varying

frame rate. There were times when Match outperformed CALIC; however, there were also times where CALIC outperformed Match and other times where the two methods resulted in nearly identical compression ratios. Therefore, as a preprocessing step, the structural similarity was examined as well as the edge quality measurements to predict which method, Match or CALIC, results in the best compression.

DEDICATION

To the love of my life, Andrew Ray Reicks. And to my incredibly supportive family who I could not have done this without.

ACKNOWLEDGMENTS

I would like to offer special thanks to the following individuals:

My advisor, Dr Khalid Sayood for all of their support, guidance, and understanding.

Miss Roxanne and Shadow for all of their love and support as I could not have remained sane without them.

My father, Timothy Sullivan for all of his love, support and guidance. Without him I wouldn't have been able to debug my code.

My mother, Kristi Sullivan for all of her love and support.

My committee members, Drs. Michael Hoffman and Dr. Benjamin Riggan

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

A camera is an optical instrument used to capture and store images, either digitally or chemically. Cameras have a lens, shutter, and a focal plane array that captures visual information and are sensitive to one, or more, ranges of wavelengths of light. As technology advances, cameras have evolved to produce better images. Film cameras have become a thing of the past as digital cameras have taken over. Today, most everyone has access to a decent camera, specifically through their smart phones. Cameras have evolved to where anyone can take a photo or video, edit the resulting images, and share those images or videos with the rest of the world in minutes. The resulting photos have evolved, specifically in resolution. I remember my first digital camera, I thought this camera was the coolest thing and the camera produced higher quality images than a film camera. Not only did this digital camera result in higher resolution, but you can see your images right away instead of waiting and hoping that you captured what you wanted as your film develops. You can also store many more photos on an S.D. card than the limited amount of film old cameras use. Cameras have become so common, that these days, no one can imagine life without photography. But how did we get here? Where do cameras come from? How did the world of photography start?

The camera was created based on the principle of camera obscura. Obscura is Latin for “dark room”. Therefore, camera obscura is an optical device that creates an image by focusing rays of light onto a screen or sheet of paper [1]. More than 2,000 years before the invention of camera obscura, Aristotle discovered that by passing sunlight through a pinhole, he could create a reversed image of the sun on the ground. He then proceeded to use this discovery to view eclipses without having to stare directly into the sun [2]. Before photo paper was a thing, artists would use this concept to cast the image on a wall and trace the reflection. Using this concept, I once made a pin hole camera out of a shoe box. To do this, you paint the inside of the box black, and make sure that there's no light coming in other than from a little hole you create. From there, you tape photo paper to the inside and let light shine through the pin hole for a discrete amount of time. The photo paper then needs to be developed. The resulting photograph is nothing like the pictures that are taken today. The idea of camera obscura using a pinhole camera is illustrated in Figure 1.1 [3]. The earliest known written record of a pinhole camera is found in the Chinese text called Moxi which is from the 4^{th} century BC. The concept of camera obscura has been known for millennia; even Aristotle used a pinhole camera to observe solar eclipses. During the 18^{th} century, this technique led to the creation of portable "camera boxes" [4].

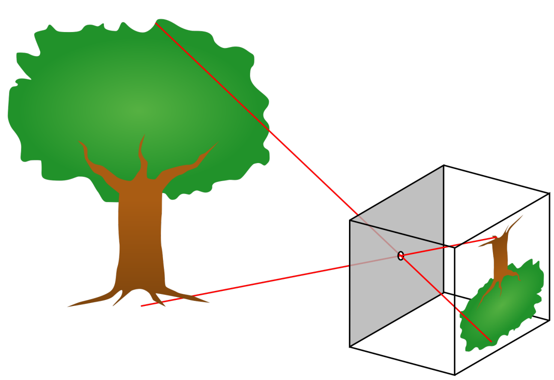

Figure 1.1: Camera Obscura Pinhole Camera Example [3]

In 1839, Louis Daguerre invented the daguerreotype which is an early form of a photo camera and was the first mass marketed camera. The daguerrotype is a direct-positive process that creates a highly detailed image on a sheet of a copper plate that is coated with a thin layer of silver without the use of a negative. This process required a clean and polished silver-plated copper plate so that it looked like a mirror. From there, the plate was sensitized in a closed box over iodine until the plate would take on a yellow-rose appearance. The plate, held in a lightproof holder, would then be transferred to the camera. Using the concept of camera obscura, the plate is then exposed to light. After the exposure to light, the plate was then developed over hot mercury until an image appeared. To fix the image, the plate was immersed in a solution of sodium thiosulfate or salt and then toned with gold chloride [5]. A drawing of a daguerrotype camera is illustrated in Figure 1.2 [4]. This camera had an exposure time of five to thirty minutes.

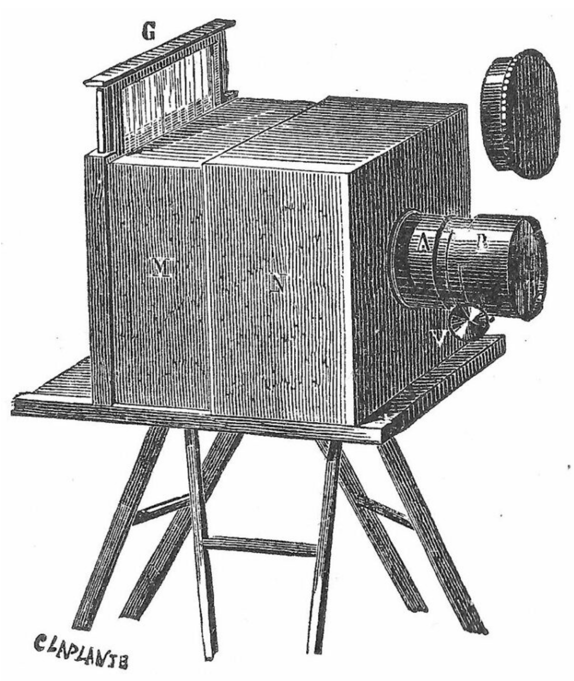

Fig. 327.

Figure 1.2: Daguerreotype Camera [4]

In 1850, the daguerrotype was replaced by a new “colloid process” which required treating the plates before using them. This process resulted in a shorter exposure time and produced sharper images. The exposure time was so little that the shutter was invented so that the plate would only be exposed to light for a short period of time[4].

The first roll film camera, the Kodak, was invented in 1888 by George Eastman. This camera, unlike the daguerrotype, would capture the negative picture. The film needed to remain in a dark box camera before being sent off to Eastman's company for the film to be processed into pictures. The first kodak camera could hold a roll of film up to 100 pictures [4]. Figure 1.3 illustrates the first Kodak film camera released in 1888 and the Kodak 35 which was introduced in 1938, which used 35mm film.

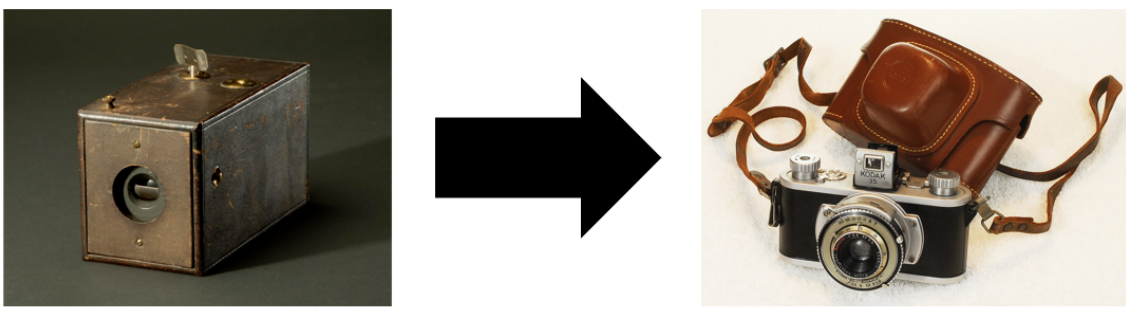

Figure 1.3: Original Kodak Film Camera (left)[6] and the Kodak 35 (right)[7]

As cameras developed, they became more affordable and therefore more popular. People were now able to capture moments in time, but what about capturing more than just a second? The first movie camera was invented by French inventor Etienne-Jules Marey in 1882. This video camera was known as the "chronophotographic gun", illustrated in Figures 1.4 and 1.5. It took 12 images a second and exposed them on a single curved plate. At it's most superficial level, a video camera is simply a photographic camera that can take repeated images at a high rate, where each photo is referred to as a frame. In 1893, William Dickson invented the most famous early movie camera, the "kinetograph", illustrated in Figure 1.6. This video camera was powered by an electric motor, used celluloid film and ran at 20-40 frames per second. This invention signaled the beginning of cinematography [4].

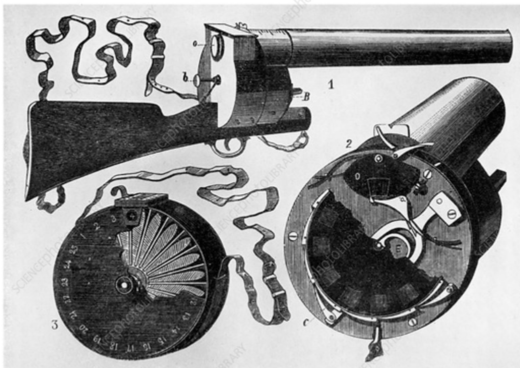

Figure 1.4: Chronophotographic Gun[8]

Figure 1.5: Chronophotographic Gun[9]

Figure 1.6: The Kinetograph[10]

Thomas Sutton developed the first camera to use single-lens reflex (SLR) technology in 1861. Knowing that the reflected image from the concept of camera obscura, the SLR camera used mirrors to reflect the image so that the user would see the exact image recorded on film when looking through the camera's lens. The SLR camera became the camera of choice in the mid 1900's as new technology allowed the reflective mirror to "flip up" when the shutter opened. This means the resulting image through the viewfinder was perfectly like that captured on film [4].

Digital photography may have been theorized in 1961 but it wasn't created until 1975 by Steven Sasson. His creation weighed four kilograms and captured black and white images onto a cassette tape. This camera couldn't have been invented without the "charged-coupled device" (CCD), which was developed in 1969 by Willard S. Boyle and George E. Smith. This device used electrodes that would change voltage when exposed to light. Sasson's camera had a resolution of 0.01 megapixels which is [100 x 100], and took 23 seconds of exposure to record an image. Note that in the Graphic's industry, the standard is to present width by height.

The first commercially available digital camera, the Dycam Model 1, also known as the Fotoman, became available in 1990 and cost $995. This digital camera is illustrated in Figure 1.7. Created by Logitech, it used a similar CCD to Sasson's original design, however it recorded the data onto the internal memory, which came in the form of one megabyte of RAM [4]. Today, the capabilities of this camera are not very impressive; this camera could only take black and white photographs at a measly resolution of [367 x 240] pixels. This camera also only had 1MB of internal storage, which can only handle approximately 32 photographs. To then see these photos, you had to plug the camera into your computer where you would then use Logitech's software to view and edit the images. When using this camera, you have to keep an eye on how charged the batteries are because if they died, then the images stored in the camera would be lost. Not only would you lose all the photos you took, but you would have to reinstall it's operating software from scratch. Despite it's many drawbacks, this camera was just the beginning. [11]

Figure 1.7: The Fotoman[11]

Three years later, Logitech released the Fotoman Plus, which was $200 cheaper and has an increased resolution of [496 x 360]. The resulting photographs were still in black and white, but now the images were in the format finalized by the Joint Photographic Experts Group (JPEG) in 1992. Everything changed after this. JPEG has become a standard image format that is compatible across all our different digital architectures. JPEG made the images usable in a wider variety of contexts as the images were no longer in a format that can only be read by Logitech's software [11].

This led to the first digital single-lens reflex (DLSR) camera, the Kodak DCS-100, which was also released in 1991. This camera had a built-in 1.3-megapixel Kodak CCD to capture images, which results in a resolution of [1288 x 1024]. This camera required an external data storage unit that was connected via a cable. This made photographers wear a shoulder strap to carry around the storage unit [12].

A few years later, in 1994, Kodak came out with the Kodak AP NC2000. This camera was designed specifically for photojournalists. This camera was also a 1.3 mega-pixel camera, however, it had a removable memory card. This camera also has an ISO up to 1600[12]. ISO stands for the International Organization for Standardization and is a parameter that determines how bright your image will be, which gives photographers an extra setting to manipulate their exposures.

ISO gets its name from the international Standards Organization, which set this standard in 1988. ISO doesn't change the amount of light coming into the camera like shutter speed and aperture. It does, however, determine how the camera deals with the light the sensor receives. The larger the ISO the more sensitive the sensor is [13].

Other companies other than Kodak also began to create digital cameras. Not only was the Kodak AP NC200 created in 1994, but Apple put out the Apple Quick Take 100. This was Apple's first digital camera and was built in collaboration with Kodak. Apple emphasized the ease of use for this camera. This camera has a 307-kilopixel sensor which produced images at a resolution of [640 x 480] and produced color images. However, storage was an issue. The camera conveniently features internal memory, but there's no way to expand it. It can hold 32 "standard" ([320 x 240]) images or a mere 8 images of its 307-kilopixel pictures [14]. This camera also had a fixed 50mm equivalent F2 lens, an optical viewfinder, and an LCD display to view the settings [13].

In 1995, the Casio QV-10 was released; which was the first consumer digital camera that included a built-in LCD screen. This camera has the same resolution as the "standard" images of the Kodak AP NC200 [16].

Not only was the Casio QV-10 released in 1995, but Ricoh released the RDC-i700 image capturing device. This is the first camera to combine still image digital captures with video and audio recording. This camera was designed to look and operate as much like a hand-held computer as a point-and-shoot digital camera. This camera was really more of a portable image-capture and manipulation computer system than a typical camera. Accommodating these functions, however, brings bulk, cost, and complexity beyond what's expected of a digital camera. The RDC-i700 provides 0.41-megapixel and $\frac{1}{3}$" CCD, which is a resolution of [768 x 576]. It not only has 8MB of internal memory, but a removable storage option as well. Some special features of this camera include motion picture recording, continuous shooting mode, interval recording, timed exposures up to 8 seconds, macro lens adjustment, white balance adjustment, exposure compensation, a self-timer option for delayed shutter release, and on-screen image editing and annotation [15].

Two major cameras were released in 1996, the Nikon Coolpix 100 and the Kodak DC25. Their resolutions are 0.24 mega-pixel ([512 x 480]) and 0.18 mega-pixel ([493 x 373]) respectively. The Nikon Coolpix 100 plugged into a laptop's PC card slot to transfer pictures, while the Kodak DC25 was the first digital camera to incorporate Compact Flash media for storage. Compact Flash is a flash memory mass storage device used in portable electronic devices [16].

A year later, in 1997 Sony released the Sony Digital Mavica FD5, which is the first digital camera to write to a 3.5-inch floppy disk for photo storage. This camera has a resolution of 0.3 mega-pixel, or [640 x 480].

Three major cameras were released in 1999, the first being the WWF Slam Cam which was the first digital camera on the market aimed towards kids. It could only store six photos of a resolution of 0.02 mega-pixel or [160 x 120]. The second camera that was released in 1999 was the Nikon D1, the first fully integrated digital SLR camera. This camera is notable for using lenses from it's equivalent film camera, the Kodak 35 which used 35mm film. This camera had incredible resolution for this time of 2.62 mega-pixels which is [2000 x 1312] [16]. As cameras were improving and becoming more common, cell phones were also becoming a common item. These days, nearly everyone has a cell phone with high resolution cameras. The third major camera released in 1999 is the first camera phone released by Kyocera, the Kyocera Visual Phone VP-210. This phone had a 0.11-megapixel front camera and could store 20 still photos and transmit live “video” at a rate of 2 frames per second [13].

The next year, in 2000, the Fujifilm FinePix S1 Pro was released. It’s the first camera with interchangeable-lens DSLR. It has a super CCD sensor that resulted in output images of 6.13-megapixels ([3040 x 2016]) and enabled sensitivity setting up to ISO 1600 [13]. Another camera released in this year was the Canon S100, which pushed digital pocket cameras toward smaller sizes and higher resolutions. This one has a 1.92 mega-pixel resolution, which is [1600 x 1200]. Canon also released the EOS D30, which is Canon’s first digital SLR camera with a resolution of 3.11 mega-pixels ([2160 x 1440]) [16].

2002 saw three revolutionary digital cameras, the Casio Exilim EX-S1, the Contax N Digital, and Canon’s EOS-1Ds. The Casio Exilim EX-S1 continued Canon’s trend of small cameras, however this camera was tiny with dimensions of 55 x 88 x 11.3mm. This camera was about the same height and width of a credit card (53.98 x 85.6mm). This tiny camera has a resolution of 1.22 mega-pixels which generates a photo that’s [1280 x 960] pixels. The Contax N Digital camera, on the other hand, was the first camera to include a CCD sensor the size of a full 35mm frame. Its resolution is 6.1 mega-pixels which creates an image size of [3040 x 2008] pixels. Unlike the other two, Canon’s EOS-1Ds was Canon’s first full-frame camera with a 10.99 mega-pixel resolution ([4064 x 2704]) [16].

In 2003, Canon’s digital camera evolved to the EOS Digital Rebel D300, which was the first SLR camera under $1,000. It had a resolution of 6.29 mega-pixels or [3072 x 2048] pixels. This same year, the Olympus E-1 was also released, it was the first camera to use the $\frac{4}{3}$ SLR system and has a resolution of 4.91 mega-pixel ([2560 x 1920]) [16].

The Epson R-D1 was released in 2004 and was the first digital rangefinder camera with a resolution of 6.01 mega-pixel ([3008 x 2000]) [16].

In 2007, Steve Jobs released the first iPhone. Phone memories have gotten larger so more pictures could be stored on them. The first iPhone had a 3.5-inch screen, a 2 mega-pixel ([1600x1200]) camera and 16GB of storage. The CCD sensors were replaced by CMOS chips that use less power. The internet, 3G, 4G, and 5G made it possible to share photos instantly, whether through text, email, or social media. Even though the iPhone isn't the first phone camera, the iPhone is the most popular camera in the market [13].

Nikon released the Nikon D3X which was Nikon's top-of-the-line DSLR camera that targeted professional photographers in 2008. It has an outstanding resolution of 24.38 mega-pixels resulting in images of [6048 x 4032] pixels [16].

In 2010, Sony released the Sony Cyber-DSC-TX7, which is a full-featured pocket point-and-shoot camera with intelligent panorama features that has a resolution of 9.98 mega-pixels ([3648 x 2736]). This same year the Pentax 645D was released. It was the first medium-format DSLR camera that was sold for under $10,000. This was the opening for the high-end super high resolution photography to be more accessible to the average person. This camera has an incredible resolution of 39.51 mega-pixels ([7264 x 5440]) [16].

Digital cameras have continued to evolve, to the point where the world's biggest digital camera is coming into focus. While a powerful, personal camera has megapixel resolution, astronomers have constructed a device to image the distant universe with a 3.2 giga-pixel resolution, this camera is illustrated in Figure 1.8. This resolution is nearly 100 times that of the Pentax 645D. This camera is to be the workhorse for the

Vera C. Rubin Observatory's telescope, which has been in the works for nearly two decades. Aaron Roodman, who is an astrophysicist working on this camera says, "In the combination of the camera's giant focal plane and 25-foot mirror to collect light, we are unparalleled." This camera is in the *Guinness Book of World Records* for the extraordinary sizes of its 5.5 foot lenses, their lens caps, and the focal plane.[17].

This camera is absolutely revolutionary, Ramin Skibba states in his Wired article, "The camera will image each piece of the sky every three days, providing snapshots that can be used together to examine faint or distant objects, or spot changing ones, such as supernova explosions and the paths of near-Earth asteroids and comets slowly moving in their orbits. "It's making a 10-year color movie," says Risa Wechsler, a Stanford University astrophysicist and member of the Rubin Observatory scientific advisory committee. "And in addition, it's stacking the frames of that movie to get a really deep image. That will give us a map of all of the galaxies, which traces where all of the matter is, which is mostly dark matter. We'll see what the universe looked like billions of years ago and learn more about what dark matter is."[17].

Figure 1.8: World's Biggest Digital Camera[17]

Cameras have evolved greatly over the years, from film to digital, from [100 x 100] pixels to [6048 x 4032] pixels. Not only that, but cameras have become incredibly affordable with great resolution. A Nikon COOLPIX B600 along with a memory card, memory card wallet, deluxe soft bag, 12 inch flexible tripod, deluxe cleaning set, and USB card reader costs only $439.99 on amazon [19]. The resulting images from this camera are [4608 x 3456] pixels. A few years ago, I bought myself a Nikon D3400 which has a resolution of 24.2 mega-pixels and produces images that are [6000 x 4000] pixels. These days, you don't even need a digital camera to capture incredibly detailed images, nearly every phone has at least one camera. Smartphones today incorporate three cameras to provide better images. These cameras are as follows: the long-focus camera, which helps to magnify distant objects and include them in the resulting photo, the color camera which assists in capturing color information of the objects you are shooting, and the monochrome camera which plays an important role in capturing details.

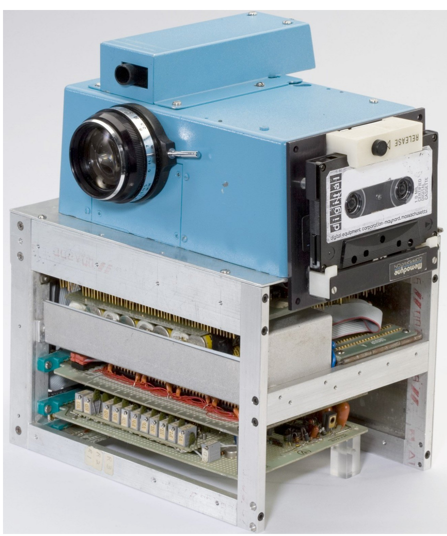

Figure 1.9: Kodak's First Digital Camera invented by Stephen Sasson 1975[20]

Figure 1.10: Nikon D6 Camera 2023[21]

Digital cameras can routinely exceed 200 fps when recording video while most smart phones can record video up to 60fps. However, the fastest camera out there records video at 70 trillion frames per second [18].

The technological evolution of cameras has lead to a common issue, storage space. When cameras first became common, they used film, which allowed for a finite amount of images that you had to hope and pray turned out once the film was developed. Digital cameras, on the other hand, are stored onto an SD memory card. Initially, the first commercially available digital camera could only hold 32 photographs. Today, SD cards can go up to one terabyte of data, which can hold 250,000 photos taken with a 12 mega-pixel ([4000 x 3000]) camera. However, storage space is still limited.

Not only is storage space a problem, but bandwidth is limited; where bandwidth refers to the volume of information per unit of time that a transmission medium can handle - specifically with uploading and downloading files. In simpler terms, the larger the information file is, the more time it will take to upload and download. Therefore, video compression is used to reduce the size of the file to clear up storage space and ease upload and download times.

There are two kinds of compression, lossy and lossless. In lossy compression, data is lost to increase the amount of compression. Lossy compression is great for movies as a lot of information can be lost without visually affecting the quality of the image. However, in certain cases, no information can be lost, such as when looking at a series of medical images. Lossless compression results in less compression than lossy but maintains the high quality of the image. In this thesis, we will be focusing on lossless compression due to the incorporation of medical images in our datasets.

A new non-linear prediction method that depends on the previous frame of a video to predict the next pixel in the current frame is explored. The first time Match was introduced, it was introduced as a prediction method that used conditional averages known as Conditional Average Prediction (CAP) [33]. Unlike other predictive schemes, this one does not depend on the assumption that neighboring pixels tend to be alike. The neighboring pixels are examined to find a matching neighborhood in the previous frame in order to find the best predictor for the current pixel. This new method implements adaptive arithmetic coding and uses CALIC to predict the first image which are first discussed.

The rest of the thesis is organized as follows: first, other lossless video and medical image compression algorithms are discussed before we delve into the necessary background information. The method we describe in this thesis uses adaptive arithmetic coding to encode the prediction residuals generated by the method. It also uses measures of structural similarity and edge quality measurements to select the encoding mode. Adaptive arithmetic coding, CALIC, structural similarity, and edge quality measurements are all discussed in chapter 3. In chapter 4, the method of the new algorithm, Match, is explained. The resulting compression ratio of Match is then examined and compared to the compression ratio of CALIC. This is done for 22 videos, 65 C.T. scans and 17 M.R.I. scans. The performance of the algorithm for different resolutions and frame rates are also examined and discussed. Lastly, we show how we can use the structural similarity and edge quality measurements to determine which method, Match or CALIC, should be used to compress the videos.

Chapter 2: Related Work

Before we delve into the proposed non-linear prediction method, it's important that we first see what others have done in not only the world of lossless video compression, but lossless medical image compression as well. Both videos and medical scans are a series of images, or frames, that make the datasets 3-dimensional. Lossless image compression algorithms can be divided into two categories, transform and prediction-based coding. In transform coding, a reversible transformation to the image is applied. Prediction-based coding, on the other hand, predicts the pixels of the image using spatial correlation and the residual image [22]. When dealing with three-dimensional images, many prediction methods have been previously explored.

Li developed a lossless video sequence compression that utilizes adaptive prediction [23]. It exploits the spectral, spatial, and temporal redundancies. This method selects the best predictor out of a set of predictors without using any side information. It employs a backward pixel-based temporal predictor without using motion vectors.

Li's predictive method follows the block diagram in Figure 2.1. To start, the source frames need to be preprocessed, where the first step estimates the amount of temporal redundancy by the interframe correlation coefficients of the test video sequence. If the average of these coefficients is smaller than a predefined threshold of 0.9, then the video sequence is likely to be high motion. If this is the case, then motion compensation in the wavelet domain is inefficient; and thus the sequence is

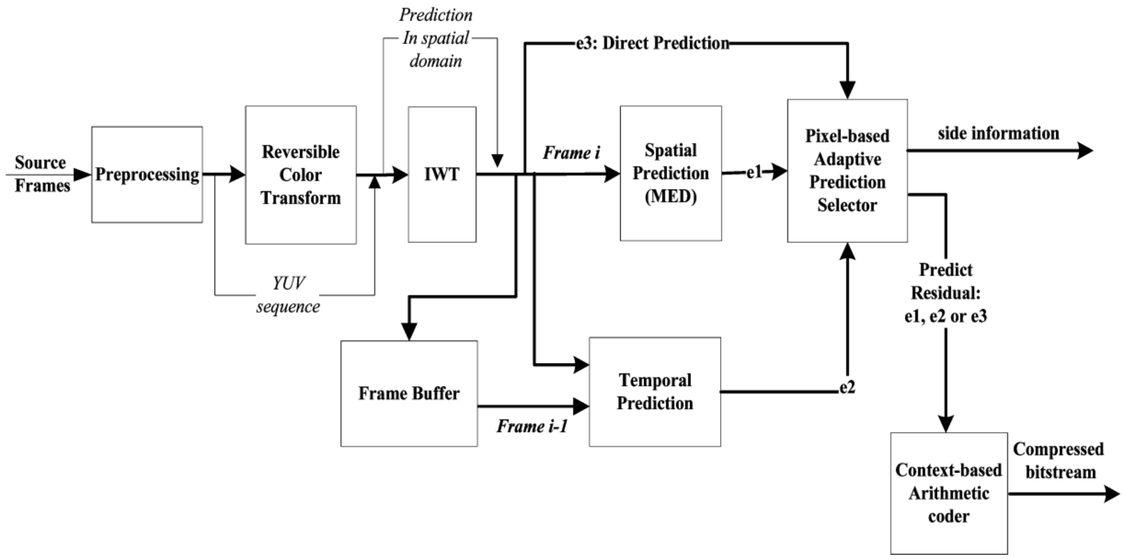

Figure 2.1: Block Diagram of Ying Li's Proposed Algorithm[23]

operated in the spacial domain. Once the preprocessing is complete, a reversible color transform is implemented before the suitable integer wavelet transform (IWT) is determined for the test sequence by estimating its spatial redundancy. To then reduce the spacial redundancy, a prediction is computed based on the neighboring symbols in the same frame as the symbol to be encoded. In this scheme, the predictive method is the median edge detector (MED) used in JPEG-LS. Not only is there a prediction based on the neighboring symbols, but a temporal prediction is also found. This is an adaptive pixel-based predictor based on the symbols in the reference frame with improvement to reduce the temporal redundancy. This predictor aims to find the best matched symbol in a window of the reference frame. Due to the energy compaction property of the IWT, the wavelet coefficients in the high frequency subbands usually have small amplitudes, which may be smaller than the amplitudes of the spatial prediction residuals and temporal prediction residuals. Therefore, if this is the case, then the wavelet coefficients are encoded and transmitted directly. There are now three possible predictions, the spacial prediction or MED, the temporal prediction, and the direct mode prediction. To decide which prediction is the best one, a backward adaptive prediction mode selector is implemented to adaptively select the predictor among the three candidates based on previous prediction accuracy. Finally, context modeling is used for efficient coding of the prediction residuals. By utilizing suitable context models, the given prediction residuals are then encoded by switching between different probability models according to already encoded neighboring symbols of the symbol to be encoded [23].

A contex-based predictive coder for lossless and near-lossless compression of video was developed by Yang and Frayer [24]. They implement interframe and intraframe coding modes. The coding mode is adaptively chosen for the pixel by comparing the temporal and spatial variations.The intraframe coding utilizes the JPEG-LS standard where the JPEG-LS coder is a context-based predictive method operating in two coding modes, the run mode and the regular mode. The interframe coder also operates in two sub-modes, the temporal run mode and the temporal prediction mode which are conceptually very similar respectively to the run and regular modes. In the temporal prediction mode, the temporal prediction is performed and then the prediction error is corrected by a context dependent bias. In the temporal run mode, the encoder looks for a sequence of consecutive samples each of which has a value near identical to the value of the corresponding reconstructed sample in the reference frame. To determine which mode should be selected the scheme is illustrated in listing 2.1. Here, Ra, Rb, Rc, and Rd are the reconstructed values of the neighboring samples of the current frame where Ra is the value of sample to the left of the current pixel, Rb is the value of the upper sample, Rc is the value of the upper-left sample, and Rd is the value of the upper-right sample. Ra', Rb', Rc', and Rd' are the reconstructed values in the reference frame and respectively in the same locations as Ra, Rb, Rc, and Rd. Vt is the temporal variation around the pixel, Vs is the spacial variation around the pixel and a is set to 0.5.

```
if(|Rd − Rb| <= Near && |Rb − Rc| <= Near && |Rc − Ra| <= Near){
        run mode
}
else if (|Ra − Ra'|␣<=␣Near␣&&␣|Rb␣−␣Rb'| <= Near && |Rc − Rc'|␣<=␣Near␣&&␣|Rd␣−␣Rd'| <= Near){
        temporal run mode
}
else if (Vt < a * Vs){
        temporal prediction mode
}
else{
        regular mode
}
```

Listing 2.1: Prediction Scheme [24]

A method specific for high definition, HD, video coding using significant bit truncation was developed by Kim and Kyung to compress videos in real time [25]. Their algorithm consists of two steps. The first step being a hierarchical prediction method that is based on pixel averaging and copying. It uses as many average predictions in a block as possible, which then gives a more accurate prediction. The second step then involves significant bit truncation (SBT) which encodes the prediction errors without any data dependency so that multiple prediction errors in a group are decoded in a clock cycle.

The proposed Hierarchical Average and Copy Prediction (HACP) is represented in Figure 2.2 for an 8x8 block where the level number L represents the distance by 2^{L-1} between the original pixel and the source pixel for predicting the pixel value. The pixels on level 1 and level 2 are predicted by the pixels of the one-pixel and two-pixel distance respectively. In level 3, each of the four pixels is predicted by their average value. The arrow tail represents the source pixel, which is then used to predict the destination pixel indicated by the arrow head. There are four types of prediction methods in HACP: horizontal average prediction (HAP), vertical average prediction (VAP), horizontal copy prediction (HCP), and vertical copy prediction (VCP). If a pixel is pointed to by more than one arrow, the pixel is then predicted by the average value of those pixels. However, if only one arrow points to the pixel, then the prediction is made by copying the neighboring pixel.

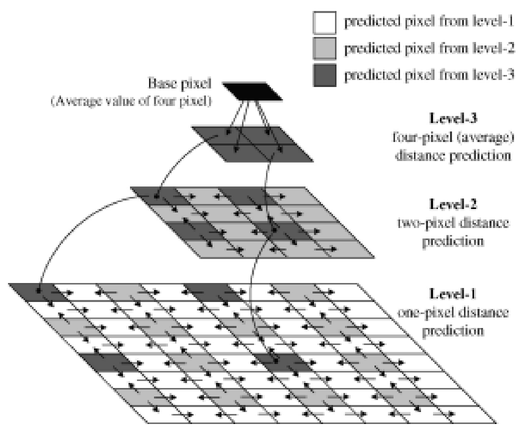

Figure 2.2: Proposed HACP Scheme for 8x8 Block[25]

Kim and Kyung's second step is significant bit truncation (SBT), which is a novel coding method. The basic concept of SBT coding is to express all of the prediction errors in a group with as many significant bits truncated as the information of each prediction error is preserved [25].

Choi and Ho considered the statistical differences of the residual data between lossy and lossless coding [26]. They modified the context-based adaptive binary arithmetic coding (CABAC) mechanism to convert from lossy to lossless coding. They do this by analyzing the statistical characteristics of lossless coding and designing an efficient level binarization method, which leads to a binarization table selection method that uses the weighted sum of the previously encoded results. This method not only increases the compression ratio, but it also reduces decoding complexity. The algorithm for this prediction method is illustrated in the block diagram in Figure 2.3.

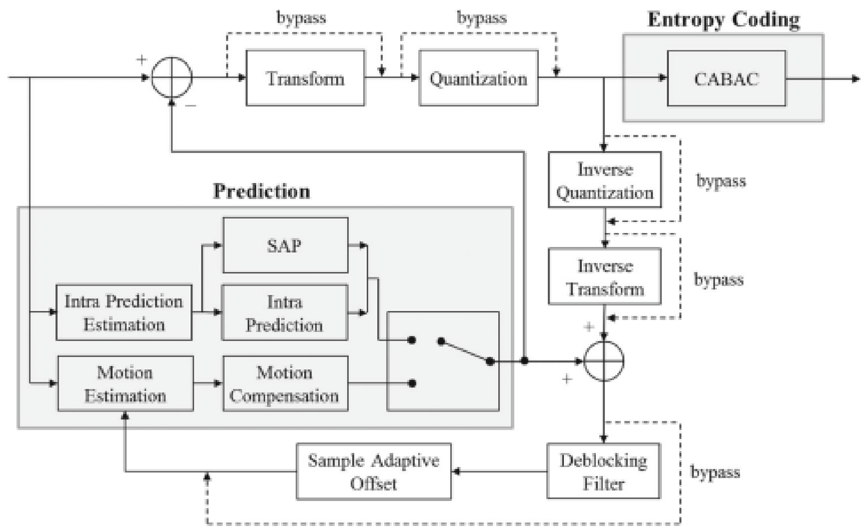

Figure 2.3: The HEVC Lossless Coding Block Diagram[26]

They propose an efficient residual data coding method for HEVC lossless video compression by using sample-based angular prediction (SAP), modified level binarization, and binarization table selection with the weighted sum of previously encoded level values to improve the HEVC. SAP is used to explore the spacial sample redundancy of the intra-coded frame. The prediction for this method can be performed sample by sample to achieve better intra-prediction accuracy. The samples in the prediction unit are processed in predefined orders where the raster scanning and vertical scanning processing order is applied to vertical and horizontal angular predictions. The reference samples around the right and bottom boundaries of the prediction unit are padded. As an entropy coder, HEVC employed context-based adaptive binary arithmetic coding (CABAC).

We've explored multiple video compression techniques varying from prediction based coding to transform coding. In their most basic form videos are 3-dimensional datasets, but what about other 3-dimensional datasets? These include magnetic resonance imaging (MRI) and computerized tomography (CT) scans. MRI scans combine a magnetic field, radiofrequency waves, and a computer to create detailed images of your body. The resulting images are based on how the water molecules in your body move in response to the magnetic field. CT scans, on the other hand, are 3-dimensional x-rays, where instead of an x-ray beam being fixed in place, it rotates 360° around you taking hundreds of pictures, or slices, of your body. Two proposed methods specifically for medical scans are explored: one that uses gradients to form a prediction and another that implements a wavelet transform [27].

Taquet and Labit proposed a new hierachial approach to resolution scalable lossless and near-lossless compression [28]. Their method combines the adaptability of DPCM schemes with new hierarchical oriented predictors to provide resolution scalability. This proposed prediction is not very efficient on smooth images, thus they also introduce new predictors, which are dynamically optimized using a least-square criterion. They refer to their hierarchical oriented prediction as HOP.

HOP begins with hierarchical decomposition which can be summarized in two steps. A common decomposition is IHINT, where the first step (HStep) consists of predicting the pixels of H using an interpolative finite impulse response filter on L. H then contains the residual values of the prediction. The next step (VStep), is the mathematical transposition of HStep applied independently on L to obtain two sets, LL and LH. The transposition is then applied on H to obtain HL and HH. This is visually illustrated in Figure 2.4. The proposed approach to hierarchical decomposition is similar to IHINT and is illustrated in Figure 2.5. In the proposed method, the first step (HStep) consists of predicting horizontally odd indexed pixels with the aid of already known pixels. This proposed approach uses not only the even indexed pixels but it can also take advantage of any previoulsy predicted pixels, which are now causal values. In the second step (VStep), the mathematical transposition of HStep is applied and acts on the lower resolution images.

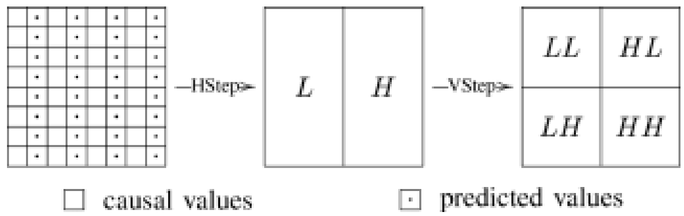

Figure 2.4: One Prediction Level of IHINT Algorithm[28]

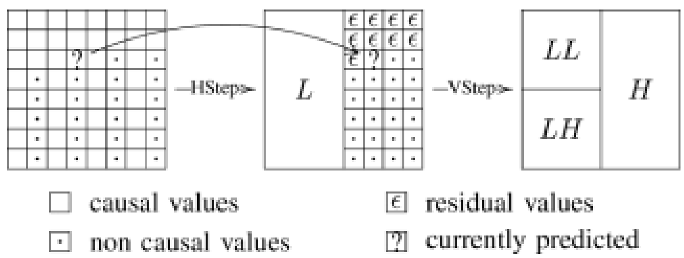

Figure 2.5: One Prediction Level of HOP Algorithm[28]

Most medical images are noisy images that contain structured objects with sharp edges, therefore HOP is designed for images such as those. Thus an orientation estimation is done using the pattern presented in Figure 2.6. To start, the absolute value of the local differences is computed for each orientation belonging to the set. Then, using a noise threshold T_{noise}, the diagonal gradient and the horizontal/vertical gradient are used to find the most favorable orientation which is then selected for the prediction. T_{noise} is set to a noise estimation that is computed on the highest frequencies of an orthonormal Haar transform of the image to compress. This threshold allows for a slight improvement of compression on noisy datasets for HOP and smooth datasets for HOP-LSE$^+$. However, on noisy CTs, HOP-LSE$^+$ compression is not improved because the noise in CTs have strong location dependence and thus cannot be captured by using the least square optimization process.

Since HOP's prediction is not effective on smooth images due to the small prediction support size that is not adequate for the decorrelation of diffuse information.

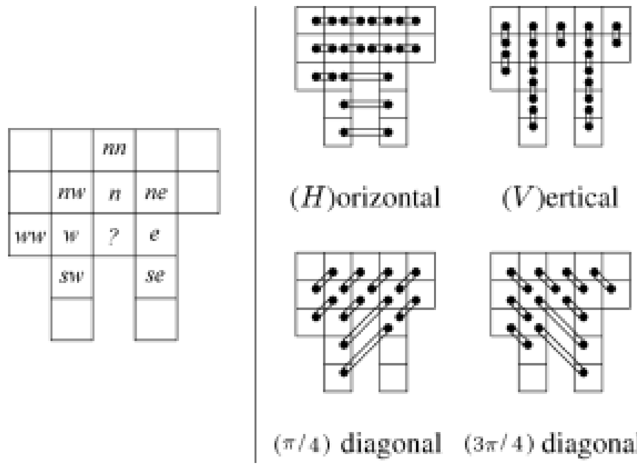

Figure 2.6: Contextual Prediction Pattern (left) and Linked Pixels used for Gradient Estimation (right)[28]

Therefore, two new predictive approaches, HOP-LSE and HOP-LSE$^+$ are introduced. These two predictive approaches exploit the extended sets of causal pixels compared with HOP. These two approaches are dynamically built using the least square estimations, giving a better adaptation to the specific characteristics of each image. Figure 2.7 illustrates the causal pixels used for HOP-LSE and HOP-LSE$^+$.

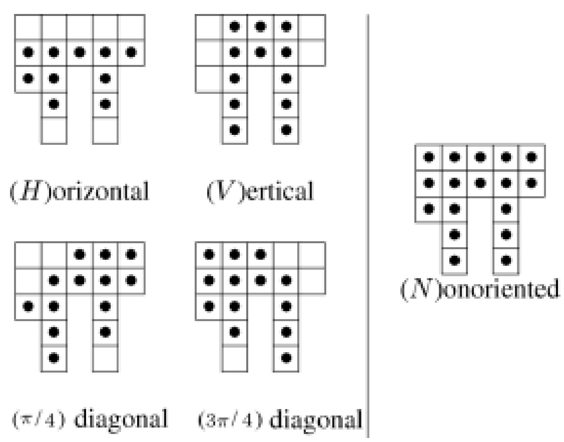

Figure 2.7: Set of Causal Pixels used for HOP-LSE (left) and HOP-LSE$^+$ (right)[28]

To avoid systematic errors, or biases, that generally occur within context-based static predictors, a common technique to correct the prediction in a context is to use the average of the previous errors that occurred within the same context. To do this, you add the average of the errors to your current prediction. To better accommodate for the biases, Taquet and Labit propose a sequential context-based error correlation (SCEC). SCEC always allows the asymptotic improvement of the compression, with further improvements for smooth images. They do this by sequentially applying the following correction scheme for $k = 1$ to K in equation 2.1. Here, $\widehat{x}$ is the prediction, $\in$ is the prediction error, $\mu_k^{\in}(C_k(\widehat{x}_{k-1}))$ is the average prediction in a context, and α_k is a coefficient that is fixed to $\frac{1}{K}$.

$$\widehat{x} \leftarrow \widehat{x}_{k-1} + \alpha_k \mu_k^{\in}(C_k(\widehat{x}_{k-1})) \tag{2.1}$$

.

Taquet and Labit proposed a method based on gradient prediction, similar to CALIC. Anusuya, Raghavan, and Kavitha, on the other hand, propose a prediction to losslessly compress MRI images using a 2D-stationary wavelet transform (SWT) [29]. Their system proposes to implement a lossless codec using and entropy coder. This method provides random access as well as resolution and quality scalability to the compressed data. Here, random access refers to the ability to decode any section of the compressed image without having to decode the entire dataset.

The main objective of this system is to effectively implement lossless compression by reducing the amount of data that is required to represent a digital image. Figure 2.8 illustrates the architectures of this system. The original 3-dimensional medical images is given as an input that is then converted into 2-dimensional slices. Then, segmentation is used to extract the region of interest alone for the 2-dimensional slices. Then, the extracted information is decimated using a 2-dimensional SWT. These decimated coefficients are then compressed in parallel using embedded block coding with optimized truncation of the embedded bit stream. These bit streams are then decoded and reconstructed using the inverse SWT. The system concentrates on minimizing the time computation by introducing parallel computing on the arithmetic coding stage as it deals with multiple subslices.

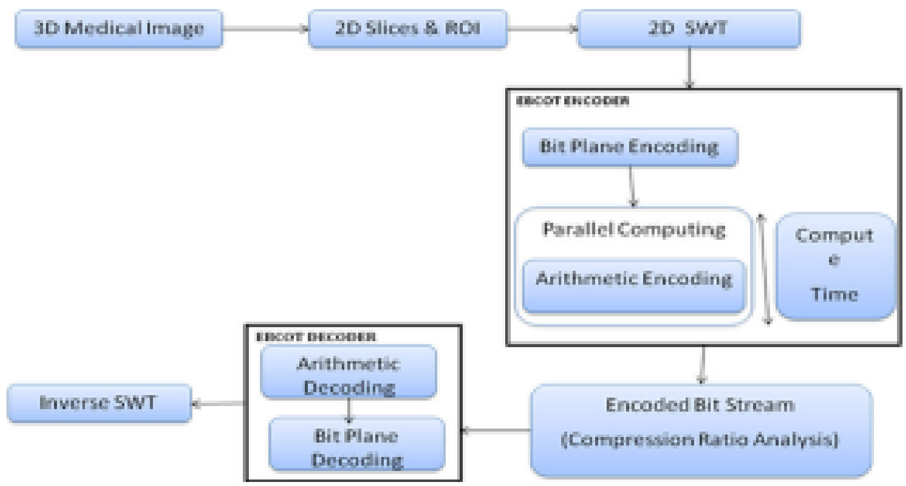

Figure 2.8: SWT Block Diagram[29]

Many techniques varying from transforms to prediction-based coding have been previously evaluated for lossless video compression. Li composed a lossless video sequence compression that uses adaptive preditions while Yang and Frayer implemented interframe and intraframe coding modes to form a context-based predictive coder. Using significant bit truncation to losslessly compress high definition videos was developed by Kim and Kyung and was also discussed. Choi and Ho's compression method modifies the context-based adaptive binary arithmetic coding mechanism to convert from lossly to lossless coding which considers the statistical differences of the residual data between lossy and lossless coding.

Similarly, some medical imaging, such as MRIs and CT scans, are 3-dimensional. Taquet and Labit proposed a prediction-based coding method that, like CALIC, depend on the gradients in the 2-dimensional frames. They used a hierachial approach to resolution scalable lossless compression combining the adaptability of DPCM schemes with new hierarchical oriented predictors. The other method we delved into used a 2-dimensional stationary wavelet transform (SWT) proposed by Anusuya, Raghavan, and Kavitha. Their system implements a lossless codec using an entropy coder and provides random access as well as resolution and quality scalability to the compressed data.

Now that we've seen what others have done, the necessary background information for the proposed method will be discussed before delving into the proposed method itself. One commonality between each video and scan is motion and how much change there is between frames. Classification is implemented based on the amount of motion between frames or scans and the prediction is found accordingly. This idea of classification is used to determine if Match or CALIC should be the prediction method implemented on each dataset.

Chapter 3: Preliminary Information

There are two main methods of entropy coding, Huffman and arithmetic. Huffman encoding is dependent on prefix codes that are optimal for a given model, or set of probabilities. Prefix codes are the bit strings represent some particular symbol. The procedure for Huffman encoding is based on two observations regarding the optimal prefix codes: the first is in an optimal code, symbols that occur more frequently will have shorter codewords than symbols that occur less frequency; and the second is that in an optimal code, the two symbols that occur the least frequently will have the same length. In cases where the alphabet is small and the probability of occurrence of the different letters is skewed, Huffman coding can become inefficient when compared to the entropy. Where the entropy is the lowest rate at which the source can be coded. One way to avoid this issue is to block more than one symbol together and generate an extended Huffman code. Unfortunately, however, this approach does not always work. Due to the shortcomings of Huffman coding, adaptive arithmetic coding was selected to encode the datasets used in this thesis.

Instead of simply compressing the datasets with strictly adaptive arithmetic coding, a pixel value is predicted and the error is then encoded to improve compression. One of the best prediction methods that uses the gradients in an image to predict the value of the current pixel is a Context-Based Adaptive Lossless Image Compression technique, which is referred to as CALIC. So we do not have to worry about encoding negative values, the error calculated by taking the actual pixel value minus the prediction is remapped to a value between 0 and 255 before being encoded.

CALIC is a linear prediction method and depends only on the current image. On the other hand, Match is a non-linear prediction method that looks at the previous frame of the video to make a prediction. Due to this non-linearity, the correlation between frames is examined at in two ways: the structural similarity and edge quality. The global structural similarity looks at local patterns of pixel intensities while the edge quality measurement looks at how the edges of an image shift between the frames. The greater the structural similarity the greater the compression ratio, however the smaller the edge quality measurement is the greater the compression ratio.

This section first delves into the specifics of adaptive arithmetic coding, both from the encoder side and the decoder side. Next, CALIC is explained in terms of how the linear prediction works as well as how the error is remapped. The discussion shifts to the correlation between the frames of videos explaining how to calculate the structural similarity and edge quality measurements.

3.1 Adaptive Arithmetic Coding

Adaptive arithmetic coding was utilized by me because arithmetic coding is better than Huffman coding for sources with skewed probabilities and adaptive coding outperforms a fixed model in terms of compression efficiency. Arithmetic coding is represented with a probability model, where the probabilities are counts. In adaptive arithmetic coding, the probability model is initialized by setting the counts for all possible variables to one and initializing cumulative counts based on the initial counts. However, the counts and cumulative counts start at index one and not zero because count[0] must not be the same as count[1]. Therefore, a translation table that maps the range of the pixels, [0, 255], to symbols [1, 256] is also initialized. This model is then updated with each pixel that's encoded.

To encode a pixel, the symbol that represents the pixel on the translation table must be determined. From there, the upper and lower limits must be updated. As shown in equations (3.1) and (3.2), where x is the symbol that's to be encoded, u is the upper limit, and l is the lower limit.

$$l = l + \left\lfloor \frac{(u - l + 1)\text{CumCount}(x - 1)}{\text{TotalCount}} \right\rfloor \tag{3.1}$$

$$u = l + \left\lfloor \frac{(u - l + 1)\text{CumCount}(x - 1)}{\text{TotalCount}} \right\rfloor - 1 \tag{3.2}$$

Now that the upper and lower limits have been updated, the most significant bits must be checked. If the most significant bits match, then that bit is encoded and shifted out, with a zero shifting into the least significant bit for the lower limit and a one shifting into the upper limit. If the most significant bits differ, then nothing is encoded and the scale 3 condition is checked. A scale 3 conditions occurs when the second most significant bit of the upper limit is 0 and 1 for the lower limit. If this occurs, then the most significant bit of each limit is shifted out with a 0 or 1 being shifted in depending on the limit. From there, the most significant bit is then complemented, resulting in a 1 for the most significant bit for the upper limit and a 0 for the lower limit. A variable is then incremented each time this occurs. The next time a bit is encoded, the complement of that bit is encoded for each scale 3 condition that has been seen. This then clears the variable count for that condition.

Once the pixel is encoded, the model is then updated. This is done by first checking if the total count is equal to the maximum count. This maximum count is chosen based off of the size of the upper and lower limits, also referred to as word lengths. Given a word length m, it is possible to only accommodate a total count of 2^{m-2} or less. In this case, the upper and lower limits are unsigned sixteen bit integers, therefore the total counts is 2^{14} - 1 = 16,383. If the total count is equal to the set maximum count, then all of the counts are then scaled down by two and rounding up the result so that no count gets rescaled to zero. From there, the model reorders the symbols to place the current one in its correct rank in the cumulative count ordering. This keeps the cumulative counts in descending order and is kept track of through the translation tables. The final step of updating the model is to increment the appropriate count and adjusts the cumulative counts accordingly. Once the entire image has been encoded, the lower limit is then written to the file.

To decode what's been encoded, a tag value is incorporated. The tag is the same size as the upper and lower limits (unsigned 16 bit integer), and is initialized as the first 16 bits that were encoded. To decode a symbol, the following equation, equation 3.3, is utilized.

$$\left\lfloor \frac{(t - l + 1)\text{TotalCount} - 1}{u - l + 1} \right\rfloor \tag{3.3}$$

The resulting value is compared to the cumulative counts. Whatever cumulative count the value is less than but greater than or equal to the previous count, the symbol that corresponds to that cumulative count is the decoded symbol. To determine which pixel value the symbol represents, the symbol is translated to a character through the index to pixel array.

The decoder then repeats the same algorithm as the encoder, only whatever bit manipulation is performed to the upper and lower limits is also performed on the tag value, which each new bit being pulled in from the encoded bits [31].

3.2 Context-Based Adaptive Lossless Image Compression

CALIC, a context-based adaptive lossless image compression technique, utilizes the gradients in an image to predict the pixel value. CALIC assumes a given pixel has a value close to one of its neighbors. Which neighboring pixel that is, is dependent on the local structure of the image. When the neighboring pixels are close to that of the current pixel and there's little variation, CALIC provides a better prediction and thus a smaller error and larger compression ratio. However, when there are hard lines in the image and the current pixel isn't close to one of it's neighbors, CALIC had a harder time predicting the value which results in a larger error and less compression. Therefore, CALIC takes into consideration the environment of the pixel to be encoded to make the prediction. Figure 3.1 illustrates the context, or the neighborhood, that CALIC references to make the prediction.

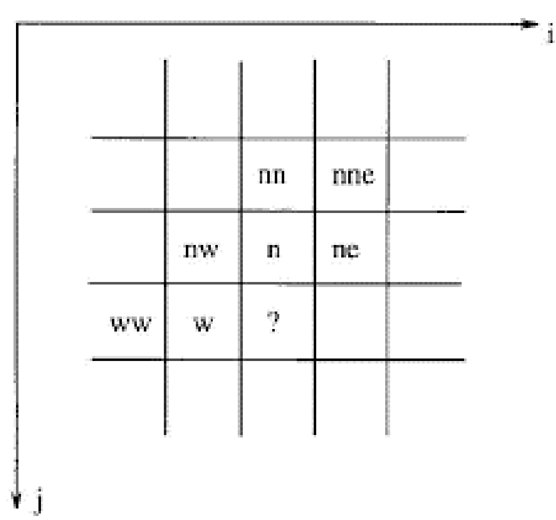

Figure 3.1: CALIC Neighborhood [32]

To get an idea of what boundaries may or may not be in the neighborhood of the current pixel, the horizontal, d_h, and vertical, d_v, gradients are calculated using equations 3.4 and 3.5 respectively.

$$d_h = |w - ww| + |n - nw| + |ne - n| \tag{3.4}$$

$$d_v = |w - nw| + |n - nn| + |ne - nne| \tag{3.5}$$

The gradients are used to determine how to predict the pixel as they take into consideration the surrounding texture and are used to make a determination about the environment of the pixel to be encoded. If the vertical gradient is much larger than the horizontal gradient, then there is a large amount of vertical variation, thus the initial prediction is taken to be w. Having a large vertical gradient makes the assumption that there's a horizontal edge, therefore the prediction should be a pixel value on the same row as the current location, hence why the initial prediction is w. However, if the horizontal gradient is much larger than the vertical gradient, then there's a large amount of horizontal variation and therefore n is chosen to be the initial prediction. A large horizontal gradient assumes that there's a sharp vertical edge in the image. Thus selecting a pixel in the same column as the current location becomes the initial prediction. If the gradients are closer together, the predicted value becomes a weighted average of the neighboring pixels. The exact algorithm for calculating the prediction, $\widehat{x}$, is illustrated in listing 3.1.

Once the gradients have been calculated, they're first checked to see if there's large horizontal or vertical variations which assumes there's a sharp edge. If there isn't large horizontal or vertical variation, then the prediction becomes a weighted average of the neighboring pixels. If there's no large variation, then the pixel prediction, $\widehat{x}$, becomes a weighted average using equation 3.6 before being further refined.

$$\widehat{x} = \frac{n + w}{2} + \frac{ne - nw}{4} \tag{3.6}$$

The difference between the gradients is then used to determine how to further refine the prediction. If the difference between the gradients is greater than 32, then either equation 3.7 or 3.8 is used depending on which gradient is larger. If the horizontal gradient is larger, then equation 3.7 is used, otherwise equation 3.8 is used to form the final prediction.

$$\widehat{x} = \frac{\widehat{x} + n}{2} \tag{3.7}$$

$$\widehat{x} = \frac{\widehat{x} + w}{2} \tag{3.8}$$

If the difference between the gradients is less than 32, but greater than 8; then either equation 3.9 or 3.10 is used to form the final prediction. Equation 3.9 is used if the horizontal gradient is greater than the vertical gradient. On the other hand, equation 3.10 is used when the vertical gradient is greater than the horizontal. However, if the difference between the gradients is less than 8, then the prediction is not altered and remains as the value found in equation 3.6.

$$\widehat{x} = \frac{3\widehat{x} + n}{4} \tag{3.9}$$

$$\widehat{x} = \frac{3\widehat{x} + w}{4} \tag{3.10}$$

```
if d_h - d_v > 80
        x_hat = n
else if d_v - d_h > 80
        x_hat = w
else{
        x_hat = (n + w)/2 + (ne - nw)/4
        if d_h - d_v > 32
                x_hat = (x_hat + n)/2
        else if d_v - d_h > 32
                x_hat = (x_hat + w)/2
        else if d_h - d_v > 8
                x_hat = (3 x_hat + n)/4
        else if d_v - d_h > 8
                x_hat = (3 x_hat + w)/4
        }
```

Listing 3.1: CALIC $\widehat{x}$ Prediction [33]

Once the prediction has been calculated, the error is then remapped to a value between 0 and 255. Listing 3.2 follows the psuedo code for remapping the error. x_n represents the value of the pixel, p_n is the prediction value, d_n is the prediction error, I_n is the remapped value, and m is the number of bits per pixel. Since d_n is the error, it's calculated by subtracting p_n from x_n which is illustrated in equation 3.11. There are two ways to perform remapping that depends on the value of the prediction, p_n. Note that in this situation, there are 8-bits per pixel.

$$d_n = x_n - p_n \tag{3.11}$$

```
if p_n <= (2^m) - 1{
        if |d_n| <= p_n{
                if d_n < 0
                        l_n = 2 |d_n|
                else
                        l_n = 2 |d_n| - 1
        }
else{
        if |d_n| <= 2^m - 1 - p_n{
                if d_n < 0
                        l_n = 2 |d_n|
                else
                        l_n = 2 |d_n| - 1
        }
        else{
                l_n = |d_n| + (2^m - 1 - p_n)
        }
}
```

Listing 3.2: Remapping the Error

With the error remapped to a positive value, the remapped value is then arithmetically encoded [33].

3.3 Structural Similarity

The structural similarity (SSIM) measurement compares local patterns of pixel intensities that are normalized for luminance and contrast. This system separates the task of calculating the SSIM into three comparisons: luminance, contrast, and structure. Figure 3.2 illustrates the block diagram for calculating the SSIM. One of the outputs of this measurement is "S", which is the SSIM image where each pixel is the local structural similarity measurement. These local measurements are then averaged to provide the mean SSIM: mssim.

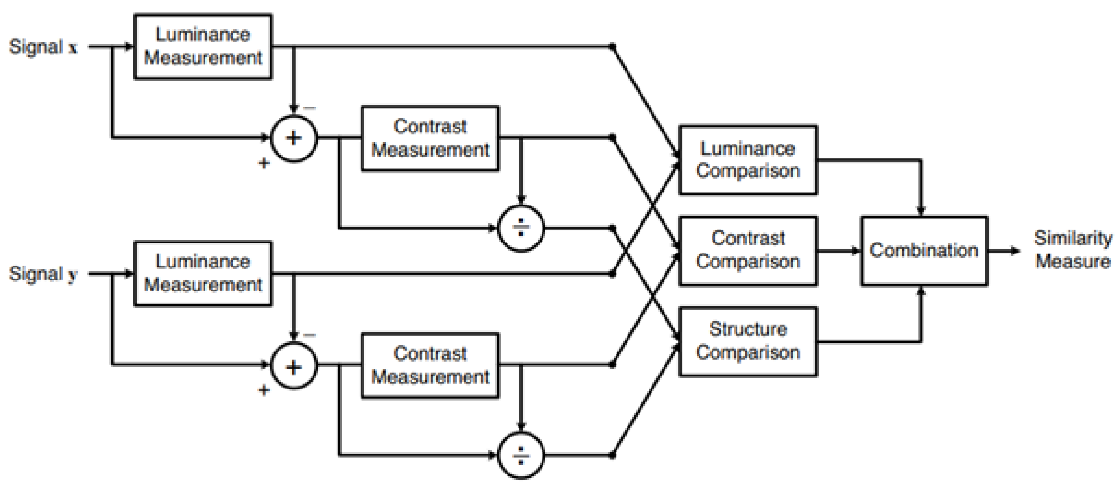

Figure 3.2: Block Diagram of SSIM Measurement [34]

The SSIM measurement starts by comparing the luminance of each signal, which is estimated as the mean intensity, illustrated in equation 3.12. This results in the luminance function, $l(\mathbf{x}, \mathbf{y})$, being a function of μ_x and μ_y. Typically this is a local mean, where there is no "structure" unless this becomes a local, patch based measurement.

$$\mu_x = \frac{1}{N} \sum_{i=1}^{N} x_i \tag{3.12}$$

The mean intensity is then removed from the signal, resulting in the signal $\mathbf{x} - \mu_x$. Standard deviation is then used to estimate the signal contrast, the unbiased estimate is given by equation 3.13. The contrast comparison, $c(\mathbf{x}, \mathbf{y})$, is the comparison of σ_x and σ_y. Similar to the luminance function, the standard deviation is also a local mean.

$$\sigma_x = \left(\frac{1}{N-1} \sum_{i=1}^{N} (x_i - \mu_x)^2 \right)^{\frac{1}{2}} \tag{3.13}$$

The signal is then normalized by dividing it by its own standard deviation. This is so the two signals being compared have unit standard deviation. The structure comparison, $s(\mathbf{x}, \mathbf{y})$, is then conducted on these normalized signals.

Once the three main comparisons are calculated, they are combined to yield an overall similarity measure, illustrated in equation 3.14. Each of the three comparisons are relatively independent.

$$S(\mathbf{x}, \mathbf{y}) = \mathbf{f}\,(\mathbf{l}(\mathbf{x}, \mathbf{y}), \mathbf{c}(\mathbf{x}, \mathbf{y}), \mathbf{s}(\mathbf{x}, \mathbf{y})) \tag{3.14}$$

This similarity measurement needs to satisfy symmetry, boundness, and unique maximum. For symmetry, $S(\mathbf{x}, \mathbf{y})$=$S(\mathbf{x}, \mathbf{y})$; for boundness, $S(x, y) \leq 1$; and for unique maximum, $S(\mathbf{x}, \mathbf{y}) = 1$ if and only if $\mathbf{x} = \mathbf{y}$.

The luminance comparison is defined by equation 3.15, where the constant C_1 is included to avoid instability when $\mu_x^2 + \mu_y^2$ is very close to zero. This constant is determined by $C_1 = (K_1 L)^2$, where L is the dynamic range of the pixel values and $K_1 << 1$.

$$l(\mathbf{x}, \mathbf{y}) = \frac{2\mu_x \mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \tag{3.15}$$

The contrast comparison, defined in equation 3.16, has a similar form to that of the luminance comparison. Here, $C_2 = (K_2L)^2$ where $K_2 << 1$. With the same amount of contrast change, $\Delta\sigma = \sigma_y - \sigma_x$, the contrast comparison is less sensitive to a case of high base contrast, σ_x, than low base contrast.

$$c(\mathbf{x}, \mathbf{y}) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \tag{3.16}$$

The definition of the structure comparison is illustrated in equation 3.17. The correlation between the two signals is a simple and effective measure to quantify the structural similarity. As in the luminance and contrast measurements, a small constant, σ_{xy}, is introduced. This constant can be estimated in discrete form using equation 3.18.

$$s(\mathbf{x}, \mathbf{y}) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3} \tag{3.17}$$

$$\sigma_{xy} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)(y_i - \mu_y) \tag{3.18}$$

These three comparisons are then combined, resulting in the similarity measure, or the SSIM index, defined in equation 3.19. If we set $\alpha = \beta = \gamma = 1$ and $C_3 = \frac{C_2}{2}$, the SSIM measurement simplifies to equation 3.20.

$$SSIM(\mathbf{x}, \mathbf{y}) = [l(\mathbf{x}, \mathbf{y})]^\alpha \cdot [c(\mathbf{x}, \mathbf{y})]^\beta \cdot [s(\mathbf{x}, \mathbf{y})]^\gamma \tag{3.19}$$

$$SSIM(\mathbf{x}, \mathbf{y}) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{3.20}$$

The SSIM index has a range from 0 to 1, where 1 signifies a perfect match between the two images being compared [34].

3.4 Edge Stability

Edge stability is defined as the consistency of edges that are evident across different scales in each frame of the videos. Edge maps are generated using simple edge detection for different scale parameters. Where the simple edge detection is using the gradient images where a value greater than a threshold means it's an edge. Before the derivatives of the image are calculated, the image is first blurred using a Gaussian filter with standard deviations of $\sigma_s = 1.19$, 1.44, 1.68, 2.0, and 2.38.

Sobel filters in the x and y direction are then implemented to determine the derivative of the images, where the magnitude of the derivatives is defined as C. A threshold is then calculated depending on the maximum and minimum values of the norm of the gradient output, which are referred to as C_{max} and C_{min}, respectively. Once C_{max} and C_{min} have been determined, the threshold for each σ_s is calculated using equation 3.21.

$$T^s = 0.1(C_{max} - C_{min}) + C_{min} \tag{3.21}$$

Now that the threshold, T^m, has been found, the edge map at scale σ_m of image C is obtained using equation 3.22. Here, $C^s(i,j)$ is the output of the derivative of the Gaussian operator at the s^{th} scale. Equation 3.23 illustrates the Gaussian filter which is used to calculate $C^s(x,y) = C(x,y) **G_s(x,y)$, where $**$ represents convolution.

$$E(x,y,\sigma_m) = \begin{cases} 1 & C^m(i,j) > T^m \text{ at } (i,j) \\ 0 & \text{otherwise} \end{cases} \tag{3.22}$$

$$G_m(x,y) = \frac{1}{2\pi\sigma_m^4} xy \exp\left\{ -\frac{x^2+y^2}{2\sigma_m^2} \right\} \tag{3.23}$$

The edge stability map, $Q(i, j)$, is then constructed by considering the longest subsequence $E(i, j, \sigma_m)...E(i, j, \sigma_{(m+l-1)})$ of the edge maps such that $Q(i, j) = l$ where equation 3.24 is true.

$$l = {}^{argmax}_{\quad l} \bigcap_{\sigma_m \leq \sigma_k \leq \sigma_{m+l-1}} E(i, j, \sigma_k) = 1 \tag{3.24}$$

Once the edge stability maps are created, each frame is compared to the next frame by calculating the edge stability mean square error (ESMSE), which is calculated using Equation 3.25. Here, M and N are the dimensions of the edge stability maps. The smaller the ESMSE, the more stable the edges are [35].

$$ESMSE = \frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} [Q_1(i, j) - Q_2(i, j)]^2 \tag{3.25}$$

In this chapter we delved into the process of arithmetic encoding and decoding. One of the best predictive lossless image compression techniques, CALIC, was discussed as well as how to remap the error to be a positive 16-bit integer. Once the compression information has been discussed, we shifted to talking about how the frames of each video are correlated through the structural similarity as well as an edge quality measurement. Now that all of the necessary background information has been presented, we will now delve into the proposed algorithm of Match. Once Match's algorithm has been explained, we then evaluate the technique by comparing its compression ratio's to CALIC. The two classification methods, the structural similarity and edge quality measurements, will be used to predict which method results in the best compression.

Chapter 4: Match

The first time Match was introduced, it was presented as a prediction method that used conditional averages to obtain the prediction, known as CAP, a Conditional Average Prediction [33]. This method uses the history of what's already been encoded of the image to find a neighborhood that matches the neighborhood of the current pixel. The value of the pixel with the same neighborhood is then the prediction. However, finding a neighborhood that is the same as the current one is unlikely, and thus a conditional average prediction is applied.

To start, a neighborhood is defined to be the contexts for which a match will be searched for. Using the labeling in Figure 3.1 the neighborhood that is used consists of w, nw, n, and ne. In order to generate a prediction, the encoder looks for matches to the context in the already encoded portion of the image. The matching criteria can be rigid where an exact match is needed or it can be defined more loosely where the match is accepted if the difference between the pixels is less than a threshold. Here, to guard against the possibility of a bad prediction, this proposed algorithm requires at least five matches to be observed before a prediction can be generated. If more than five matches are found, then the algorithm takes the average of the pixels that have the matching contexts as the predicted value. If, however, the larger context cannot garner five or more matches, then the algorithm shifts to the next smaller context: w, nw, and n. If there are no sufficient matches for the smallest context allowed, then the algorithm uses the version of the median adaptive predictor used in JPEG-LS, equation 4.4, as a default.

From there, Match was adapted and presented by Babacan as a prediction algorithm based on estimating the conditional expectation of a pixel [36]. Statistically, the optimal estimate, in the mean squared sense, of a random variable X with observations $\{Y_i\}$ is the conditional expectation of X given $\{Y_i\}$ illustrated in equation 4.1.

$$E\left[X|\left\{Y_i\right\}\right] = \sum xP\left[X = x|Y_1 = y_1, ..., Y_N = y_N\right] \tag{4.1}$$

The optimal predictor of the current pixel is then the conditional expected value $E\left[X_{i,j}|\left\{X_{i-l,j-m}\right\}_{(l,m)=(1,1)}^{i,j}\right]$. Here, $X_{i,j}$ is assumed to be conditionally independent of the surrounding pixels that are some distance from it which limits the conditional variables to be the pixels in the causal neighborhood, or context, of $X_{i,j}$. The context, or neighborhood, for this method depends on the pixels directly surrounding the current one, like in CALIC, illustrated in Figure 3.1. In jointly Gaussian processes, the conditional expectation can be simplified to a linear combination of the observations. However, if the process in non-Gaussian, the computation of the conditional expectations requires the conditional probability density function. It is difficult to assume that that image pixels are Gaussian and the conditional density required to obtain the optimal prediction is not available.

Since calculating the optimal predictor is impossible, the prediction method is simplified to depend on the textual information that is found in images. Therefore, given a pixel $x_{i,j}$, $C_{i,j}$ is the set of pixels in the causal context of $x_{i,j}$. The pixels found in this causal context are referred to as $x_1^{i,j}$, $x_2^{i,j}$, ... , $x_k^{i,j}$. Given a set of values, $\overline{\alpha} = (\alpha_1, \alpha_2, ..., \alpha_k)$, $C_k(\overline{\alpha})$ is defined in equation 4.2.

$$C_k(\overline{\alpha}) = \left\{ x_{l,m} : x_1^{l,m} = \alpha_1, x_2^{l,m} = \alpha_2, ..., x_k^{l,m} = \alpha_k \right\} \tag{4.2}$$

Once we have $C_k(\overline{\alpha})$, $E[X_{i,j}|x_1^{i,j} = \alpha_1, x_2^{i,j} = \alpha_2, ..., x_k^{i,j} = \alpha_k]$ can be estimated by the sample mean in equation 4.3, where $||\cdot||$denotes the cardinality.

$$\widehat{\mu}_{X|\overline{\alpha}} = \frac{1}{||C_k(\overline{\alpha})||} \sum_{x \in C_k(\overline{\alpha})} x \tag{4.3}$$

When using this approach, the size and composition of the context first needs to be decided. It is important that $||C_k(\overline{\alpha})||$ is large enough that $\widehat{\mu}_{X|\overline{\alpha}}$ is a good estimate. However, if $||C_k(\overline{\alpha})||$ is not large enough for $\widehat{\mu}_{X|\overline{\alpha}}$ to be a good estimate, the MED predictor used in JPEG-LS becomes the default prediction method. This method, illustrated in equation 4.4 uses the same context labels from CALIC in Figure 3.1, where $\widehat{X}$ is the prediction.

$$\widehat{X} = \begin{cases} min\left\{n, w\right\} & \text{if } nw \geq max\left\{n, w\right\} \\ max\left\{n, w\right\} & \text{if } nw \leq min\left\{n, w\right\} \\ n + w - nw & \text{otherwise} \end{cases} \tag{4.4}$$

It was found that contexts of sizes greater than four gave marginal gains over the smaller contexts. Thus, the context consisting of w, nw, n, and ne were used.

However, when implementing this version of Match, it did not perform better than CALIC. Therefore, the algorithm was adjusted to form a non-linear prediction method. This new version of Match utilizes the similarity between frames in a video. Not only was Match adjusted to form a non-linear prediction method, but it more heavily falls back onto how the method was initially presented. The main difference, however, is that instead of the previously encoded portion of the image being searched, the previous frame of the video is. The other major difference is that instead of shrinking the context to find a direct match, a threshold value is increased each time until a matching context is found.

The context consisting of w, nw, n, and ne remained the same; however, these pixel values are no longer used to calculate the prediction, instead they become search parameters. To find the prediction, Match searches the previous image to find the closest matching context. When searching the previous image, a match close to the current pixel location results in a more accurate prediction. Therefore, when searching the previous image, only a portion of it is looked at, illustrated in Figure 4.1. The previous image is searched some distance, D, to the left, right, above and below the current pixel location. The best distance for each video is different with no clear reasoning as to why that distance works the best, therefore, the distance is found through trial and error.

To start, the frame of the previous image is searched for a context that is identical to the current context. Due to the differences between frames, an exact match is not common, therefore a threshold needs to be implemented. If no exact match is found, then the frame is searched again where any of the pixels in the context can differ from the current context by a threshold of one. If no match is found, then the threshold is increased and the frame is searched until a match is found. The pixel in the previous image that has the closest matching neighborhood then becomes the prediction. For simplicity, the frame is searched from top left to bottom right.

Once the prediction has been found, the error is calculated, then remapped and encoded using adaptive arithmetic encoding. However, this method is dependent on the previous image, so the very first image in the video needs to be compressed in a different way. Since CALIC is the best lossless image compression method, it is used to compress the first image of the videos.

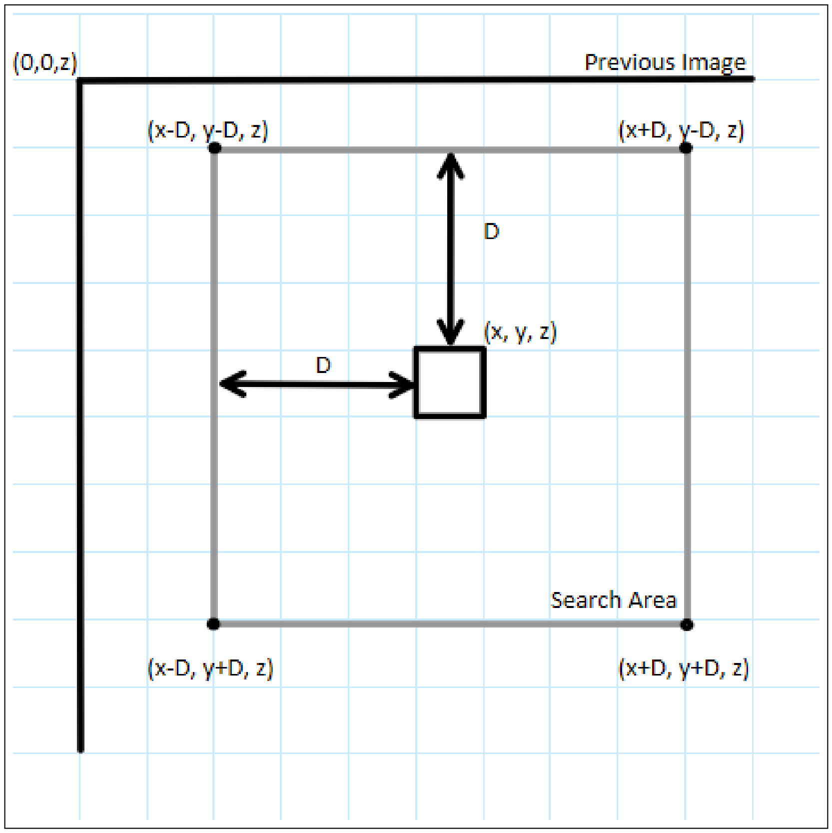

Figure 4.1: Distance

Match has one major drawback, like all other search methods, it is computationally complicated. Therefore, as the distance increases and the resolution of the datasets increases, so does the amount of time it takes for the algorithm to run.

The proposed algorithm of match has been described, so we are now going to go through the results. The resulting compression ratio at varying distances is looked at to find the best distance for each dataset. The largest resulting compression ratio is then compared to that of CALIC to evaluate how well the proposed algorithm works. From there, resolution and frame rate are examined to see how Match is affected by these variables. Finally, the structural similarity and edge quality measurements are looked at to predict which of these two methods should be used to compress each dataset.

Chapter 5: Results

Since the proposed algorithm has now been explained, we need to explore how well this method performs. Therefore, to determine how efficient Match is, its compression ratio, CR, will be compared to that of CALIC's. The larger the compression ratio, the more efficient the method of compression is. The compression ratio is a measurement of the relative reduction in size of data and is calculated using equation 5.1. The uncompressed size was calculated by how many bits are contained in the videos or scans and the compressed size was found by counting how many bits were encoded.

$$CR = \frac{\text{Uncompressed Size}}{\text{Compressed Size}} \tag{5.1}$$

There are five video sets at resolution [144, 176, 3], five video sets at resolution [1280, 720, 3], eight video sets at resolution [1920, 1080, 3], and four video sets at resolution [4096, 2160, 3] from [37]. These resolutions will be referred to as 176, 720, 1080 and 4k respectively. Along with the various video datasets, 10 datasets containing multiple C.T. scans, and 17 M.R.I scans were also compressed from [38]. For each dataset, the first 6 images were compressed. The medical datasets were compressed from distances one to fifteen while the video frames were compressed to varying distances based on where the largest compression ratio was found.

The compression ratio of these datasets is looked at as the distance for Match varies. The largest compression ratio is then compared to CALIC's compression ratio. From there, resolution and frame rates are looked at.

5.1 176 Resolution

Five video sets were tested at resolution [144, 176, 3]: Claire, Claire at six frames per second, Carphone, Foreman, and Miss_Am. Table 5.1 contains Match's compression ratios as the distance varies between one to ten pixels and are plotted in Figure 5.1. Looking at the plot in Figure 5.1, as the distance increases, the compression ratio evens out and does not vary much. However, the plot also illustrates that the compression ratio slightly decreases after the best distance is found. A distance of two pixels resulted in the largest compression ratio of 2.606 for Claire while a distance of four pixels provided the best compression ratio of 2.439 for Claire at six frames per second. Carphone has the largest compression ratio of 1.834 with a distance of three pixels. Similar to Claire at six frames per second, Foreman also performed the best at a distance of four pixels with a maximum compression ratio of 1.835. Miss_Am, on the other hand, has the largest compression ratio, 2.022, at a distance of six pixels.

Distance	Claire	Claire (6fps)	Carphone	Foreman	Miss_Am
1	2.571	2.348	1.748	1.716	1.938
2	**2.606**	2.414	1.829	1.795	1.990
3	2.590	2.436	**1.834**	1.824	2.009
4	2.581	**2.438**	1.832	**1.835**	2.019
5	2.568	2.431	1.829	1.833	2.017
6	2.567	2.428	1.826	1.828	**2.022**
7	2.551	2.421	1.822	1.822	2.018
8	2.547	2.418	1.822	1.816	2.020
9	2.539	2.415	1.816	1.811	2.015
10	2.535	2.413	1.813	1.806	2.017

Table 5.1: Match CRs with Varying Distance - 176 Resolution

Selecting the pixel from the previous frame with the closest matching context is

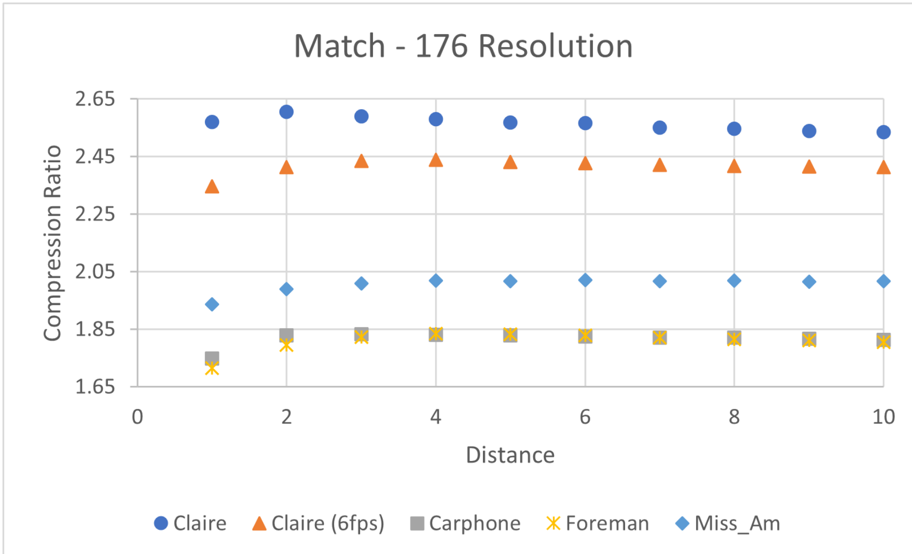

Figure 5.1: Match CRs with Varying Distance - 176 Resolution

one way to predict what the current pixel would be. However, what if CALIC was used to predict the pixel using the matching context of the previous frame instead of the current context? The compression results for this prediction method are in Table 5.2 which are plotted in Figure 5.2. For each of the datasets, the best distance to use was found to be ten pixels. Using this distance results in a compression ratio of 2.338 for Claire, 2.333 for Claire at six frames per second, 1.762 for Carphone, 1.600 for Foreman, and 2.001 for Miss_Am.

Distance	Claire	Claire (6fps)	Carphone	Foreman	Miss_Am
1	2.238	2.183	1.655	1.515	1.877
2	2.301	2.267	1.708	1.570	1.936
3	2.317	2.301	1.719	1.589	1.969
4	2.323	2.313	1.722	1.596	1.979
5	2.327	2.319	1.724	1.598	1.987
6	2.331	2.323	1.725	1.599	1.991
7	2.333	2.326	1.726	1.599	1.995
8	2.333	2.328	1.726	1.600	1.997
9	2.333	2.331	1.726	1.600	2.000
10	**2.338**	**2.333**	**1.726**	**1.600**	**2.001**

Table 5.2: CALIC with Match Context CRs with Varying Distance - 176 Resolution

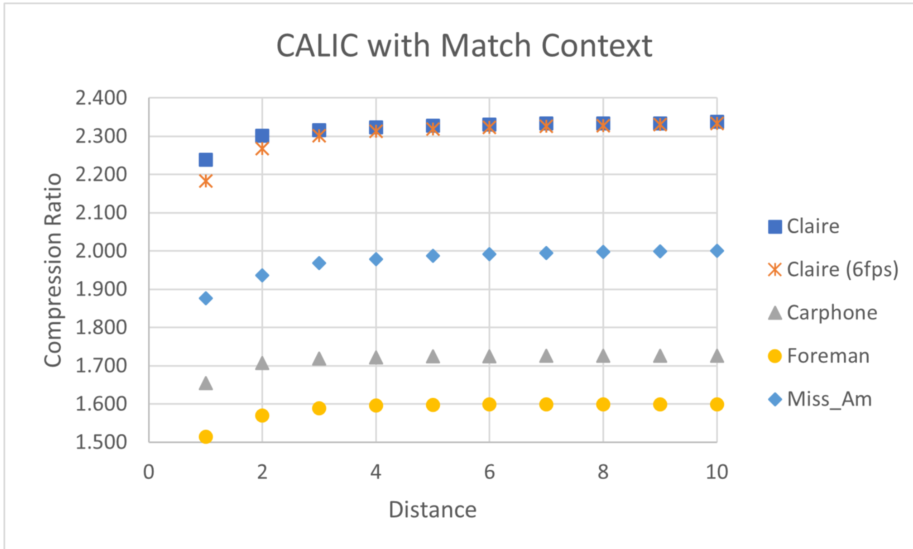

Figure 5.2: CALIC with Match Context CRs with Varying Distance - 176 Resolution

To evaluate how well Match performs, the maximum compression ratios are compared to CALIC as well as CALIC with the context found for Match. For all of the videos, but one, Match outperformed CALIC, as illustrated in Table 5.3. Figure 5.3 illustrates the direct comparison between the compression ratio of CALIC and the maximum compression ratio found for Match. The percent increase for Claire was calculated to be 9.313% while Claire at six frames per second only increased by 2.193%. Carphone increased by 5.015% and Foreman was found to have the greatest increase of 13.971%. The only outlier, Miss_Am, had a decrease of 1.555%. Fore Claire and Carphone, it can be said that Match outperforms CALIC, while it only slightly outperforms CALIC for Carphone. Due to small percent differences for Claire and six frames per second and Miss_Am, Match is comparable to CALIC for these two datasets.

When using CALIC with the closest matching context from the previous frame to predict the current pixel, it results in the smallest compression ratio of the three methods. Comparing this method to CALIC results in minor percent differences;

where CALIC outperforms this method by 1.938% for Claire, 2.223% for Claire at six frames per second, 1.130% for Carphone, 0.627% for Foreman, and 2.458% for Miss_Am. The compression ratios when using the context found with Match for CALIC seem to approach CALIC's compression ratio as the distances increases, especially with such minor percent differences. When comparing this method to Match, Match outperformed it with a percent difference of 10.292% for Claire and 12.811% for Foreman. Claire at six frames per second and Carphone, on the other hand, only have minor percent increases, 4.309% and 5.874% respectively, when using Match. Miss_Am is the only dataset where all three methods have minor percent differences, where the one between this method and Match is 0.912%. Since this method is found to be the worst prediction method, it isn't tested on the remaining datasets.

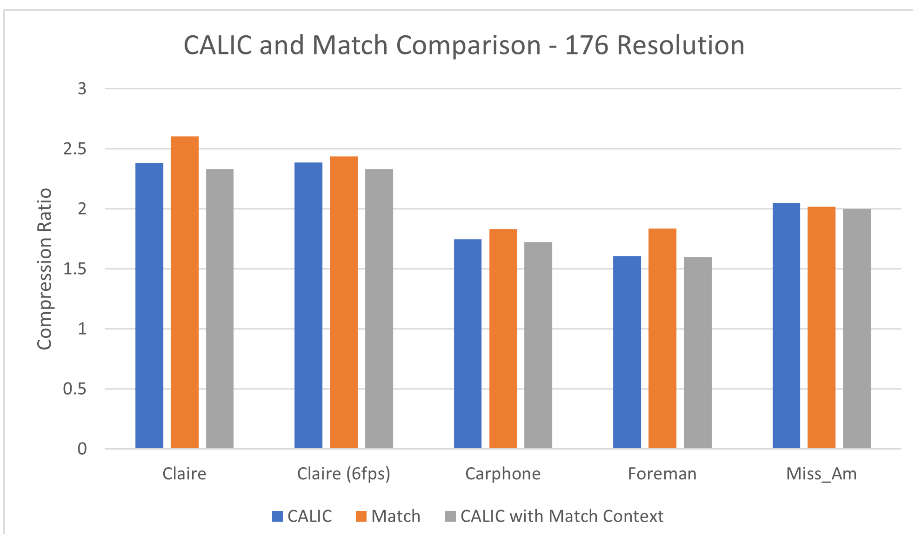

Figure 5.3: CALIC's CR Compared to Match's CR Compared to CALIC with Match's Context - 176 Resolution

Li [23] implemented their algorithm on Claire, Miss_Am, and Foreman; however, they present their data with the bit rate while the compression ratio was looked at for this thesis. Therefore, these two methods cannot be directly compared.

Method	Claire	Claire (6fps)	Carphone	Foreman	Miss_Am
CALIC	2.384	2.386	1.746	1.610	2.051
Match	2.606	2.438	1.834	1.835	2.019
CALIC with Match Context	2.338	2.333	1.726	1.600	2.001
Percent Difference (Match and CALIC)	9.312	2.179	5.040	13.975	-1.560
Percent Difference (Match and CALIC with Match Context)	10.292	4.309	5.874	12.811	0.912
Percent Difference (CALIC and CALIC with Match Context)	1.938	2.223	1.130	0.627	2.458

Table 5.3: CALIC's CR Compared to Match's CR - 176 Resolution

5.2 720 Resolution

Five data sets, Johnny, KristenAndSara, Mobcal, Parkrun, and Sheilds of resolution [1280, 720, 3] were compressed using Match. Figure 5.4 illustrates how the compression ratio for Match changes with the distance; the values are listed in Table 5.4. These datasets were tested from distance of one to five pixels, as the compression ratio started decreasing for each dataset after . A distance of four pixels results in the largest compression ratios for Johnny, Parkrun, and Shields with compression ratios of 2.572, 1.308, and 1.549 respectively. A maximum compression ratio of 2.654 was found for KristenAndSara at a distance of three pixels and Match performed the best at a distance of one pixel for Mobcal with a compression ratio of 1.563.

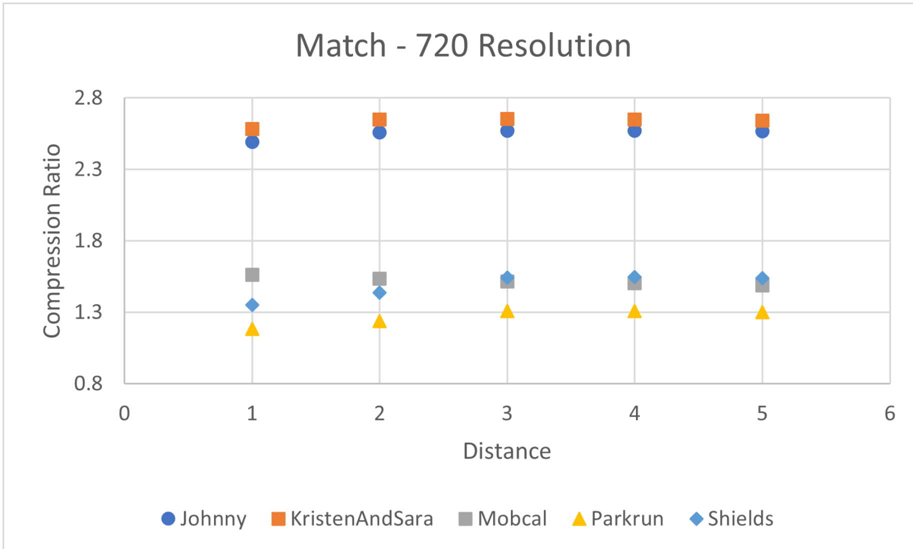

Figure 5.4: Match CRs with Varying Distance - 720 Resolution

Distance	Johnny	KristenAndSara	Mobcal	Parkrun	Shields
1	2.492	2.583	**1.563**	1.183	1.353
2	2.560	2.649	1.535	1.240	1.439
3	2.571	**2.654**	1.517	1.307	1.543
4	**2.572**	2.651	1.503	**1.308**	**1.549**
5	2.566	2.640	1.49	1.303	1.541

Table 5.4: Match CRs with Varying Distance - 720 Resolution

To evaluate how well Match performed for this resolution, the largest compression ratio found is compared to CALIC's compression ratio. These comparisons are in Table **??** and graphed in Figure **??**. It was found that Match resulted in a larger compression ratio than CALIC for Johnny, KristenAndSara, and Mobcal. The percent increases were found to be 13.906%, 15.441% and 9.684%. Due to the large percent increases, it can be said that Match outperforms CALIC. Parkrun and Shields, on the other hand, have percent decreases of 2.096% and 1.777% respectively. Due to such small percent decreases, Match results in a comparable compression ratio to CALIC for these two datasets.

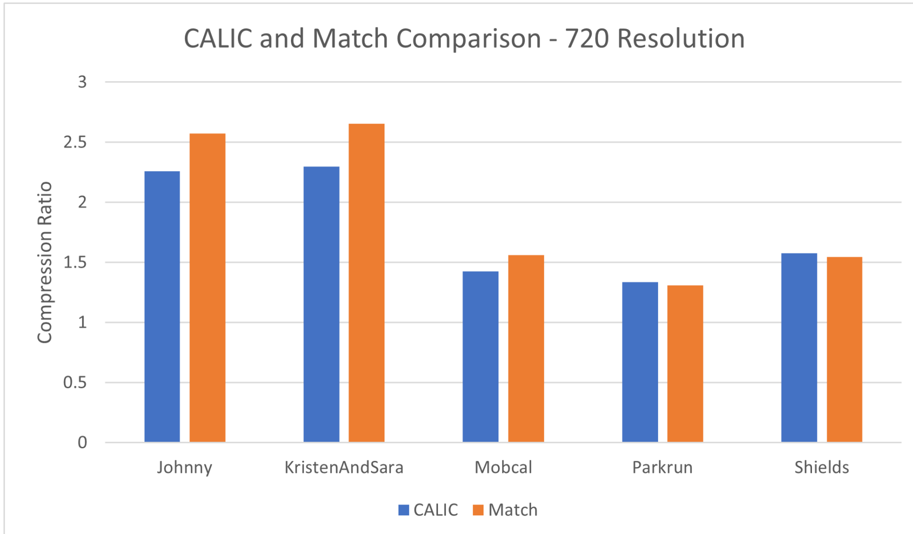

Figure 5.5: CALIC's CR Compared to Match's CR - 720 Resolution

Method	Johnny	KristenAndSara	Mobcal	Parkrun	Shields
CALIC	2.258	2.299	1.425	1.336	1.576
Match	2.572	2.654	1.563	1.308	1.548
Percent Difference	14.677	15.442	9.684	-2.096	-1.777

Table 5.5: CALIC's CR Compared to Match's CR - 720 Resolution

Choi and Ho implemented their proposed compression method of using residual data coding in CABAC for HEVC lossless video compression on Johnny and KristenAndSara. Their method resulted in compression ratios of 3.15 and 3.18 respectively which is 18.349% and 16.541% larger than the resulting compression ratios for Match [26].

5.3 1080 Resolution

The datasets for the [1920, 1080, 3] resolution are split into two sets. For the first five videos, the distance at which Match performs the best is less than fifteen, therefore the distance varies between one and fifteen. Figure 5.6 plots the compression ratios with their corresponding distance that are in Table 5.6. At a distance of thirteen pixels, Blue_Sky has the largest compression ratio of 2.046 and a distance of twelve pixels results in the largest compression ratio of 2.023 for Station2. A distance of one pixel results in the largest compression ratio of 1.945 for Controlled_Burn while a distance of six pixels is needed for the best compression ratio of 1.446 for Crowd_Run. Lastly, four pixels was found to result in the largest compression ratio of 2.597 for Life.

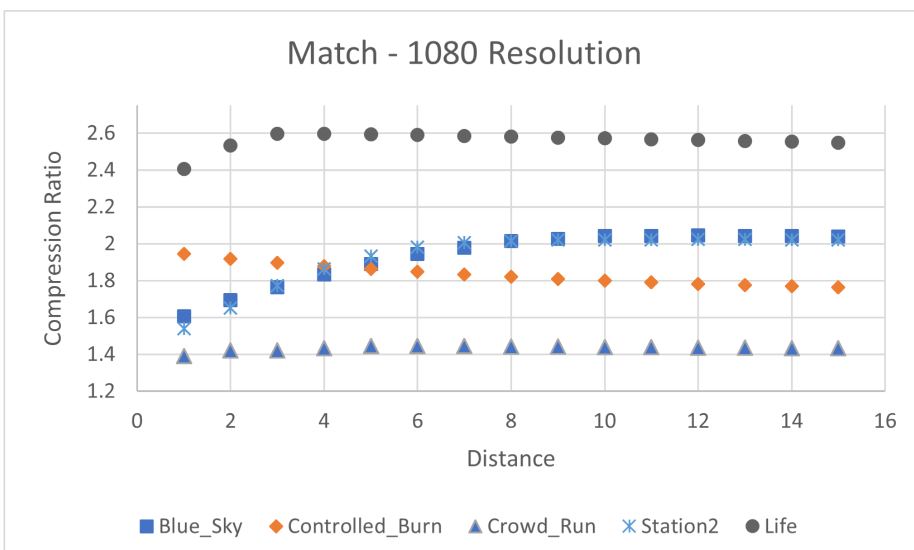

Figure 5.6: Match Compression Ratios with Varying Distance - 1080 Resolution

The second sets of 1080 resolution videos include three videos, all of which need a greater distance for the best Match results. Figure 5.7 plots the Match compression ratios with varying distance that are listed in Table 5.7. For Riverbed and Aspen,

Distance	Blue_Sky	Controlled_Burn	Crowd_Run	Station2	Life
1	1.606	**1.945**	1.391	1.538	2.406
2	1.693	1.918	1.420	1.653	2.533
3	1.764	1.897	1.420	1.772	2.597
4	1.833	1.879	1.433	1.863	**2.597**
5	1.890	1.863	1.445	1.934	2.594
6	1.946	1.847	**1.446**	1.981	2.591
7	1.979	1.834	1.445	2.005	2.585
8	2.015	1.821	1.444	2.046	2.582
9	2.027	1.810	1.442	2.021	2.577
10	2.042	1.800	1.440	2.023	2.573
11	2.043	1.791	1.439	2.023	2.567
12	**2.046**	1.783	1.437	**2.023**	2.562
13	2.044	1.776	1.436	2.023	2.557
14	2.043	1.769	1.435	2.023	2.553
15	2.041	1.762	1.434	2.022	2.548

Table 5.6: Match CRs with Varying Distance - 1080 Resolution

the best distances found were seventy and forty-one respectively. However Table 5.7 only goes to a distance of thirty pixels, which was determined to be the best distance for Dinner. Riverbed's best compression ratio was found to be 1.852, which is only 1.591% greater than Riverbed's compression ratio at a distance of thirty pixels, which was found to be 1.823. The best compression ratio for Aspen was found to be 2.266 with a distance of fourty one, which is merely 0.177% larger than the compression ratio of 2.262 at a distance of thirty.

To determine how well Match performs, the best compression ratios are compared to the compression ratios from compressing the datasets with CALIC. The compression ratios that are illustrated in Figure 5.8 are from Table 5.8 and Table 5.9. Match outperformed CALIC for two of the datasets, Controlled_Burn and Life. The percent increase of Match was calculated to be 17.239%, and 38.634% respectively. Although Match has a better compression ratio than CALIC for Dinner, the percent increase was found to be only 1.852%. Therefore, Match performs comparably to CALIC for

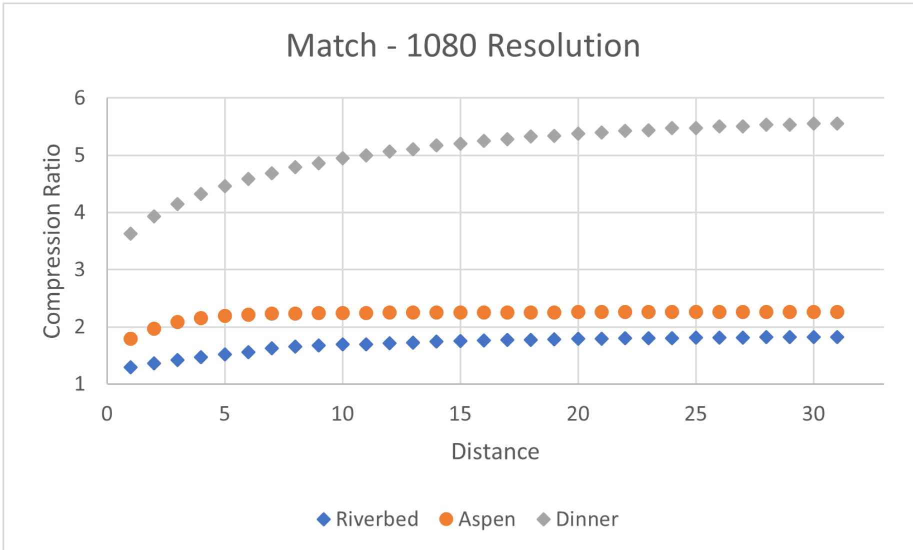

Figure 5.7: Match CRs with Varying Distance - 1080 Resolution

Dinner. On the other hand, Blue_Sky, Crowd_Run, Station2, Riverbed, and Aspen see a percent decrease when Match is used to compress the videos. The percent decreases are calculated to be -3.627%, -3.856%, -6.602%, -4.781% and -7.774% respectively. For all of these datasets but Station2 and Aspen, Match performs comparably to CALIC with negligible percent decreases.

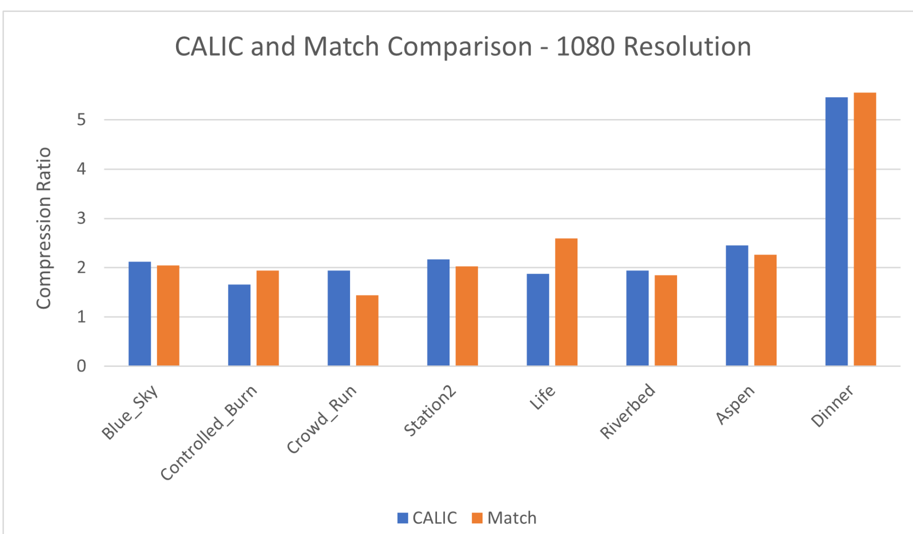

Figure 5.8: CALIC's CR Compared to Match's CR - 1080 Resolution

Distance	Riverbed	Aspen	Dinner
1	1.295	1.789	3.634
2	1.358	1.965	3.928
3	1.420	2.084	4.145
4	1.474	2.152	4.328
5	1.521	2.192	4.458
6	1.561	2.215	4.592
7	1.627	2.228	4.674
8	1.654	2.237	4.792
9	1.676	2.242	4.859
10	1.696	2.245	4.946
11	1.696	2.247	4.996
12	1.713	2.249	5.069
13	1.728	2.251	5.108
14	1.740	2.252	5.170
15	1.751	2.253	5.201
16	1.761	2.255	5.256
17	1.770	2.256	5.280
18	1.777	2.256	5.326
19	1.783	2.257	5.344
20	1.789	2.258	5.384
21	1.794	2.258	5.396
22	1.799	2.259	5.432
23	1.803	2.259	5.441
24	1.807	2.260	5.472
25	1.810	2.261	5.472
26	1.813	2.261	5.505
27	1.816	2.261	5.509
28	1.818	2.262	5.532
29	1.821	2.262	5.532
30	**1.823**	**2.262**	**5.555**

Table 5.7: Match CRs with Varying Distance - 1080 Resolution

Method	Blue_Sky	Controlled_Burn	Crowd_Run	Station2	Life
CALIC	2.123	1.659	1.504	2.166	1.874
Match	2.046	1.945	1.446	2.023	2.598
Percent Difference	-3.627	17.239	-3.856	-6.602	38.634

Table 5.8: CALIC's CR Compared to Match's CR - 1080 Resolution

Method	Riverbed	Aspen	Dinner
CALIC	1.945	2.457	5.454
Match	1.852	2.266	5.555
Percent Difference	-4.781	-7.774	1.852

Table 5.9: CALIC's CR Compared to Match's CR - 1080 Resolution

5.4 4k Resolution

Match was used to compress four datasets at a resolution of [4096, 2160, 3]: Netflix_Boat, Netflix_BoxingPractice, Netflix_Narrator, and Netflix_Tango. The compression ratios with varying distance can be found in Figure 5.9 with the values listed in Table 5.10. Due to the large size of the images, the runtime for Match drastically increased, and further increases as the distance increases. Therefore, the 4k videos were tested until the compression ratio decreased.

A distance of six pixels results in the largest compression ratio, 1.075, for Netflix_Boat while a distance of five pixels is best for Netflix_BoxingPractice which results in a compression ratio of 1.103. Netflix_Narrator is best compressed by Match at a distance of four pixels with a compression ratio of 1.144. Netflix_Tango, on the other hand, has the best compression ratio of 1.079 at a distance of twelve pixels.

To evaluate how well Match performed, the largest compression ratios are compared to the compression ratios of CALIC. These ratios are charted in Figure 5.10 with their values listed in Table 5.11. For only one of these datasets, Netflix_Narrator, Match slightly outperforms CALIC with a percent increase of 5.333%. For the remaining datasets, Match performed comparably to CALIC with small percent increases. Netflix_Boat, Netflix_BoxingPractice, and Netflix_Tango have percent differences of 0.422%, 2.309%, and -0.678% respectively. The percent increases could be minimal due to the resolution of the datasets or due to the structural similarity between the frames of the videos.

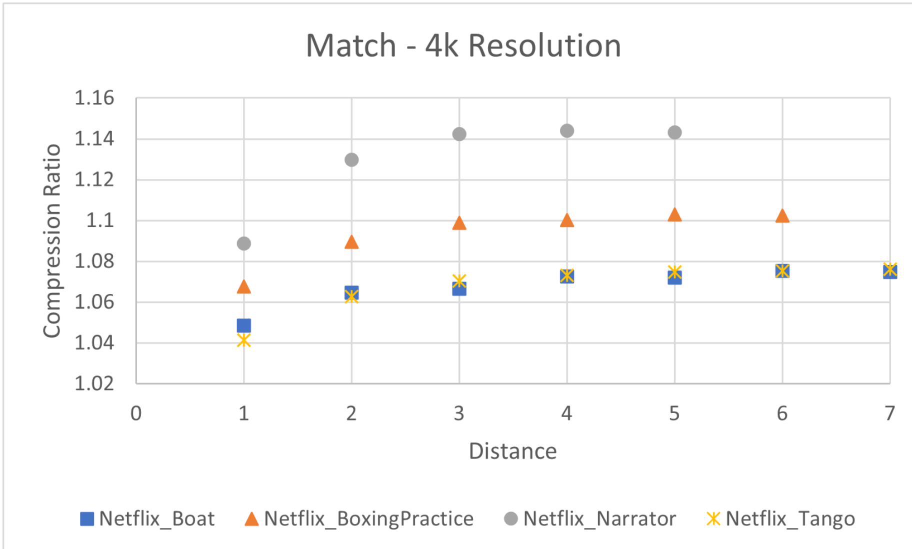

Figure 5.9: Match CRs with Varying Distance - 4k Resolution

Distance	Netflix_Boat	Netflix_BoxingPractice	Netflix_Narrator	Netflix_Tango
1	1.0484	1.0677	1.089	1.041
2	1.065	1.089	1.123	1.063
3	1.067	1.099	1.142	1.070
4	1.073	1.100	**1.144**	1.073
5	1.072	**1.103**	1.143	1.075
6	**1.075**	1.102	-	1.075
7	1.075	-	-	1.076
8	-	-	-	1.076
9	-	-	-	1.077
10	-	-	-	1.078
11	-	-	-	1.078
12	-	-	-	**1.079**

Table 5.10: Match CRs with Varying Distance - 4k Resolution

Method	Boat	BoxingPractice	Narrator	Tango
CALIC	1.071	1.078	1.086	1.086
Match	1.075	1.103	1.144	1.079
Percent Difference	0.422	2.309	5.333	-0.678

Table 5.11: CALIC's CR Compared to Match's CR - 4k Resolution

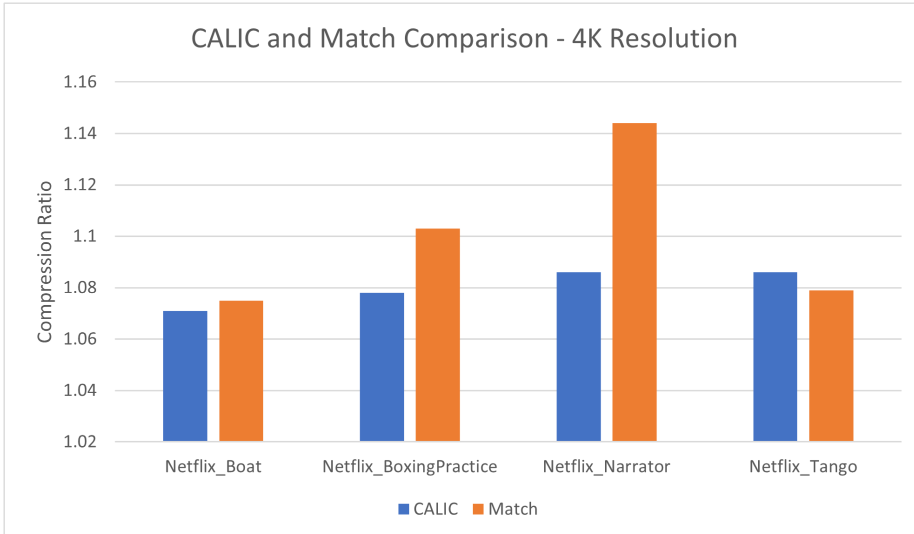

Figure 5.10: CALIC's CR Compared to Match's CR - 4k Resolution

In some cases, Match outperformed CALIC; in others, CALIC outperformed Match; and in the remaining videos, both methods performed nearly the same. Most often, a distance of one resulted in the lowest compression ratio. Overall, as the distance varied for each of the video datasets the compression ratio of Match varied slightly. The greatest variation was found to be 1.921 for the video Dinner in the 1080 resolution datasets. At a distance of one, Match results in the smallest compression ratio of 3.634 while the largest compression ratio found at a distance of thirty was 5.555.

Miss_Am was the most frustrating dataset as Match resulted in a larger compression ratio for the rest of the 176 resolution datasets with the smallest percent increase being 2.179%. Similar to Claire, the frames of Miss_Am didn't visually change much frame to frame; which leads one to expect that Match would perform better as it's dependent on the contexts being as close of a Match as possible.

Since Match is dependent on how much one frame varies from the next, it can be assumed that as the frame rate decreases the resulting compression ratio also decreases as the differences between the frames will be greater. The only datasets which show this are Claire and Claire(6fps) where Claire(6fps) is every sixth frame of the original video. The largest compression ratio for Claire was found to be 2.606 while the compression ratio of Claire(6fps) was 2.438. Therefore, Claire is better compressed by 0.168 in comparison to Claire(6fps).

For all but one of the 176 resolution videos, Match outperformed CALIC; for all but two of the 720 resolution videos, Match outperformed CALIC; however, only three of the 1080 videos, Match outperformed CALIC; and Match barely outperformed CALIC for all but one of the 4k videos. Excluding the results of the 4k resolution videos as the cost for using Match doesn't outweigh the slight increase in compression, one can assume that as the resolution of the video increases the less likely Match outperforms CALIC. Not only does the likelihood of Match performing better seem to decrease, but it takes longer for Match to run, especially as the distance increases.

5.5 C.T. Scans

Match was tested on various C.T. scans that were collected from the National Cancer Institute [38]. Like the video datasets, the first six images of each scan were compressed; however, each dataset was tested from a distance of one to fifteen pixels.

According to the Mayo Clinic, "a computerized tomography (CT) scan combines a series of X-ray images taken from different angles around your body and uses computer processing to create cross-sectional images (slices) of the bones, blood vessels and soft tissues inside your body. C.T. scan images provide more-detailed information than plain X-Rays do" [39].

The C.T. datasets are split up into different categories dependent on the label they were collected under from the National Cancer Institute. In total, 65 datasets were collected, however for the majority of these scans, the percent difference between Match and CALIC was found to be less than $|5|\,\%$. Therefore, all of the resulting Match compression ratios as the distance varies between one and fifteen pixels as well as the comparison between Match and CALIC can be found in Appendix A. The results in the appendix include the datasets that are discussed in this section.

The remaining datasets have been split into two categories, those with compression ratios that have a percent difference greater than $|5|\,\%$ but less than $|10|\,\%$ and those with a percent difference of greater than $|10|\,\%$.

Firstly, the datasets with a percent difference between Match and CALIC that falls in the range between $|5|\,\%$ and $|10|\,\%$ will be discussed. The datasets that fall into this category are coronal from AMC-002; CORONAL_MPR_2MM, CT_FUSION, and CTAC from AMC-003; WB_MAC_P690 and WB_NAC_P690 from AMC-004; CHEST_7.0_MIP_Axial from AMC-005; CORONAL_SP, CT_images, and PET_BODY_NO_AC from AMC-006; Thorax_2.0_SPO_cor and Thorax_2.0_SPO_sag from AMC007; and BODY_5.000CE_1 and BODY_5.000CE_2 from CMB-CRC-MSB-02381. Tables 5.12 and 5.13 contain Match's compression ratio for these datasets as the distance varies, where Table 5.12 contains the datasets in AMC-002 through AMC-005. The compression ratios for the remaining datasets are in Table 5.13.

The following label substitutions were made for the datasets in Table 5.12: COR for CORONAL_MPR_2MM and FUSION for CT_FUSION from AMC-003; MAC for WB_MAC_P690 and NAC for WB_NAC_P690 from AMC-004; and Axial for CHEST_7.0_MIP_Axial from AMC-005. Of the seven datasets in this table, four of them have the largest compression ratio at a distance of 15. These C.T. scans include coronals with a compression ratio of 2.399, CORONAL_MPR_2MM with a compression ratio of 2.243, WB_MAC_P690 with a compression ratio of 12.528, and WB_NAC_P690 with a compression ratio of 5.545. The remaining two datasets in

AMC-003, CT_FUSION and CTAC have the largest compression ratios of 8.087 and 15.220 at distances of four and five respectively. Lastly, CHEST_7.0_MIP_Axial has the largest compression ratio at a distance of seven with a value of 4.017.

	AMC-002	ACM-003			AMC-004		AMC-005
Distance	coronals	COR	FUSION	CTAC	MAC	NAC	Axial
1	1.785	1.722	7.903	15.000	7.936	3.446	3.615
2	1.960	1.855	8.058	**15.220**	9.299	4.006	3.811
3	2.101	1.960	8.084	15.209	10.355	4.468	3.923
4	2.198	2.043	**8.087**	15.161	10.939	4.756	3.977
5	2.257	2.210	8.077	15.153	11.275	4.933	4.004
6	2.296	2.137	8.074	15.134	11.488	5.058	4.013
7	2.324	2.162	8.071	15.123	11.672	5.146	**4.017**
8	2.344	2.181	8.069	15.108	11.859	5.244	4.015
9	2.359	2.195	8.068	15.098	11.995	5.302	4.012
10	2.370	2.207	8.066	15.084	12.106	5.354	4.009
11	2.378	2.217	8.060	15.073	12.226	5.418	4.005
12	2.386	2.226	8.055	15.062	12.321	5.447	4.002
13	2.391	2.232	8.052	15.045	12.424	5.499	3.998
14	2.395	2.238	8.050	15.038	12.479	5.529	3.994
15	**2.399**	**2.243**	8.047	15.027	**12.528**	**5.545**	3.990

Table 5.12: Match CRs with Varying Distance - C.T. Scans:
$|5|\,\% <$ Percent Difference $< |10|\,\%$

Similar to Table 5.12, Table 5.13 also has substitutions for the names of the datasets. The following substitutions were made: COR for CORONAL_AP and PET_BODY for PET_BODY_NO_AC in AMC-006; cor for Thorax_2.0_SPO_cor and sag for Thorax_2.0_SPO_sag in AMC-007; and Body_1 and Body_2 for BODY_5.000CE_1 and BODY_5.000CE_2 in CMB-CRC-MSB-02381. For these datasets, Match performed best at a distance of 15 for all but CT_images in AMC-006. Match performs the best at a distance of seven with a compression ratio of 8.926 for CT_images. The largest compression ratios for the remaining datasets are 8.836, 5.636, 2.688, 2.558, 2.017, and 2.903 respectively.

	AMC-006			AMC-007		CMB-CRC-MSB	
Distance	COR	CT_images	PET_BODY	cor	sag	Body_1	Body_2
1	7.555	8.254	2.972	2.125	2.216	1.645	2.525
2	7.714	8.695	3.330	2.219	2.359	1.744	2.638
3	7.845	8.855	3.715	2.278	2.430	1.820	2.708
4	7.972	8.914	4.075	2.329	2.466	1.876	2.757
5	8.096	8.920	4.390	2.378	2.490	1.915	2.792
6	8.228	8.926	4.621	2.423	2.504	1.943	2.819
7	8.343	**8.926**	4.813	2.468	2.514	1.962	2.839
8	8.446	8.921	5.007	2.514	2.522	1.976	2.855
9	8.536	8.919	5.144	2.556	2.528	1.994	2.866
10	8.608	8.918	5.254	2.593	2.534	1.994	2.875
11	8.670	8.915	5.343	2.623	2.540	2.000	2.884
12	8.715	8.907	5.463	2.647	2.545	2.005	2.889
13	8.764	8.904	5.536	2.665	2.550	2.010	2.895
14	8.803	8.900	5.594	2.678	2.554	2.013	2.899
15	**8.836**	8.900	**5.636**	**2.688**	**2.558**	**2.017**	**2.903**

Table 5.13: Match CRs with Varying Distance - C.T. Scans:
$|5|\,\% <$ Percent Difference $< |10|\,\%$

Figures 5.11 and 5.12 illustrate how the resulting compression ratio of Match changes as the distance varies. The various scans with a percent difference in between five and ten percent were split into two plots, the first being the datasets with a compression ratio less than six and the later being those with a compression ratio larger than six. The datasets with resulting compression ratios less than six include: coronals from AMC-002; CORONAL_MPR_2MM from AMC-003; WB_NAC_P690 from AMC-004; CHESET_7.0_MIP_Axial from AMC-005; PET_BODY_NO_AC from ACM-006; the two datasests in AMC-007; and the remaining two datasets in CMB-CRC-MSB-02381. The other datasets have compression ratios greater than six and are plotted in Figure 5.12.

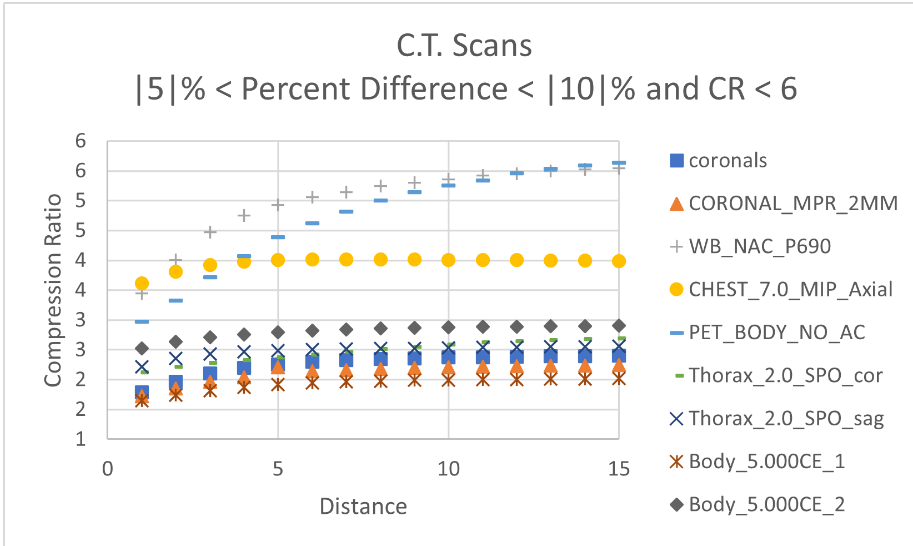

Figure 5.11: Match CRs with Varying Distance - C.T. Scans: $|5|\,\% <$ Percent Difference $< |10|\,\%$ and CR < 6

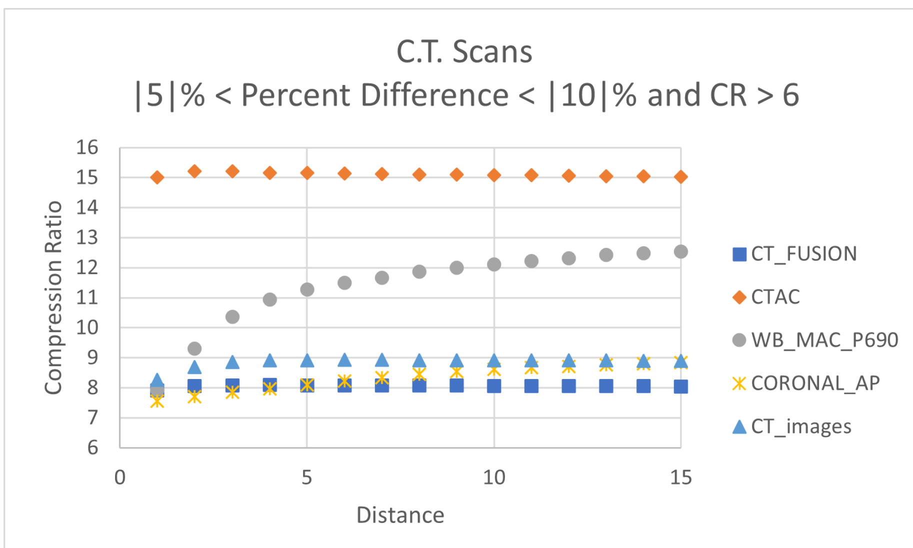

Figure 5.12: Match CRs with Varying Distance - C.T. Scans:
$|5|\,\% <$ Percent Difference $< |10|\,\%$ and CR > 6

To evaluate how well Match performed, the best compression ratio is compared to that of CALIC's. Tables 5.14 and 5.15 compare the compression ratios. These tables are split in the same way and contain the same substitutions as Tables 5.12 and 5.13. The compression ratios between Match and CALIC are plotted in the chart of Figure

5.13. In this set, clearly, Match performed the best for CTAC, not just by having the largest compression ratio, but also has the largest increase when compared to CALIC.

Between the two tables, only four of the datasets have a positive percent difference where Match results in the larger compression ratio. These datasets are CT_FUSION and CTAC from AMC-003; CHEST_7.0_MIP_Axial from AMC-005; and CT_images from AMC-005 and have percent increases of 5.451%, 8.878%, 5.934%, and 6.765% respectively. The remaining datasets have negative percent differences, which means Match underperformed when compared to CALIC. The smallest of these differences is -5.100% and is the result of the datasets Body_5.000CE_2 under CMB-CRC-MSB-02381; while the largest negative difference is -9.179% for CORONAL_AP from AMC-006.

	AMC-002	AMC-003			AMC-004		AMC-005
	coronals	COR	FUSION	CTAC	MAC	NAC	Axial
CALIC	2.546	2.415	7.669	13.979	13.415	6.034	3.792
Match	2.399	2.243	8.087	15.220	12.528	5.545	4.017
Percent Difference	-5.774	-7.122	5.451	8.878	-6.612	-8.104	5.934

Table 5.14: CALIC's CR Compared to Match's CR - C.T. Scans:
$|5|\,\% <$ Percent Difference $< |10|\,\%$

	AMD-006			AMC-007		CMB-CRC-MSB	
	COR	CT_image	PET_BODY	cor	sag	Body_1	Body_2
CALIC	9.729	8.361	6.049	2.940	2.787	2.153	3.059
Match	8.836	8.926	5.636	2.688	2.558	2.017	2.903
Percent Difference	-9.179	6.756	-6.828	-8.571	-8.217	-6.317	-5.100

Table 5.15: CALIC's CR Compared to Match's CR - C.T. Scans:
$|5|\,\% <$ Percent Difference $< |10|\,\%$

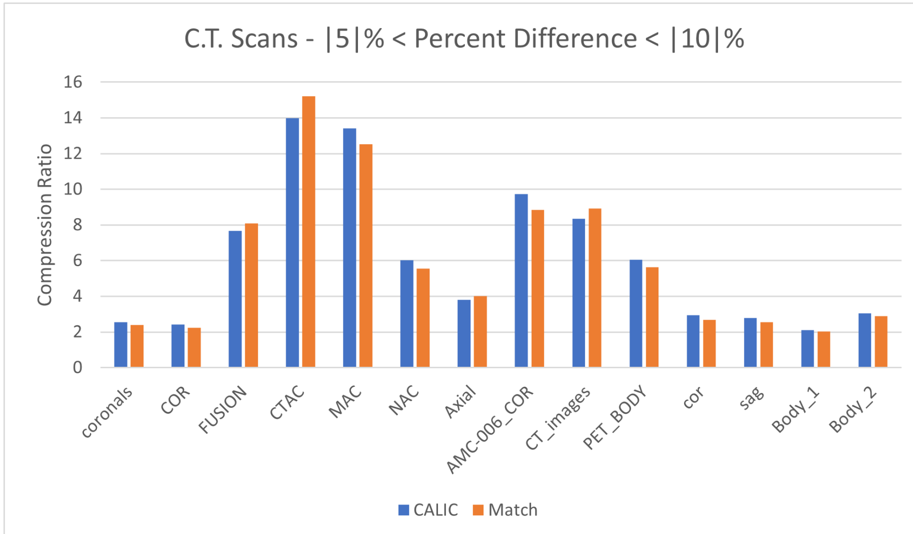

Figure 5.13: CALIC's CR Compared to Match's CR - C.T. Scans: $|5|\,\% < $ Percent Difference $< |10|\,\%$

Now that the datasets that have a percent difference that fall in the range of $|5\%|$ and $|10\%|$, the datasets that have a percent difference that's greater than $|10\%|$ will be discussed. These datasets include TCGA-38-4628 and NLST-LSS from the miscellaneous label, WB_NAC_P690 and CHST_1.25MM_SHARP from ACM-001; CHEST_2.0_coronal and CHEST_2.0_Sagittal from AMC-005; LUNG_1MM from AMC-006; and 60 from 4D-Lung. Table 5.16 contains the resulting compression ratio of Match as the distance varies between one and fifteen with the following substitutions: TCGA for TCGA-38-4628, NLST for NLST-LSS, NAC for WB_NAC_P690, CHST for CHST_1.25MM_SHARP, cor for CHEST_2.0_coronal, sag for CHEST_2.0_Sagittal, and LUNG for LUNG_1MM. These resulting Match ratios are plotted against the distance in Figure 5.14.

Similar to the datasets that have a percent difference between $|5\%|$ and $|10\%|$, the majority of these datasets also result in the largest compression ratio being at the largest distance of 15. The datasets that follow this trend are TCGA-38-4628, WB_NAC_P690, CHEST_2.0_coronal and CHEST_2.0_Sagittal with compression ratios of 2.041. 5.692, 2.512, and 2.663 respectively. The dataset 60 in the 4D-Lung C.T. scans performs best at the shortest distance of three with a compression ratio of 4.844. At a distance of one greater, four, LUNG_1MM performs the best with a resulting ratio of 2.571. Increasing the distance one more to five results in the largest compression for CHST_1.25MM_SHARP with a ratio of 2.418. NLST-LSS, on the other hand, has the largest compression ratio of 1.773 at a distance of eight.

	Miscellaneous		AMC-001		AMC-005		AMC-006	4D-Lung
Distance	TCGA	NLST	NAC	CHST	cor	sag	LUNG	60
1	1.776	1.725	3.456	2.402	2.111	2.272	2.500	4.717
2	1.847	1.754	4.025	2.413	2.190	2.400	2.557	4.829
3	1.892	1.766	4.504	2.415	2.233	2.481	2.566	**4.844**
4	1.924	1.770	4.805	2.418	2.265	2.534	**2.571**	4.839
5	1.950	1.772	4.998	**2.418**	2.293	2.571	2.570	4.831
6	1.969	1.773	5.139	2.418	2.319	2.596	2.568	4.826
7	1.985	1.773	5.238	2.417	2.349	2.614	2.564	4.824
8	1.998	**1.773**	5.349	2.417	2.379	2.629	2.561	4.817
9	2.008	1.773	5.416	2.417	2.407	2.638	2.557	4.810
10	2.016	1.772	5.481	2.416	2.431	2.644	2.554	4.806
11	2.023	1.771	5.558	2.415	2.451	2.650	2.550	4.800
12	2.029	1.770	5.589	2.414	2.470	2.655	2.546	4.793
13	2.034	1.769	5.646	2.414	2.486	2.658	2.541	4.786
14	2.038	1.768	5.675	2.413	2.500	2.661	2.539	4.780
15	**2.041**	1.768	**5.692**	2.413	**2.512**	**2.663**	2.536	4.776

Table 5.16: Match CRs with Varying Distance - C.T. Scans: Percent Difference > $|10|$ %

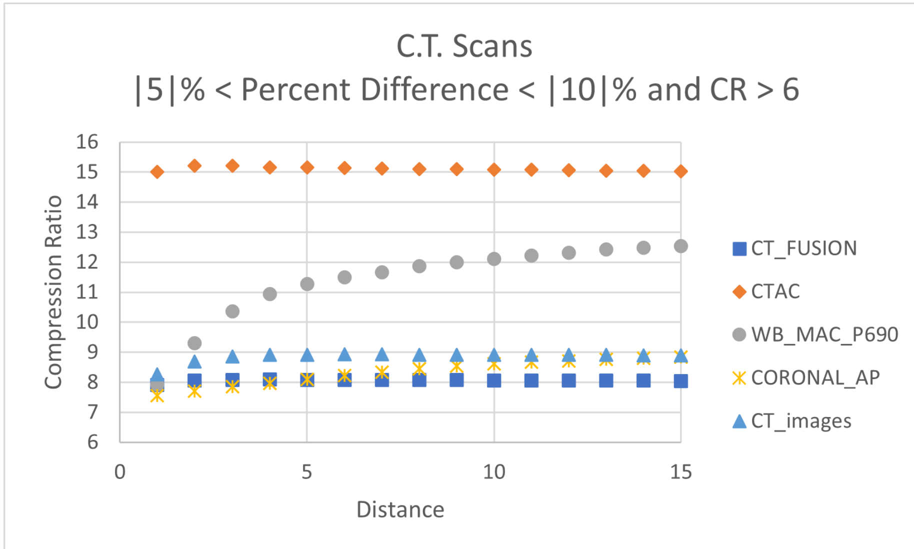

Figure 5.14: Match CRs with Varying Distance - C.T. Scans: Percent Difference > |10| %

The comparison between Match's compression ratio and CALIC's is in Table 5.17 and graphed in Figure 5.15. Match outperforms CALIC for half of these datasets: TCGA-38-4628 from miscellaneous, CHST_1.25MM_SHARP from ACM-001, LUNG_1MM from AMC-006, and 60 from 4D-Lung. Their corresponding percent increases are 13.895%, 10.159%, 20.365%, and 22.975%. CALIC results in a significantly larger compression ratio for the remaining datasets: NLST-LSS from miscellaneous; WB_NAC_P690 from AMC-001; and CHEST_2.0_coronal and CHEST_2.0_Sagittal from ACM-005. Match underperforms CALIC by -17.227%, -11.587%, -12.838%, and -10.936% respectively.

	Miscelaneous		AMC-001		AMC-005		AMC-006	4D-Lung
	TCGA	NLST	NAC	CHST	cor	sag	LUNG	60
C	1.792	2.142	6.438	2.195	2.882	2.990	2.136	3.939
M	2.041	1.773	5.692	2.418	2.512	2.663	2.571	4.844
PD	13.895	-17.227	-11.587	10.590	-12.838	-10.936	20.365	22.975

Table 5.17: CALIC's CR Compared to Match's CR - C.T. Scans: Percent Difference > |10| %

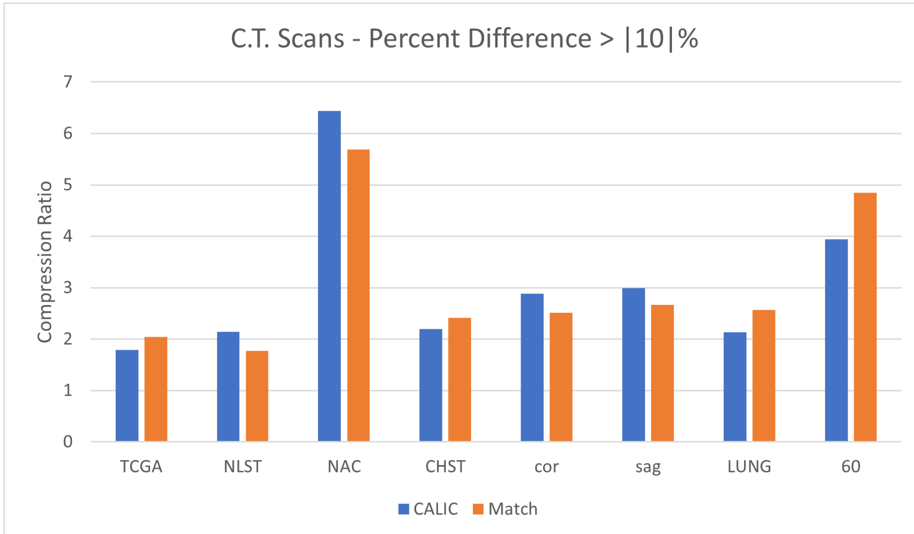

Figure 5.15: CALIC's CR Compared to Match's CR - C.T. Scans: Percent Difference $> |10|\%$

For the most part, as the distance changes, the compression ratio from Match doesn't vary much. However, there are a few datasets where Match varies greatly as the distance increases. These datasets are WB_NAC_P690 from ACM-004 and PET_BODY_NO_AC from ACM-007 in Figure 5.11; WB_MAC_P690 from AMC-004 in Figure 5.12; and WB_NAC_P690 from AMC-001 in Figure 5.14. In AMC-004, WB_MAC_P690 varies from 7.936 at a distance of one to 12.528 at a distance of 15, which is a difference of 4.592. Also in AMC-004, WB_NAC_P690 varies a total of 2.099 from 3.446 at a distance of one to 5.545 at a distance of 15. PET_BODY_NO_AC from ACM-007 varies a total of 2.664 as a distance of one results in a compression ratio of 2.972 but a distance of 15 results in a compression ratio of 5.636. Lastly, WB_NAC_P690 from AMC-001 varies a total of 2.236 as a distance of one results in a ratio of 3.456 while a distance of fifteen results in a compression ratio of 5.692. The remaining datasets have compression ratios that vary within approximately one as the distance changes.

In general, the resulting compression ratio of the C.T. datasets provides results that Match is incredibly similar to CALIC. Some examples include the LIDC-IRDI datasets all have miniscule percent differences between the compression ratios. The AMC C.T. scans. on the other hand, mostly have percent differences less than $|10|\,\%$. Only five of the thirty six datasets under AMC have a percent increase of at least 10%, with largest percent increase being 22.975%. There's only four datasets where Match underperformed CALIC with a percent decrease greater than -10% with the largest difference being -17.227%. The remaining twenty eight datasets in this category all have percent differences less than 10%, that's 77.778% of these datasets. Another overall disappointing result is that all but one of the 4D lung datasets has a percent difference of less than $|5|\,\%$.

Overall, Match's performance for the C.T. datasets is unimpressive. Out of the 65 datasets, 43 of them result in a percent difference between Match and CALIC to be less than $|5|\,\%$, which is 66.154% of the C.T. scans.

5.6 M.R.I. Scans

According to the National Institute of Biomedical Imaging and Bioengineering, a Magnetic Resonance Imaging (M.R.I.) "is a non-invasive imaging technology that produces three dimensional detailed anatomical images" [40]. 17 M.R.I. datasets under the classification ACRIN-6698-102212, were collected from the National Cancer Institute [38] and compressed with Match for distances of one to fifteen. These M.R.I. scans are split based on the compression ratios.

The first set of datasets are for a compression ratio under two, which includes ISPY2_Fieldmap, ISPY2_T2fseidealarc_BP, ISPY2_WATER_T2_fseidealarc_BP and ISPY2_Water_T2feidealarc_BP. Figure 5.16 plots the compression ratios in Table 5.18

where ISPY2_T2fseidealarc_BP is represented by T2fseidealarc_BP, ISPY2_WATER_T2_fseidealarc_BP is represented with WATER, and ISPY2_Water_T2feidealarc_BP is represented by Water. It was found that a distance of fifteen pixels results in the largest compression ratio for all of these datasets. The resulting maximum compression ratios are 1.672 for ISPY2_Fieldmap, 1.676 for ISPY2_T2fseidealarc_BP, 1.971 for ISPY2_WATER_T2_fseidealarc_BP and 1.908 for ISPY2_Water_T2_fseidealarc_BP.

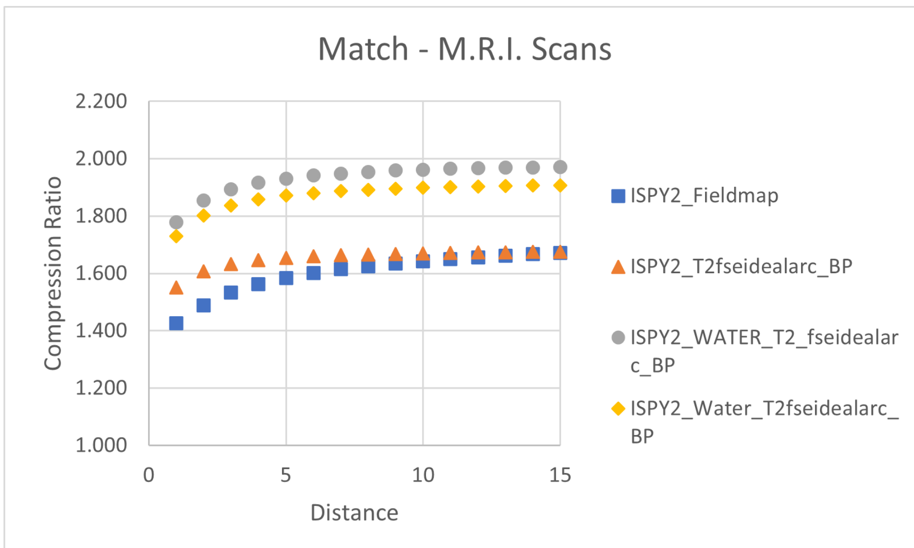

Figure 5.16: Match CRs with Varying Distance - M.R.I. Scans

To evaluate how well Match performed for these datasets, the largest compression ratio found is compared to CALIC's compression ratio. Figure 5.17 illustrates the comparison of the two with the compression ratios listed in Table 5.19. For all but ISPY2_Fieldmap, CALIC slightly outperforms Match with a percent difference of approximately 5%. ISPY2_Fieldmap, on the other hand, has a percent decrease of -15.014% when using Match. Therefore, CALIC greatly outperforms Match for this dataset.

Distance	ISPY2_Fieldmap	T2fseidealarc_BP	WATER	Water
1	1.426	1.551	1.778	1.729
2	1.489	1.608	1.855	1.801
3	1.534	1.633	1.894	1.837
4	1.563	1.646	1.916	1.858
5	1.585	1.654	1.931	1.872
6	1.601	1.659	1.941	1.881
7	1.614	1.663	1.949	1.887
8	1.625	1.666	1.954	1.892
9	1.635	1.668	1.959	1.896
10	1.643	1.670	1.962	1.899
11	1.650	1.671	1.965	1.902
12	1.656	1.673	1.967	1.904
13	1.662	1.674	1.969	1.906
14	1.667	1.675	1.970	1.906
15	**1.672**	**1.676**	**1.971**	**1.908**

Table 5.18: Match CRs with Varying Distance - M.R.I. Scans

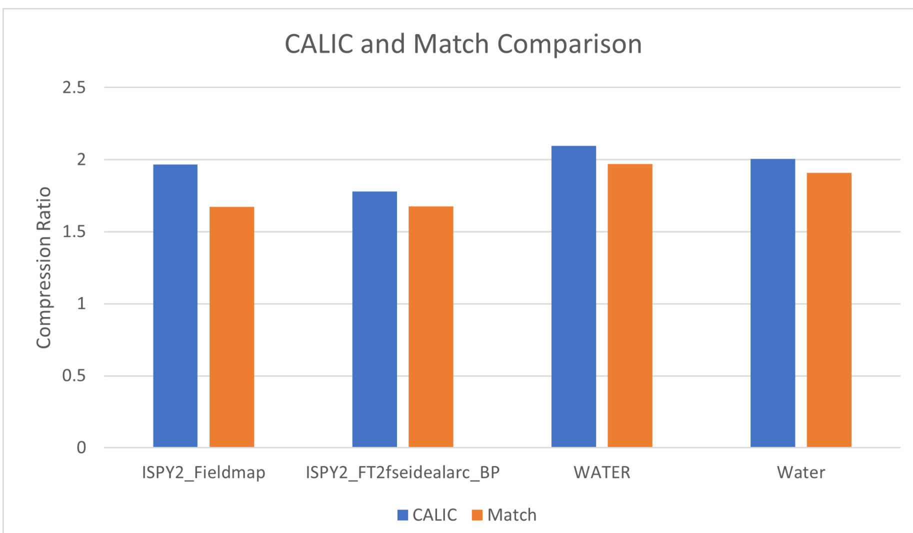

Figure 5.17: CALIC's CR Compared to Match's CR - M.R.I. Scans

Method	ISPY2_Fieldmap	T2fseidealarc_BP	WATER	Water
CALIC	1.967	1.781	2.095	2.007
Match	1.672	1.676	1.971	1.908
Percent Difference	-15.014	-5.897	-5.891	-4.966

Table 5.19: CALIC's CR Compared to Match's CR - M.R.I. Scans

The next batch of datasets consist of ACRIN-6698_ADC, ISPY2_Fat_T2fseidealarc_BP, ISPY2_IP_T2fseidealarc_BP, ISPY2_multiphase384, ISPY2_OP_T2fseidealarc_BP, and ISPY2_VOLSER_DCE which are represented in Table 5.20 and Table 5.21 with ADC, Fat, IP, multiphase384, OP, and VOLSER_DCE respectively. For these datasets, the compression ratio when compressing with Match falls between two and three. A distance of fifteen pixels results in the largest compression ratio for all but ISPY2_multiphase384 with ratios of 2.935, 2.691, 2.429, 2.591, and 2.259. Match has the largest compression ratio of 2.040 at a distance of thirteen pixels for ISPY2_multiphase384.

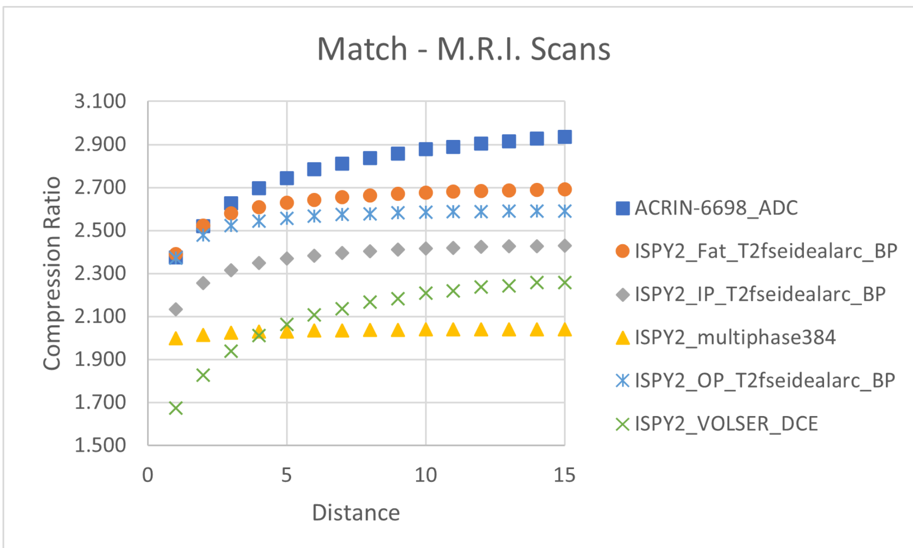

Figure 5.18: Match CRs with Varying Distance - M.R.I. Scans

Comparing Match to CALIC results in Figure 5.19 and table 5.21. For all but ACRIN-6698_ADC, CALIC slightly outperforms Match with percent differences of -7.305%, -7.901%, -3.606%, -5.660% and -4.111% respectively. ACRIN-6698_ADC has the only positive percent increase of 0.131, however, the difference is so small that Match and CALIC result in nearly identical compression ratios.

Distance	ADC	Fat	IP	multiphase384	OP	VOLSER_DCE
1	2.374	2.389	2.133	1.997	2.373	1.675
2	2.520	2.523	2.256	2.014	2.479	1.826
3	2.626	2.580	2.316	2.024	2.521	1.939
4	2.696	2.609	2.348	2.030	2.543	2.012
5	2.743	2.628	2.369	2.030	2.556	2.062
6	2.783	2.643	2.383	2.033	2.566	2.107
7	2.810	2.654	2.395	2.035	2.573	2.135
8	2.837	2.662	2.403	2.037	2.578	2.167
9	2.857	2.669	2.410	2.038	2.582	2.183
10	2.878	2.675	2.416	2.039	2.585	2.208
11	2.890	2.680	2.419	2.039	2.586	2.218
12	2.905	2.683	2.423	2.039	2.587	2.237
13	2.915	2.687	2.425	**2.040**	2.589	2.242
14	2.928	2.689	2.427	2.040	2.590	2.257
15	**2.935**	**2.691**	**2.429**	2.039	**2.591**	**2.259**

Table 5.20: Match CRs with Varying Distance - M.R.I. Scans

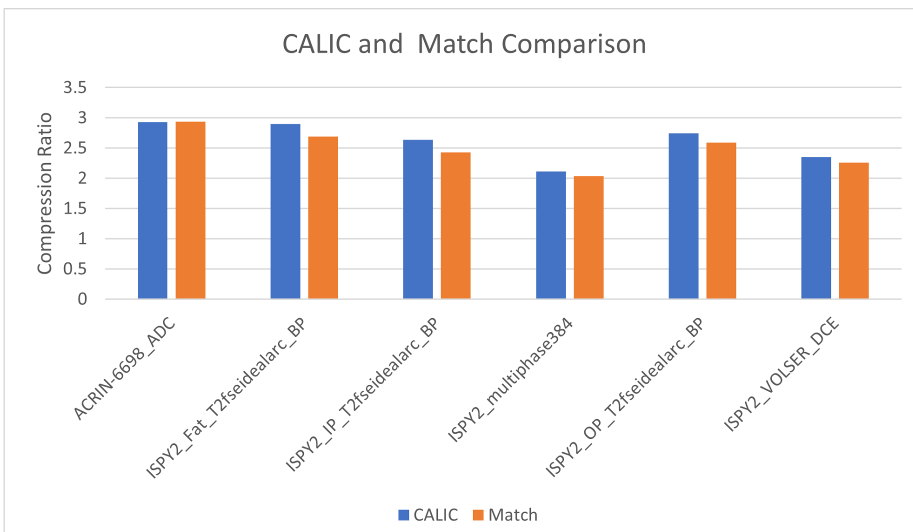

Figure 5.19: CALIC's CR Compared to Match's CR - M.R.I. Scans

Method	ADC	Fat	IP	multiphase384	OP	VOLSER_DCE
CALIC	2.931	2.903	2.637	2.116	2.746	2.355
Match	2.935	2.691	2.429	2.04	2.591	2.259
Percent Difference	0.131	-7.305	-7.901	-3.606	-5.660	-4.111

Table 5.21: CALIC's CR Compared to Match's CR - M.R.I. Scans

The compression ratios for ACRIN-6698_4bval, ACRIN-6698_DWI_TRACE, ISPY2_3_Plane_Scout, ISPY2_VOLSER_PE2, and ISPY2_VOLSTER_PE6 fall between approximately five and twenty. Figure 5.20 plots the compression ratios in Table 5.22. In Table 5.18 ACRIN-6698_4bval is represented with 4bval, DWI_TRACE represents ACRIN-6698_DWI_TRACE, ISPY2_3_Plane_Scout is represented with 3_Plane_Scout, PE2 is used in place of ISPY2_VOLSER_PE2, and similarly PE6 represents ISPY2_VOLSER_PE6. It was determined that a distance of fourteen pixels produces the largest compression ratios for ACRIN-6698_4bval, ACRIN-6698_DWI_TRACE, and ISPY2_3_Plane_Scout with values of 6.571, 20.230, and 4.486 respectively. The remaining two datasets, ISPY2_VOLSER_PE2 and ISPY2_VOLSTER_PE6, have the largest compression ratios of 18.483 and 21.237 respectively at a distance of fifteen pixels.

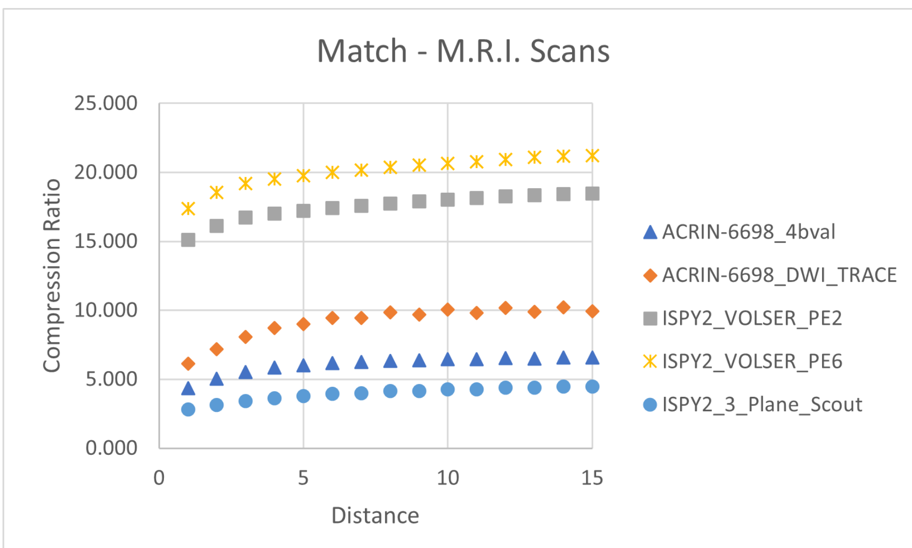

Figure 5.20: Match CRs with Varying Distance - M.R.I. Scans

Distance	4bval	DWI_TRACE	3_Plane_Scout	PE2	PE6
1	4.351	6.139	2.824	15.117	17.376
2	5.033	7.184	3.146	16.136	18.565
3	5.525	8.082	3.432	16.749	19.203
4	5.840	8.734	3.641	17.006	19.535
5	6.011	9.022	3.782	17.219	19.757
6	6.175	9.462	3.934	17.428	20.000
7	6.242	9.456	4.005	17.584	20.168
8	6.348	9.835	4.130	17.746	20.371
9	6.367	9.686	4.163	17.889	20.514
10	6.447	10.042	4.278	18.025	20.651
11	6.454	9.818	4.282	18.131	20.764
12	6.520	10.163	4.393	18.247	20.953
13	6.515	9.905	4.380	18.340	21.081
14	**6.571**	**10.230**	**4.486**	18.428	21.187
15	6.557	9.947	4.458	**18.483**	**21.237**

Table 5.22: Match CRs with Varying Distance - M.R.I. Scans

Comparing the largest compression ratios of Match to CALIC's compression ratio result in Figure 5.21 and Table 5.23. For all of these datasets, CALIC outperforms Match; however, for ISPY2_VOSER_PE2 Match and CALIC perform nearly identically as the percent difference is -1.623. When compressing ACRIN-6698_DWI_Trace and ISPY2_VOSER_PE2 with Match, it slightly under performs compared to CALIC with percent differences of -5.948% and -3.564% respectively. For the remaining datasets, ACRIN-6698_4bval and ISPY2_3_Plane_Scout, CALIC outperforms Match with percent differences of -7.344% and -10.681% respectively.

Method	4bval	DWI_TRACE	3_Plane_Scout	PE3	PE6
CALIC	7.092	10.877	5.023	18.788	22.022
Match	6.571	10.230	4.486	18.483	21.237
Percent Difference	-7.344	-5.948	-10.681	-1.623	-3.564

Table 5.23: CALIC's CR Compared to Match's CR - M.R.I. Scans

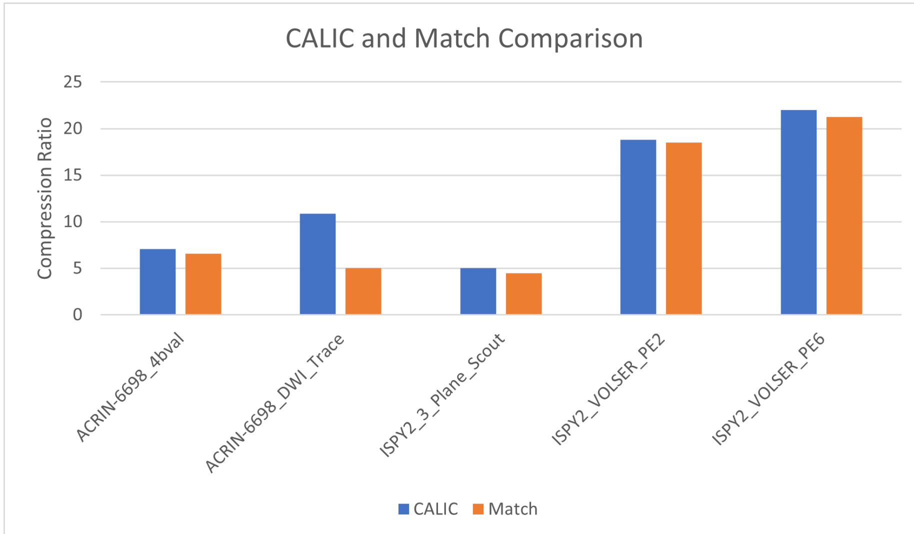

Figure 5.21: CALIC's CR Compared to Match's CR - M.R.I. Scans

The final two M.R.I. datasets are ACRIN-6698_DWI_MASK and ISPY2_Volser_SER. Both of these datasets result in compression ratios greater than 100. Figure 5.22 plots the compression ratios with varying distance found in Table 5.24. A distance of fifteen pixels results in the largest compression ratio of 108.843 for ACRIN-6698_DWI_MASK. On the other hand, Match results in the largest compression ratio of 193.808 for ISPY2_Volser_SER at a distance of three pixels.

Comparing these compression ratios to CALIC results in Figure 5.23 and Table 5.25. For both of these datasets, compressing with Match results in greater compression compared to CALIC. Match greatly outperforms CALIC for ACRIN-6608_DWI_Mask with a percent difference of 39.268%. On the other hand, Match outperforms CALIC ISPY2_Volser_SET with a percent difference of 7.389%.

Similar to the C.T. scans, the majority of the results from the M.R.I. scans are unimpressive, and even disappointing. Out of the 17 M.R.I. datasets, Match outperforms CALIC for only two of these datasets: ACRIN-6698_DWI_MASK and ISPY2_Volser_SER. The compression ratio for ACRIN-6698_DWI_MASK increases

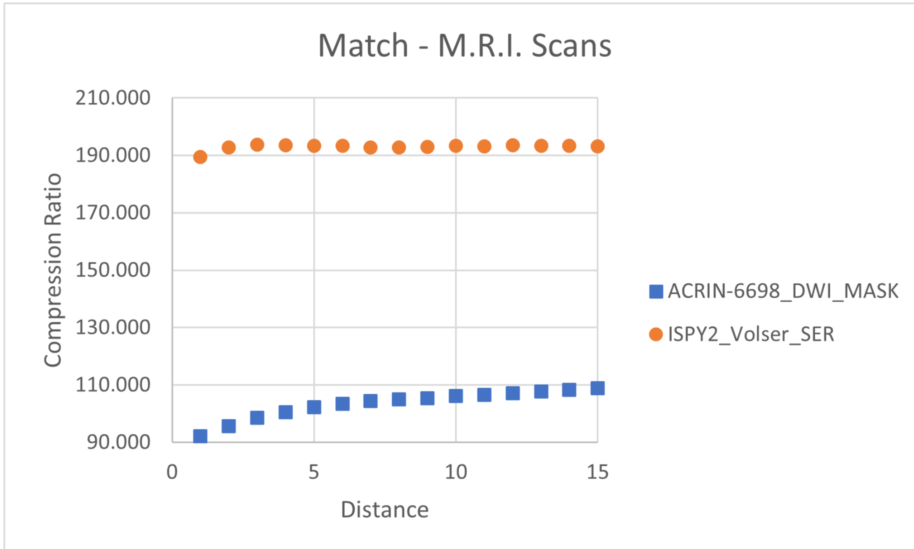

Figure 5.22: Match CRs with Varying Distance - M.R.I. Scans

Distance	ACRIN-6698_DWI_MASK	ISPY2_Volser_SER
1	92.156	189.457
2	95.696	192.793
3	98.622	193.808
4	100.516	193.636
5	102.309	193.333
6	103.420	193.265
7	104.430	192.797
8	104.939	192.781
9	105.419	192.890
10	106.128	193.344
11	106.637	193.246
12	107.138	193.464
13	107.707	193.399
14	108.274	193.312
15	**108.843**	**193.133**

Table 5.24: Match CRs with Varying Distance - M.R.I. Scans

by an impressive 39.268% while the increase for ISPY2_Volser_SET is only 7.389%. ACRIN-66698_DWI_MASK not only results in the largest percent increase, but it also is the dataset with the second largest compression ratios. The largest compression ratios come from the only other dataset where Match outperforms CALIC,

Method	ACRIN-6698_DWI_MASK	ISPY2_Volser_SER
CALIC	78.154	180.473
Match	108.843	193.808
Percent Difference	39.268	7.389

Table 5.25: CALIC's CR Compared to Match's CR - M.R.I. Scans

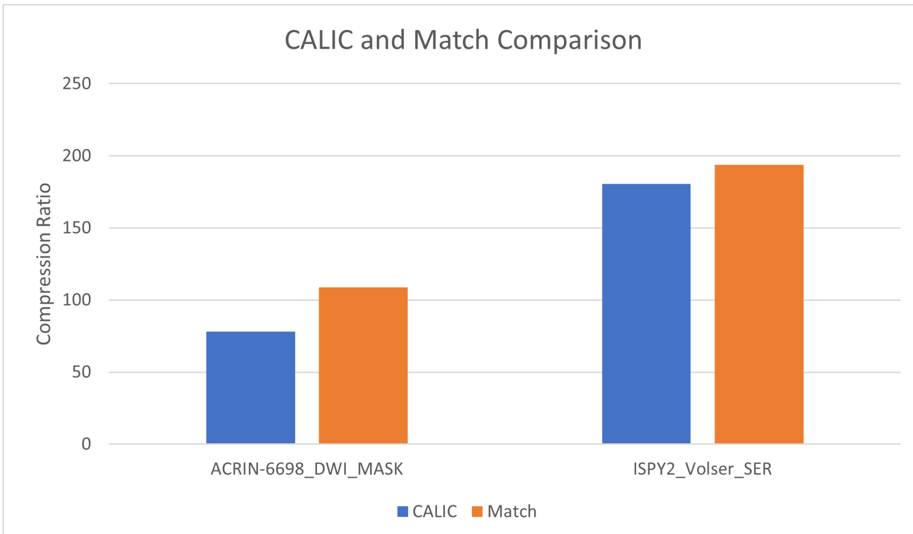

Figure 5.23: CALIC's CR Compared to Match's CR - M.R.I. Scans

ISPY2_Volser_SER, with an astounding ratio of 193.808. Six of these datasets result in a percent difference between Match and CALIC less than |5| %: ACRIN-6698_ADC, ISPY2_multiphase384, ISPY2_VOLSER_DCE, ISPY2_VOLSER_PE2, ISPY2_VOLSER_PE6, and ISPY2_Water_T2fseidealarc_BP. This leaves nine datasets where CALIC outperformed Match: ACRIN-6698_4bval, ACRIN-6698_DWI_TRACE, ISPY2_3_Plane_Scout, ISPY2_Fat_T2fseidealarc_BP, ISPY2_Fieldmap, ISPY2_IP_T2fseidealarc_BP, ISPY2_OP_T2fseidealarc_BP, ISPY2_T2fseidealarc_BP, and ISPY2_WATER_T2_fseidealarc_BP. Therefore, Match performs better than CALIC for 11.765% of the M.R.I. scans, nearly the same as CALIC for 35.294% of the M.R.I. scans, and worse than CALIC for 52.941%. of the M.R.I. scans

5.7 Resolution

We've seen the results for how the distance affects Match as well as how effective of a compression technique it is compared to CALIC. These results were explored for various videos at different resolutions. Figure 5.24 illustrates the best compression ratios for each of the 22 video datasets. In general, as resolution increases, the resulting compression ratio decreases.

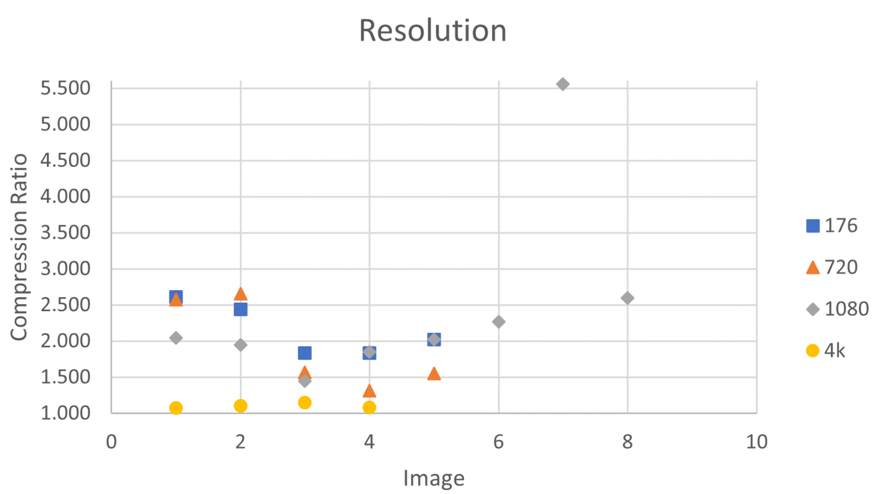

Figure 5.24: Match CR For Each Video Dataset

Looking at each of these various videos at different resolutions may give us some idea of how Match is affected by resolution, but there are other variables changing between the datasets as each video is different. Therefore, to determine how Match is affected by resolution, four datasets from [37] were tested. Each dataset includes videos at dimensions [720, 1280, 3], [1080, 1920, 3], and [2160, 3840, 3]. These resolutions will be referred to as 720, 1080, and 2160 respectively. The first six frames of each video were compressed using Match with the distance varying from one to twenty pixels and the compression ratios are examined.

The first dataset is Ducks_Take_Off, where the best compression ratio for the smallest resolution was found to be 1.496. This compression ratio is slightly better than the best compression ratio that was determined for the 1080 resolution, which is 1.468. At resolution 2160, the best compression ratio is 1.391, which is worse than the other two resolutions. The 720 resolution is 4.110% better than the 1080 and 7.549% better than the 2160. The 1080 resolution is 5.536% better than the 2160 compression. Figure 5.25 illustrates the compression ratios that are in Table 5.26. For each distance, the 720 resolution results in the best compression while 2160 results in the worst compression. Therefore, the trend found with Ducks_Take_Off is that as resolution increases, Match's performance decreases.

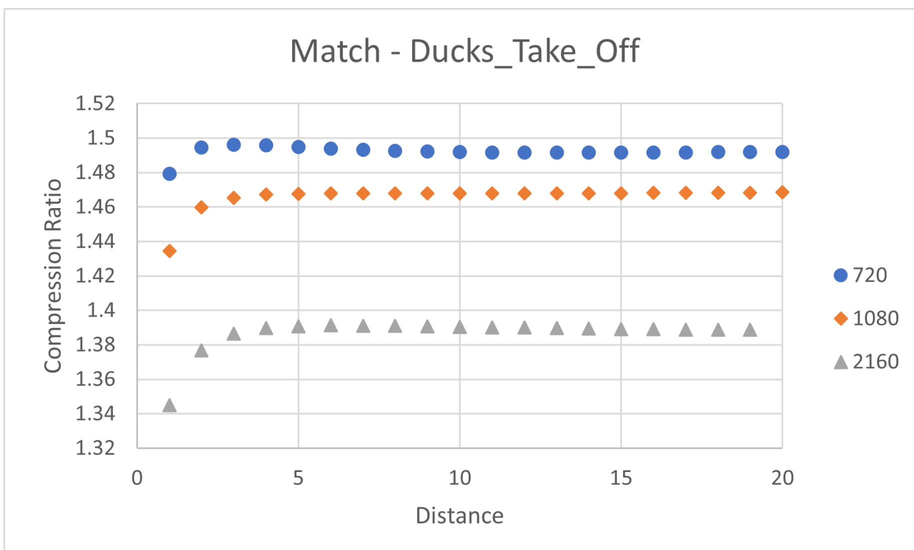

Figure 5.25: Match CRs with Varying Distance - Ducks_Take_Off

Distance	720	1080	2160
1	1.479	1.435	1.345
2	1.495	1.460	1.377
3	1.496	1.465	1.387
4	1.496	1.467	1.390
5	1.495	1.468	1.391
6	1.494	1.468	1.391
7	1.493	1.468	1.391
8	1.493	1.468	1.391
9	1.162	1.468	1.391
10	1.492	1.468	1.390
11	1.492	1.468	1.390
12	1.492	1.468	1.390
13	1.492	1.468	1.390
14	1.492	1.468	1.389
15	1.492	1.468	1.389
16	1.492	1.468	1.389
17	1.492	1.468	1.389
18	1.492	1.468	1.389
19	1.492	1.468	1.389
20	1.492	1.468	1.389

Table 5.26: Match CRs with Varying Distance - Ducks_Take_Off

The second dataset is In_To_Tree. Unlike Ducks_Take_Off, it was determined that the best compression results from the 1080 resolution with a ratio of 1.586. The best compression for 720 and 2160 are 1.564 and 1.555 respectively. Therefore, the 1080 resolution is 1.407% better than the 720 resolution and 1.994% better than the 2160 resolution. The 720 resolution is 0.579% better than the 2160 resolution. Figure 5.26 illustrates the compression ratios for Table 5.27. For each distance, the 1080 resolution performs the best, however 720 performs better for shorter distances compared to the 2160, but for larger distance the 2160 performs better.

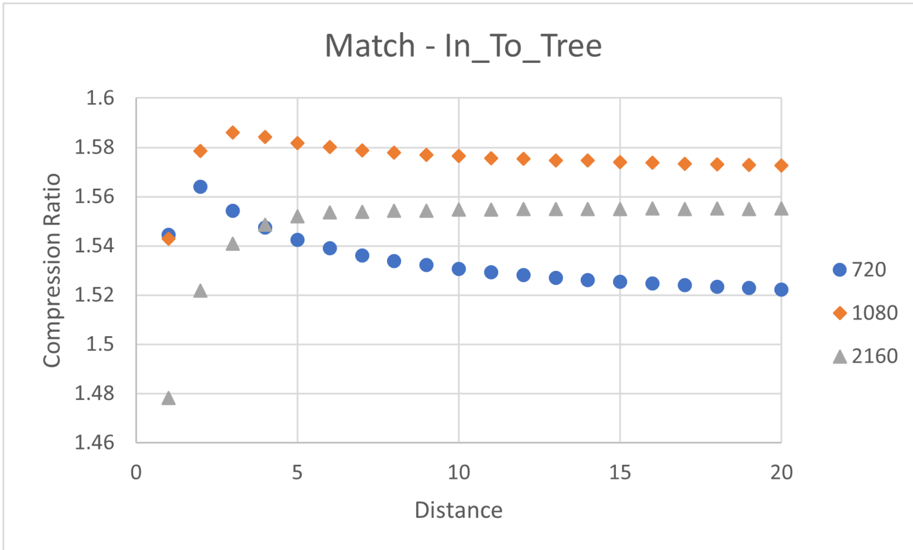

Figure 5.26: Match CRs with Varying Distance - In_To_Tree

Distance	720	1080	2160
1	1.544	1.543	1.478
2	1.564	1.578	1.522
3	1.554	1.586	1.541
4	1.547	1.584	1.549
5	1.543	1.582	1.552
6	1.539	1.580	1.553
7	1.536	1.579	1.554
8	1.534	1.578	1.554
9	1.532	1.577	1.554
10	1.531	1.576	1.555
11	1.523	1.576	1.555
12	1.528	1.575	1.555
13	1.527	1.575	1.555
14	1.526	1.575	1.555
15	1.525	1.574	1.555
16	1.525	1.574	1.555
17	1.524	1.573	1.555
18	1.523	1.573	1.555
19	1.528	1.573	1.555
20	1.522	1.573	1.555

Table 5.27: Match CRs with Varying Distance - In_To_Tree

Old_Town_Cross is similar to the results from In_To_Tree where the best compression ratio is given with 1080 resolution. The best compression ratio for 720, 1080, and 2160 resolutions were found to be 1.517, 1.547, and 1.444 respectively. The 1080 resolution is 1.978% better than the 720 resolution and 7.133% better than the 2160 resolution. The best compression ratio for resolution 720 is 5.055% better than the compression ratio for resolution 2160. Figure 5.27 illustrates the compression ratio for each resolution, the exact values are in Table 5.28. For each distance, the 1080 resolution has the best compression while the 2160 has the worst.

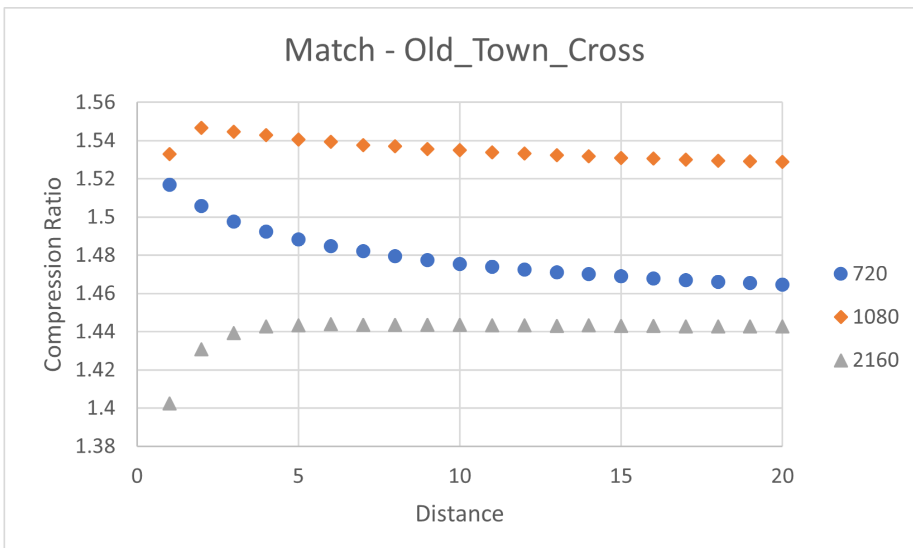

Figure 5.27: Match CRs with Varying Distance - Old_Town_Cross

Distance	720	1080	2160
1	1.517	1.533	1.403
2	1.506	1.547	1.431
3	1.498	1.545	1.439
4	1.492	1.543	1.443
5	1.488	1.541	1.443
6	1.485	1.539	1.444
7	1.482	1.538	1.444
8	1.480	1.537	1.444
9	1.477	1.536	1.444
10	1.476	1.535	1.444
11	1.474	1.534	1.443
12	1.472	1.533	1.443
13	1.471	1.532	1.443
14	1.470	1.532	1.443
15	1.469	1.531	1.443
16	1.468	1.531	1.443
17	1.467	1.530	1.443
18	1.466	1.529	1.443
19	1.466	1.529	1.443
20	1.465	1.529	1.443

Table 5.28: Match CRs with Varying Distance - Old_Town_Cross

Unlike the other datasets, Park_Joy has the best compression ratio for resolution 2160 at 1.613. The 720 resolution has the best compression ratio at 1.482 while the best compression ratio for resolution 1080 is 1.528. The compression ratio for 2160 is 8.839% larger than the ratio for resolution 720 and 5.563% larger than the ratio for resolution 1080. The best compression ratio for resolution 1080 is 3.104% larger than the ratio for resolution 720. Therefore, the trend for Park_Joy is as the resolution increases Match's performance increases. Figure 5.28 illustrates the compression ratios for the varying resolutions that are found in Table 5.29.

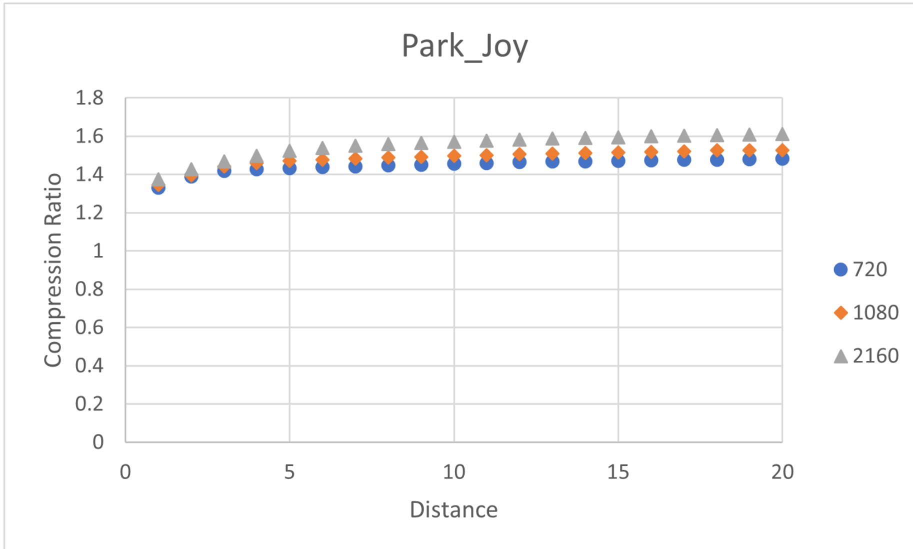

Figure 5.28: Match CRs with Varying Distance - Park_Joy

Distance	720	1080	2160
1	1.332	1.347	1.374
2	1.390	1.396	1.428
3	1.420	1.442	1.469
4	1.427	1.460	1.498
5	1.433	1.471	1.524
6	1.439	1.477	1.538
7	1.444	1.483	1.550
8	1.448	1.488	1.558
9	1.452	1.493	1.565
10	1.457	1.497	1.572
11	1.461	1.501	1.577
12	1.465	1.505	1.582
13	1.467	1.509	1.587
14	1.469	1.512	1.591
15	1.472	1.515	1.595
16	1.474	1.579	1.599
17	1.476	1.521	1.603
18	1.478	1.526	1.607
19	1.480	1.526	1.610
20	1.482	1.528	1.613

Table 5.29: Match CRs with Varying Distance - Park_Joy

The trend from Ducks_Take_Off was determined to be as resolution increases, Match's performance decreases. This is the exact opposite of the trend found in Park_Joy. In_To_Tree and Old_Town_Cross, on the other hand had the same trend where the middle resolution performed the best and the largest resolution performed the worst. Therefore, there is no clear conclusion on how Match is affected by resolution.

5.8 Frame Rate

Only one dataset was found to see how Match is affected by the frame rate. It is assumed that the faster the frame rate, the more similar the frames are to one another and therefore the better Match will do. These two datasets are from the 176 resolution videos, they are Claire and Claire(6fps). Claire is the original video while Claire(6fps) is the video with a 6Hz frame rate that is obtained by skipping 5 frames. Table 5.30 compares the compression ratio of Match for both videos as the distance varies from one to ten. For every distance, Match results in a larger compression ratio for Claire over Claire(6fps). The far right column is the percent difference between the compression ratios of the two videos, which was calculated using equation 5.2. However, since there's only one dataset, it can only be assumed that as frame rate increases the compression ratio of Match will also increase.

$$PercentDifference = 100\left(\frac{CR_{Claire} - CR_{Claire(6fps)}}{CR_{Claire}}\right) \tag{5.2}$$

Distance	Claire	Claire (6fps)	Percent Difference
1	2.571	2.348	8.682
2	2.606	2.414	7.401
3	2.590	2.436	5.970
4	2.581	2.438	5.519
5	2.568	2.431	5.313
6	2.567	2.428	5.403
7	2.551	2.421	5.095
8	2.547	2.418	5.048
9	2.539	2.415	4.892
10	2.535	2.413	4.822

Table 5.30: Match CRs with Varying Distance

In this chapter, we went through presenting the compression results of Match for each of the datasets. They were presented for 22 videos that were organized by their resolution. In general, for the 176 resolution images, Match outperformed CALIC. However, the one dataset where Match didn't have a larger compression ratio - Miss_Am - the percent difference between the two methods was negligible and thus it can be said that Match performed at least as well as CALIC for these videos. The same can be said for the 720 resolution videos; however, for two of these datasets the difference between the resulting compression ratio is minimal. The 1080 resolution is when you can really see that Match does have weaknesses. In these datasets, Match underperforms CALIC by over 25% for Crowd_Run; however Match did outperform CALIC by over 38% for another one of these datasets, Life. The final resolution of these video datasets is 4k. These datasets exploited Match's computational complexity and took time to run. Not only did it take a lot of time, but the percent differences for all but Narrator are negligible. Even Narrator's resulting compression ratio with Match is only 5% better and thus Match isn't worth using at this resolution.

Once the results of the videos were discussed, the results of the CT scans were examined. Overall, the resulting compression ratio of Match is unimpressive, especially when compared to CALIC. For the most part, the compression difference is so small it's negligible, therefore the compression data for all of the CT scans is located in Appendix A. However, we examined the CT scans that have a percent difference greater than $|5|\,\%$. The largest percent increase in compression ratio was found to be 20% while the largest percent decease was -17%. FLike the Similarly, the results of the MRI scans are also unimpressive. Out of the 17 datasets, Match results in a significantly larger compression ratio for only two of these datasets. The first of these datasets is ACRIN-6698_DWI_MASK has an impressive compression ratio increase of 39% while the second, ISPY2_Volser_SET, is only 7% better. Out of the remaining 15 MRI scans, CALIC outperformed Match for nine of these datasets and Match performed the same as CALIC for the remaining six datasets.

Once the results of Match were examined for each of the various datasets, we looked at how resolution and frame rate affect match. Unfortunately, no clear conclusions could be made for either. When looking at how Match performs for the 22 videos of varying resolution, one would believe that as resolution increased, the compression ratio would decrease. However, when we looked at how Match performed for different resolutions of the same video we found three different trends out of the four datasets. For frame rate, on the other hand, we only had one dataset with varying frame rate but the same resolution. The assumption is that as frame rate increases, Match's compression ratio increases as well. Logically this makes sense as a higher frame rate results in less motion and thus less differences between each frame. However, we only have the one dataset so this cannot be conclusively stated.

Now that we've seen how Match performs for each of the datasets, we need to predict which method should be used on which dataset. This needs to be done for various reasons, such as Match sometimes outperforms CALIC and sometimes does not. Not only that, but we need to consider if the computational complexity of Match is worth the resulting compression. Therefore, we now will look at the structural similarity and the edge quality measurements to see how well they work at selecting the right method to use for each video and medical scan.

Chapter 6: Determining Which Method to Use

Looking at the compression results that were just discussed, there are times when Match outperforms CALIC, underperforms CALIC, and performs the same as CALIC. Not only that, but no clear conclusion can be made for how resolution and frame rate affect Match, thus they cannot be used to predict which method is best to use. Match is also a computationally complicated algorithm, therefore it can take a lot of time for the algorithm to run. Therefore, a way to determine if not only the computational complexity is worth the wait, but if CALIC would result in a better compression ratio. It was determined that calculating the structural similarity, SSIM, and the edge quality between each frame gives a good prediction on which method will provide the larger compression ratio.

6.1 Structural Similarity

The structural similarity, SSIM, was calculated between each frame of the video sequences as well as each slide of the medical images. The SSIM has a value between zero and one, where the closer to one the SSIM is, the stronger the similarity is between the frames. In general, the stronger the similarity is between frames, the better

Match performs and the larger the compression ratio is. This trend is clearly illustrated in Figure 6.1 which plots the average SSIM compared to Match's compression ratio. Note that the maximum compression ratio in this graph is 20, which excludes the two datasets that resulted in compression ratios greater than 100. These two datasets are ACRIN-6698_DWI_MASK and ISPY2_Volser_SER with compression ratios of 108.843 and 193.808 respectively. The calculated average SSIM was 0.996 for ACRIN-6698_DWI_MASK and 0.997 for ISPY2_Volser_SER, which follows the trend.

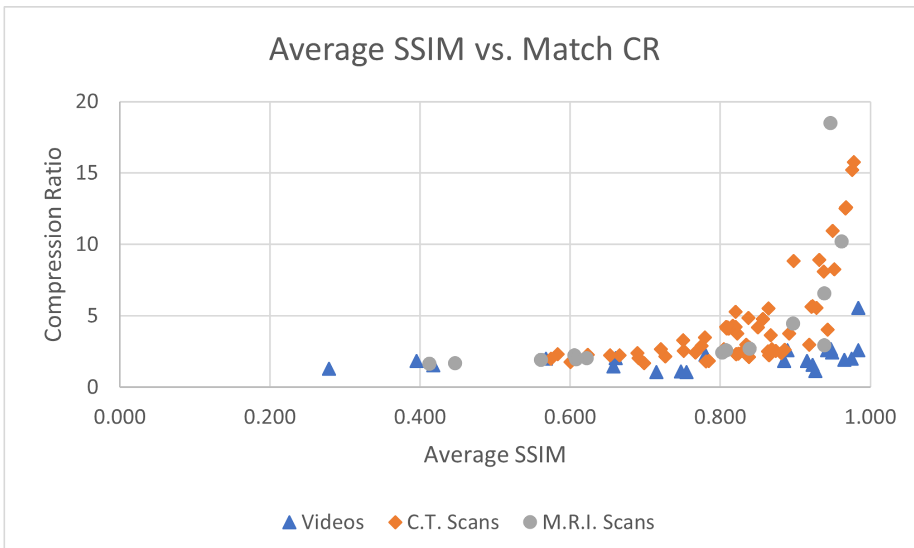

Figure 6.1: Average SSIM Compared to Match's CR

Due to Match's compression ratio increasing as the average SSIM increases, the percent difference between Match and CALIC follows the same trend; where the larger the SSIM, the larger the percent increase. Therefore, plotting the percent difference calculated between the two methods against the average SSIM results in Figure 6.2. Between approximately 0.800 and 1.000 the majority of the points are clustered between 0% and 50%.

Table 6.1 contains the SSIM measurement between frames for the various video datasets with the far right column being the average SSIM. If the percent difference

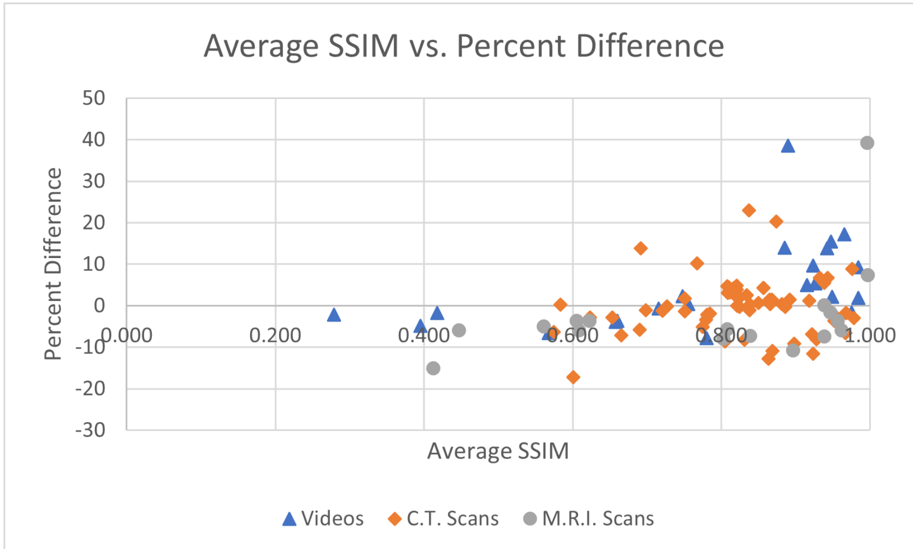

Figure 6.2: Average SSIM Compared to The Percent Difference Between Match and CALIC

between Match and CALIC is less than |5| %, then the resulting compression ratios are similar enough that it doesn't matter which compression method is used. A threshold of 0.820 is applied to determine if Match or CALIC should be used, where an average SSIM greater than 0.820 results in a prediction of Match.

All of the 176 videos have average SSIM measurements greater than 0.820, therefore the predicted method to use is Match. Since all but two of the 176 videos have percent increases greater than |5| %, Claire (6fps) and Miss_Am, the SSIM predictions are accurate. Out of these videos, Foreman was found to have the smallest average SSIM of 0.885, however Foreman had the highest compression ratio increase of 13.971%. The average SSIM for Miss_Am was found to be 0.975, which would lead one to believe that Match would outperform CALIC. However, Miss_Am was the only dataset that underperformed with a percent decrease of -1.560% when compared to CALIC. Thankfully, due to such a small percent difference, either method would suffice in compressing this video.

Match outperformed CALIC for three of the 720 datasets: Johnny, KristenAnd-

Sara, and Mobcal with percent increases of 14.677%, 15.442%, and 9.684% respectively. Their corresponding average SSIM measurements were calculated to be 0.942, 0.947, and 0.923. On the other hand, Mobcal and Parkrun had minor percent decreases of -2.096% and -1.776% when compressed with Match. Their average SSIM measurements were determined to be 0.279 and 0.418 respectively. Since they are less than 0.820, the method predicted to perform the best is CALIC and due to the percent differences being less than |5| %, the predicted method is accurate.

Only two of the 1080 datasets, Controlled_Burn and Life, resulted in large percent increases of 17.239% and 38.581% respectively. Their average SSIM measurements were determined to be 0.965 and 0.890, therefore Match is accurately predicted to be the preferred method. For two of the datasets: Station2, and Aspen, CALIC slightly outperforms match with percent differences of -6.602%, and -7.774% respectively. Their corresponding averages SSIMs were calculated to be 0.568, and 0.780, which is less than 0.820 and therefore CALIC is accurately predicted to be the better compression method for these videos. The remaining four 1080 datasets: Riverbed, Blue_Sky, Crowd_Run, and Dinner result in percent differences less than |5%| with values of -4.781%, -3.627%, -3.856%, and 1.852% respectively. Their corresponding average SSIM measurements are 0.395, 0.660, 0.658, and 0.984. Applying the 0.820 threshold results in Dinner being the only dataset where Match is the predicted method. Therefore, the average SSIM accurately predicts which method to use for all of the 1080 datasets.

In Table 6.1, all of the 4k videos are listed without the Netflix in their title. All but one of the 4k video datasets, Netflix_Narrator, results in percent differences that are less than |5%|. The percent differences found are 0.373%, 2.319%, and -0.645% for Netlfix_Boat, Netflix_BoxingPractice, and Netflix_Tango respectively. Their corresponding SSIM measurement were calculated to be 0.755, 0.747, and 0.715. Due to the average SSIM for these datasets being less than 0.820, the predicted method is CALIC. Netflix_Narrator, on the other hand, has an SSIM of 0.927 and has a percent difference of 5.314%; and thus Match is accurately predicted to be the preferred method. However, Match is computationally complicated and thus takes more time to run the higher the resolution. None of these percent differences are large enough to outweigh the run-time Match takes. Therefore, CALIC should be used to compress the 4k resolution videos as the run-time is nearly instantaneous.

The average SSIM results in 100% accuracy for the various video datasets that were collected.

Resolution	Dataset	1 - 0	2 - 1	3 - 2	4 - 3	5 - 4	Average
176	Claire	0.981	0.971	0.989	0.990	0.987	0.984
	Claire (6fps)	0.916	0.943	0.972	0.934	0.981	0.949
	Carphone	0.891	0.939	0.850	0.928	0.969	0.915
	Foreman	0.882	0.870	0.880	0.889	0.904	0.885
720	Miss_Am	0.973	0.975	0.976	0.976	0.975	0.975
	Johnny	1.000	0.929	0.927	0.926	0.929	0.942
	KristenAndSara	1.000	0.935	0.935	0.934	0.933	0.947
	Mobcal	0.921	0.932	0.914	0.916	0.934	0.923
1080	Parkrun	0.272	0.300	0.267	0.263	0.293	0.279
	Shields	0.423	0.425	0.417	0.412	0.412	0.418
	Blue_Sky	0.660	0.660	0.660	0.660	0.661	0.660
	Controlled_Burn	0.966	0.969	0.966	0.967	0.958	0.965
	Crowd_Run	0.604	0.655	0.683	0.685	0.663	0.658
	Riverbed	0.404	0.403	0.395	0.396	0.379	0.395
	Station2	0.563	0.566	0.566	0.570	0.573	0.568
	Aspen	0.757	0.784	0.763	0.787	0.808	0.780
	Dinner	0.993	0.990	0.982	0.977	0.978	0.984
	Life	0.894	0.890	0.889	0.889	0.888	0.890
4k	Boat	0.756	0.759	0.757	0.753	0.750	0.755
	BoxingPractice	0.749	0.748	0.748	0.746	0.746	0.747
	Narrator	0.926	0.927	0.927	0.927	0.926	0.927
	Tango	0.714	0.714	0.714	0.716	0.718	0.715

Table 6.1: SSIM of Video Datasets

The SSIM measurements of the various C.T. scans are split the same way as Appendix A, starting with the five miscellaneous datasets. Three of these five datasets, 4D-Lung, CT_FUSION, and WB_MAC_P690, have percent differences less than |5%| with values of 4.805%, 4.285%, and -1.785% respectively. Their corresponding average SSIM measurements were calculated to be 0.820, 0.85, and 0.968. Therefore, with their average SSIM measurements being greater than or equal to 0.820, Match is the selected method. Match outperforms CALIC by 13.895% for TCGA-38-4628; however, its average SSIM was calculated to be 0.691 which is less than 0.820. Therefore, CALIC would inaccurately be predicted to perform better. Similarly, CT_FUSION has an average SSIM measurement of 0.600 but Match underperformed CALIC by -17.227%; therefore the average SSIM accurately predicts that CALIC should be the method used to compress this dataset.

Dataset	1 - 0	2 - 1	3 - 2	4 - 3	5 - 4	Average
4D-Lung	0.811	0.817	0.824	0.825	0.825	0.820
TCGA-38-4628	0.698	0.690	0.684	0.694	0.690	0.691
NLST-LSS	0.470	0.899	0.542	0.545	0.546	0.600
CT_FUSION	0.859	0.858	0.859	0.856	0.850	0.857
WB_MAC_P690	0.945	0.963	0.974	0.977	0.980	0.968

Table 6.2: SSIM of Miscellaneous C.T. Scans

The largest percent difference between Match's compression ratio and CALIC's compression ratio for the LIDC-IRDI datasets was determined to be 1.789%. Since this increase is so small, predicting either method would result in nearly the same compression ratio. The SSIM measurements between slides as well as the average SSIM are found in Table 6.3. In numerical order of the datasets, the average SSIM was calculated to be 0.881, 0.726, 0.836, 0.824, 0.886, 0.918, 0.751, 0.865, 0.867, and 0.839. Any average SSIM greater than 0.800 should be compressed with Match; which results in all but LIDC-IRDI-0002 and LIDC-IRDI-0007 being compressed with Match.

Dataset	1 - 0	2 - 1	3 - 2	4 - 3	5 - 4	Average
LIDC-IRDI-0001	0.879	0.884	0.882	0.884	0.878	0.881
LIDC-IRDI-0002	0.727	0.715	0.726	0.728	0.736	0.726
LIDC-IRDI-0003	0.840	0.837	0.835	0.834	0.834	0.836
LIDC-IRDI-0004	0.818	0.827	0.825	0.822	0.829	0.824
LIDC-IRDI-0005	0.882	0.884	0.891	0.889	0.884	0.886
LIDC-IRDI-0006	0.913	0.921	0.920	0.920	0.917	0.918
LIDC-IRDI-0007	0.747	0.753	0.751	0.756	0.748	0.751
LIDC-IRDI-0008	0.871	0.866	0.862	0.862	0.866	0.865
LIDC-IRDI-0009	0.868	0.867	0.870	0.865	0.867	0.867
LIDC-IRDI-0010	0.837	0.841	0.839	0.839	0.838	0.839

Table 6.3: SSIM of LIDC-IRDI C.T. Scans

The SSIM between slides as well as the average SSIM for the seven various AMC C.T. scans with 36 individual datasets is in Table 6.4.

In AMC-001, datasets WB_MAC_P690, CT_FUSION, and Coronals have respective percent differences of -1.785%, 4.285% and 0.301%. Since the percent differences are so small, using either CALIC or Match will result in a nearly identical compression. These two datasets have average SSIM measurements of 0.968, 0.857, and 0.538 respectively. Due to the average SSIM for WB_MAC_P690 and CT_FUSION being greater than 0.820 it's predicted that Match should be the method chosen while CALIC would be chosen for Coronals. Using the average SSIM to predict which method to use for WB_NAC_P690 and CHST_1.25MM_SHARP, selects the wrong method. WB_NAC_P690 has an average SSIM of 0.923, which predicts Match would result in the larger compression ratio; however, Match under-performs CALIC by -11.587%. An average SSIM measurement of 0.767 was calculated for CHST_1.25MM_SHARP, thus CALIC is the method predicted to perform the best, which is incorrect as Match results in a percent increase of 10.159%.

AMC-002 contains two datasets, CHEST_1.0_B45f and coronals. Their average SSIM measurements were calculated to be 0.838 and 0.690 respectively. Since the average SSIM for CHEST_1.0_B45f is greater than 0.820, it's predicted that Match is the compression technique that results in better compression. The percent difference between Match and CALIC for this dataset was determined to be -1.064%, which is minimal and thus both methods result in nearly equivalent compression. On the other hand, coronals has an average SSIM that is less than 0.820 and therefore CALIC is the method that will result in the best compression ratio. This predictions is correct as CALIC slightly outperforms Match with a percent difference of -5.774%.

Seven datasets are found under AMC-003: CORONAL_MPR_2MM, CT_FUSION, CTAC, THORAX_LUNG_1MM, THORAX_LUNG_2MM, WB_MAC_P690, and WB_NAC_P690. For CORONAL_MPR_2MM, CALIC outperforms Match with a percent difference of -7.1224%. The average SSIM for this dataset was calculated to be 0.666 which reflects that CALIC is the method that results in the best compression as the average SSIM is less than 0.820. CT_FUSION compresses slightly better with Match while CTAC compresses better with Match with corresponding percent differences of 5.451% and 8.878% respectively. Both of these datasets have average SSIM measurements greater than 0.820 with values of 0.938 and 0.976. Therefore, for these three datasets, the average SSIM accurately predicts which method results in the best compression. The remaining datasets, however, all have minor percent differences with the largest being -3.579%. Due to this, it doesn't matter which method is chosen to compress these datasets. THORAX_LUNG_1MM, THORAX_LUNG_2MM, WB_MAC_P690, and WB_NAC_P690 average SSIM measurements were calculated to be 0.821, 0.821, 0.978, and 0.952 respectively. Since all of these values are greater than 0.820, Match would be the chosen compression method.

Similar to AMC-001, AMC-004 contains five datasets: coronals, CT_FUSION, THORAX_1.0_B45f, WB_MAC_P690, and WB_NAC_P690. Their corresponding average SSIM measurements were calculated to be 0.623, 0.823, 0.699, 0.967, and

0.928 respectively. The percent difference for the first three datasets, coronals, CT_FUSION, and THORAX_1.0_B45f are -2.834%, 1.703%, and -1.064%. These are all negligible differences, therefore it doesn't matter which method is used to compress these. The average SSIM would predict CALIC for the first and third while Match would be chosen for the second. WB_MAC_P690, and WB_NAC_P690, on the other hand, have more significant percent differences of -6.612% and -8.104%. Both of these datasets should be compressed with CALIC, however, their average SSIM measurements are both greater than 0.820 and thus the method would inaccurately be predicted.

Like the previous batch of datasets, AMC-005 contains five datasets: CHEST_1.0_B45f, CHEST_2.0_coronal, CHEST_2.0_Sagittal, CHEST_5.0_B31f, and CHEST_7.0_MIP_Axia. For CHEST_1.0_B45f and CHEST_5.0_B31f, either method could be chosen to result in the best compression as the percent differences found for these two datasets are -1.818% and -2.779% respectively. Their average SSIM measurements were found to be 0.784 and 0.653 which are both less than 0.820 and therefore CALIC would be the method that's predicted to have the larger compression ratio. CHEST_2.0_coronal and CHEST_2.0_Sagittal both have average SSIM measurements greater than 0.820 with values 0.863 and 0.869 respectively. Therefore, Match would be incorrectly chosen to predict these two datasets as Match under performs CALIC by -12.838% and -10.935-6% respectively. The only dataset in this series that results in a correct prediction that Match should be used to compress the sequence is CHEST_7.0_MIP_Axia with an average SSIM measurement of 0.943. For this dataset, Match slightly outperformed CALIC with a percent increase of 6.725%. Out of these five datasets, two were incorrectly predicted.

AMC-006 has seven datasets like AMC-003. These datasets consist of AP_1MM, CORONAL_AP, CT_images, LUNG_1MM, PET_BODY_CTAC, PET_BODY_NO_AC, and ST_CAP_5MM. Between AP_1MM, PET_BODY_CTAC, and ST_CAP_5MM, the largest percent difference is -2.083%. The corresponding SSIM measurements for these three datasets was calculated to be 0.892, 0.950, and 0.850 respectively. Since all of these average SSIM measurements are greater than 0.820, Match would accurately be the chosen compression method due to the negligible differences between the two methods. Similarly, CORONAL_AP and PET_BODY_NO_AC have average SSIM measurements of 0.897 and 0.922 and therefore Match would also be chosen. However, Match under performs CALIC for these two datasets by -9.179% and -6.828% respectively. Therefore these two datasets are inaccurately predicted. For CT_images and LUNG_1MM Match would also be the predicted method as they have average SSIM measurements of 0.932 and 0.874 respectively. However, the prediction for these two are accurate as Match outperforms CALIC by 20.365% and 6.758%.

Like many of the other AMC sets, AMC-007 also contains five datasets consisting of THORAX_1.0_B45f, Thorax_2.0_SPO_cor, Thorax_2.0_SPO_sag, Thorax_5.0_B31f and Thorax_7.0_MIP_ax. Three of these five datasets, THORAX_1.0_B45f, Thorax_5.0_B31f and Thorax_7.0_MIP_ax, have small percent differences of -2.097%, -1.183%, and 2.506% respectively. Due to the small percent error, it doesn't matter which method the average SSIM predicts. In the same order, the average SSIM was calculated to be 0.781, 0.721, and 0.834. CALIC is the chosen method for the first two datasets while Match is selected for the third. The average SSIM for Thorax_2.0_SPO_cor is 0.805 and the average SSIM for Thorax_2.0_SPO_sag was calculated to be 0.831; sticking to the same threshold of 0.820, the average SSIM accurately predicts that CALIC will result in a higher compression ratio for the first dataset but inaccurately predicts that Match will result in larger compression for the second, where CALIC outperforms Match for each of these datasets by -8.571% and -8.217% respectively.

	Dataset	1-0	2-1	3-2	4-3	5-4	Average
1	WB_NAC_P690	0.847	0.916	0.941	0.953	0.958	0.923
	WB_MAC_P690	0.945	0.963	0.974	0.977	0.980	0.968
	CT_FUSION	0.859	0.858	0.859	0.856	0.850	0.857
	coronals	0.563	0.568	0.586	0.596	0.603	0.583
	CHST_1.25MM_SHARP	0.770	0.766	0.767	0.768	0.765	0.767
2	CHEST_1.0_B45f	0.834	0.842	0.836	0.836	0.843	0.838
	coronals	0.699	0.694	0.687	0.683	0.685	0.690
3	CORONAL_MPR_2MM	0.687	0.690	0.675	0.645	0.631	0.666
	CT_FUSION	0.939	0.940	0.940	0.937	0.934	0.938
	CTAC	0.976	0.976	0.977	0.975	0.974	0.976
	THORAX_LUNG_1MM	0.818	0.819	0.827	0.821	0.819	0.821
	THORAX_LUNG_2MM	0.818	0.819	0.827	0.821	0.819	0.821
	WB_MAC_P690	0.975	0.965	0.981	0.983	0.986	0.978
	WB_NAC_P690	0.898	0.940	0.956	1.000	0.964	0.952
4	coronals	0.632	0.629	0.622	0.618	0.616	0.623
	CT_FUSION	0.833	0.829	0.822	0.817	0.812	0.823
	THORAX_1.0_B45f	0.685	0.703	0.701	0.701	0.702	0.699
	WB_MAC_P690	0.940	0.963	0.974	0.978	0.981	0.967
	WB_NAC_P690	0.856	0.918	0.947	0.954	0.964	0.928
5	CHEST_1.0_B45f	0.785	0.788	0.782	0.781	0.786	0.784
	CHEST_2.0_coronal	0.890	0.870	0.861	0.852	0.844	0.863
	CHEST_2.0_Sagittal	0.870	0.870	0.873	0.867	0.865	0.869
	CHEST_5.0_B31f	0.658	0.658	0.660	0.643	0.648	0.653
	CHEST_7.0_MIP_Axia	0.941	0.940	0.942	0.945	0.946	0.943
6	AP_1MM	0.891	0.893	0.893	0.890	0.890	0.892
	CORONAL_AP	0.954	0.925	0.896	0.871	0.841	0.897
	CT_images	0.934	0.933	0.934	0.932	0.929	0.932
	LUNG_1MM	0.844	0.838	0.842	1.000	0.847	0.874
	PET_BODY_CTAC	0.937	0.949	0.953	0.956	0.955	0.950
	PET_BODY_NO_AC	0.902	0.920	0.926	0.931	0.930	0.922
	ST_CAP_5MM	0.855	0.851	0.850	0.843	0.850	0.850
7	THORAX_1.0_B45f	0.776	0.783	0.783	0.782	0.780	0.781
	Thorax_2.0_SPO_cor	0.808	0.802	0.805	0.807	0.804	0.805
	Thorax_2.0_SP_sag	0.833	0.834	0.830	0.829	0.828	0.831
	Thorax_5.0_B31f	0.726	0.720	0.724	0.723	0.711	0.721
	Thorax_7.0_MIP_ax	0.826	0.845	0.833	0.836	0.832	0.834

Table 6.4: SSIM of AMC C.T. Scans

The SSIM measurements for the ten datasets in 4D-Lung are in Table 6.6. For only one of these datasets, 60, Match outperformed CALIC with a percent increase of 22.975%. The average SSIM for this dataset was calculated to be 0.837 which is greater than 0.800 and therefore would accurately predict that Match results in a larger compression ratio. The remaining C.T. scans in this set result in minor percent differences between the two methods; and thus either method predicted would be an accurate prediction. In order from left to right excluding dataset 60, the average SSIM measurements were calculated to be 0.817, 0.821, 0.819, 0.812, 0.809, 0.811, 0.779, 0.810, and 0.807. Of these datasets with negligible differences, the only scan with an SSIM measurement greater than 0.820 is 10 and is the only Match predicted dataset in this batch; the remainder will be compressed with CALIC.

Dataset	1-0	2-1	3-2	4-3	5-4	Average
0	0.821	0.812	0.813	0.819	0.817	0.817
10	0.826	0.822	0.819	0.819	0.819	0.821
20	0.821	0.819	0.817	0.820	0.817	0.819
30	0.814	0.811	0.812	0.813	0.810	0.812
40	0.810	0.806	0.808	0.809	0.811	0.809
50	0.812	0.813	0.809	0.811	0.809	0.811
60	0.812	0.808	0.759	1.000	0.807	0.837
70	0.813	0.810	0.646	0.814	0.814	0.779
80	0.818	0.808	0.808	0.808	0.808	0.810
90	0.814	0.803	0.807	0.809	0.805	0.807

Table 6.5: SSIM of 4D-Lung C.T. Scans

Table 6.6 contains the SSIM measurements for the C.T. scans under CMB-CRC-MSB-02381, which consist of datasets Body_5.000CE_1, Body_5.000CE_2, Body_5.0CE_1, and Body_5.0CE_2. The first two datasets, Body_5.000CE_1 and Body_5.000CE_2, result in Match slightly under-performing CALIC with percent differences of -6.317% and -5.100% respectively. The average SSIM for these two datasets was calculated to be 0.575 and 0.775, which accurately predicts CALIC is the better compression method with the threshold of 0.820. The remaining two datasest, Body_5.0CE_1 and Body_5.0CE_2, both have minor percent differences of 1.486% and -1.295% respectively. Their average SSIM measurements were calculated to be 0.864 and 0.750, which leads to a prediction of Match for Body_5.0CE_1 and a prediction of CALIC for Body_5.0CE_2. Due to the small percent differences, both methods result in near identical compressions, so the prediction can't be wrong.

Dataset	1-0	2-1	3-2	4-3	5-4	Average
Body_5.000CE_1	0.562	0.583	0.586	0.571	0.572	0.575
Body_5.000CE_2	0.776	0.786	0.778	0.776	0.760	0.775
Body_5.0CE_1	0.880	0.875	0.866	0.855	0.847	0.864
Body_5.0_CE_2	0.742	0.744	0.755	0.753	0.758	0.750

Table 6.6: SSIM of CMB-CRC-MSB-02381 C.T. Scans

The SSIM measurements for the 17 M.R.I. datasets are in Table 6.7. Six of these datasets, ACRIS-6698_ADC, ISPY2_multiphase384, ISPY2_VOLSER_DCE, ISPY2_VOLSER_PE2, ISPY2_VOLSER_PE6, and ISPY2_Water_T2fseidealarc_BP, have minor percent differences between the two compression methods of 0.136%, -3.592%, -3.626%, -1.623%, -3.565%, and -4.933%. Therefore, no matter which method the average SSIM predicts, it will be correct. The average SSIM was calculated for each of these datasets to be 0.938, 0.622, 0.606, 0.947, 0.957, and 0.561 respectively. For the threshold, 0.820, ACRIS-6698_ADC, ISPY2_multiphase384, ISPY2_VOLSER_DCE, and ISPY2_Water_T2fseidealarc_BP would all be selected to be compressed with CALIC. For the remaining two datasets, ISPY2_VOLSER_PE2, and ISPY2_VOLSER_PE6, Match is be the selected compression method.

The average SSIM inaccurately predicts that Match should result in a larger compression ratio than CALIC for four datasets. These four datasets are ACRIN-6698_4bval with an average SSIM of 0.938, ACRIN-6698_DWI_TRACE with an average SSIM of 0.961, an average SSIM value of 0.897 for ISPY2_3_Plane_Scout, and

ISPY2_Fat_T2fseidealarc_BP with an average SSIM of 0.839. Match under-performed for these datasets with percent decreases of -7.346%, -5.948%, -10.691%, and -7.303% respectively.

On the other hand, the average SSIM accurately predicts seven of these datasets: ACRIN-6698_DWI_MASK, ISPY2_Fieldmap, ISPY2_IP_T2fseidealarc_BP, ISPY2_OP_T2fseidealarc_BP, ISPY2_T2fseidealarc_BP, ISPY2_Volser_SER, and ISPY2_WATER_T2_fseidealarc_BP. In order, these datasets have average SSIM measurements of 0.996, 0.412, 0.802, 0.808, 0.447, 0.997, and 0.608 and percent differences of 39.267%, -14.997%, -7.888%, -5645%, -5.896%, 7.389%, and -5.896%. Applying the 0.820 threshold results in Match being the prediction for ACRIN-6698_DWI_MASK and ISPY2_Volser_SER and CALIC being the prediction for the remainder of the datasets.

Dataset	1-0	2-1	3-2	4-3	5-4	Average
ACRIN-6698_4bval	0.941	0.940	0.938	0.936	0.935	0.938
ACRIS-6698_ADC	0.941	0.940	0.938	0.936	0.935	0.938
ACRIN-6698_DWI_MASK	0.999	0.997	0.996	0.993	0.994	0.996
ACRIN-6698_DWI_TRACE	0.969	0.967	0.957	0.958	0.956	0.961
ISPY2_3_Plane_Scout	0.908	0.908	0.904	0.890	0.873	0.897
ISPY2_Fat_T2fseidealarc_BP	0.853	0.850	0.844	0.829	0.818	0.839
ISPY2_Fieldmap	0.404	0.392	0.391	0.436	0.439	0.412
ISPY2_IP_T2fseidealarc_BP	0.826	0.815	0.801	0.792	0.778	0.802
ISPY2_multiphase384	0.615	0.617	0.622	0.626	0.632	0.622
ISPY2_OP_T2fseidealarc_BP	0.831	0.824	0.807	0.797	0.781	0.808
ISPY2_T2fseidealarc_BP	0.453	0.466	0.455	0.414	0.446	0.447
ISPY2_VOLSER_DCE	0.601	0.607	0.608	0.602	0.610	0.606
ISPY2_VOLSER_PE2	0.969	0.961	0.946	0.931	0.925	0.947
ISPY2_VOLSER_PE6	0.980	0.971	0.959	0.943	0.931	0.957
ISPY2_Volser_SER	1.000	0.999	0.998	0.995	0.992	0.997
ISPY2_WATER_T2_fseidealarc_BP	0.628	0.620	0.603	0.602	0.588	0.608
ISPY2_Water_T2fseidealarc_BP	0.580	0.571	0.558	0.557	0.539	0.561

Table 6.7: SSIM of M.R.I. Scans

If the threshold for determining which method to use based on the average SSIM is set to 0.820, then fourteen of the 104 datasets are inaccurately predicted, resulting in an accuracy of 86.539%. The threshold remained at 0.820 for the medical images as there's a lot of correlation between the slides in the scans as the structural similarity considers luminance, contrast, and structure when calculated how alike two frames are.

6.2 Edge Stability

For each frame in each dataset, the edges were found and the mean squared error was calculated between the edge images giving the edge quality measurements. Once the measurement was calculated between each frame, the average was taken. Similar to the SSIM measurement, the edge quality measurement falls in a range between zero and one; however, unlike the SSIM, the closer the measurement is to zero, the better the edge quality. Figure 6.3 illustrates the average edge stability measurement compared to Match's compression ratio, where visually looks as though there is no clear trend. However, looking strictly at the average edge quality measurement for strictly the video datasets, the better the edge stability, the smaller the value and the better Match is going to perform. On the other hand, the C.T. datasets result in larger compression ratios spaced from 0.100 and 0.600. Similar to the C.T. scans, the M.R.I. datasets also have large compression ratios, but between the range 0.300 and 0.600. Therefore, a different prediction threshold is necessary. Like with the SSIM measurement, the maximum compression ratio in this graph is 20, which excludes the two datasets that resulted in compression ratios greater than 100. These two datasets are ACRIN-6698_DWI_MASK and ISPY2_Volser_SER with compression ratios of 108.843 and 193.808 respectively. The average edge stability measurement was calculated to be 0.086 for ACRIN-6698_DWI_MASK and 0.422 for ISPY2_Volser_SER; where the first dataset is the only medical one that follows the clear trend of the videos.

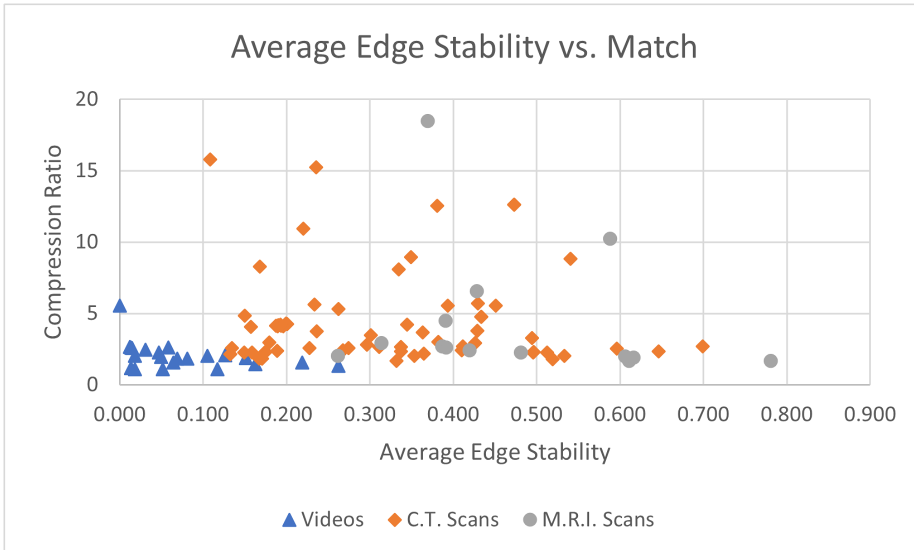

Figure 6.3: Average Edge Stability Measurement Compared to Match's CR

Comparing the average edge stability measurement to the percent difference between CALIC and Match results in the plot in Figure 6.4. Similar to the comparison between the measurement and the compression ratio, the closer to zero the edge stability is, the larger the percent difference. However, this trend is clearly seen in the video datasets and the single medical dataset that follows the trend from the comparison with the compression ratio, ACRIN-6698_DWI_MASK. For this M.R.I. dataset, Match outperformed CALIC by 39.267%. For the C.T. scans, all but seven of the datasets have edge quality measurements that fall between 0.100 and 0.500, with the largest percent increase between 0.400 and 0.500. Similarly, most of the M.R.I. datasets have edge quality measurements between 0.300 and 0.600, with the majority of the percent differences being negative.

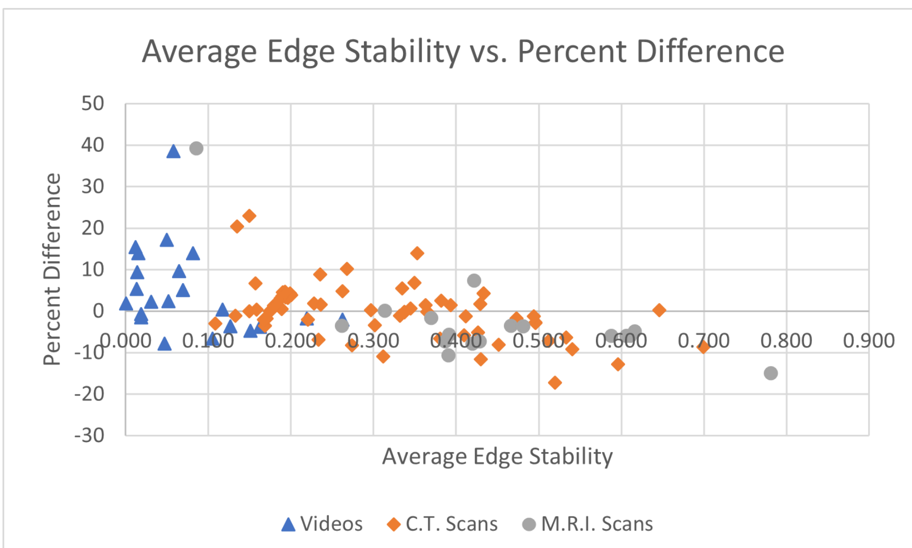

Figure 6.4: Average Edge Stability Measurement Compared to The Percent Difference Between Match and CALIC

Excluding the 4k video datasets, the smallest compression ratio when using Match is 1.446 for Crowd_Run, where Match slightly under-performs compared to CALIC by -3.856%. The edge quality measurement was calculated to be 0.163, which is the largest measurement in the video datasets. Figure 6.5 is the last image that is compressed with Match for Crowd_Run and is labeled as the fifth image. ACRIN-6698_DWI_MASK, on the other hand, has a compression ratio of 108.843 when Match is the selected prediction method. Match greatly outperforms CALIC by 39.267%, however, its edge quality measurement was calculated to be 0.086. Figure 6.6 is the fifth image but of ACRIN-6698_DWI_MASK.

Visually, Crowd_Run has very clear, hard lines throughout the image, making it easy to find the edges in the image. When compared to one of the many C.T. scans, such as WB_MAC_P690 from the miscellaneous datasets, it's clear that the edges of the subject aren't nearly as clear and look blurry. This causes the edge detection to be less accurate and thus a larger threshold is needed for determining which method results in the larger compression for the medical images.

Figure 6.5: Crowd_Run [37]

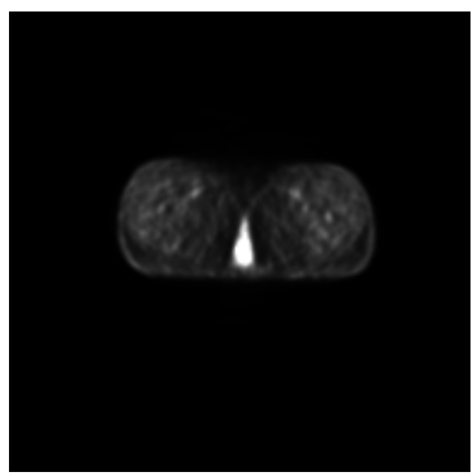

Figure 6.6: WB_MAC_P690 [38]

Figure 6.7 is the resulting edge images for each of the images that were compressed. The edge quality measurement is the mean squared error between each of these edge images. Visually, all of these edge images look identical, hence why the quality measurement is so low. However, all but one of the medical images has an edge quality measurement greater than 0.100, which is the video threshold for determining which prediction method results in better compression. Clear differences can be seen between the different edge detection images in Figure 6.8, which leads to a larger edge quality measurement. Since the majority of the medical images don't have clear defined edges like the video datasets, the threshold for which method to use needs to be adjusted to 0.300.

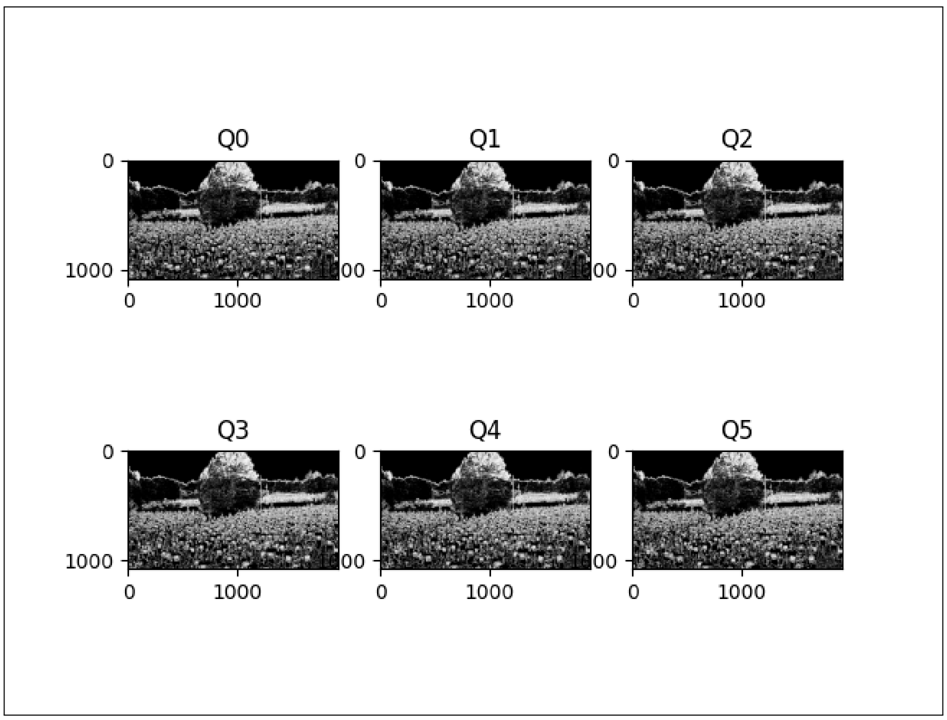

Figure 6.7: Edges of Crowd_Run

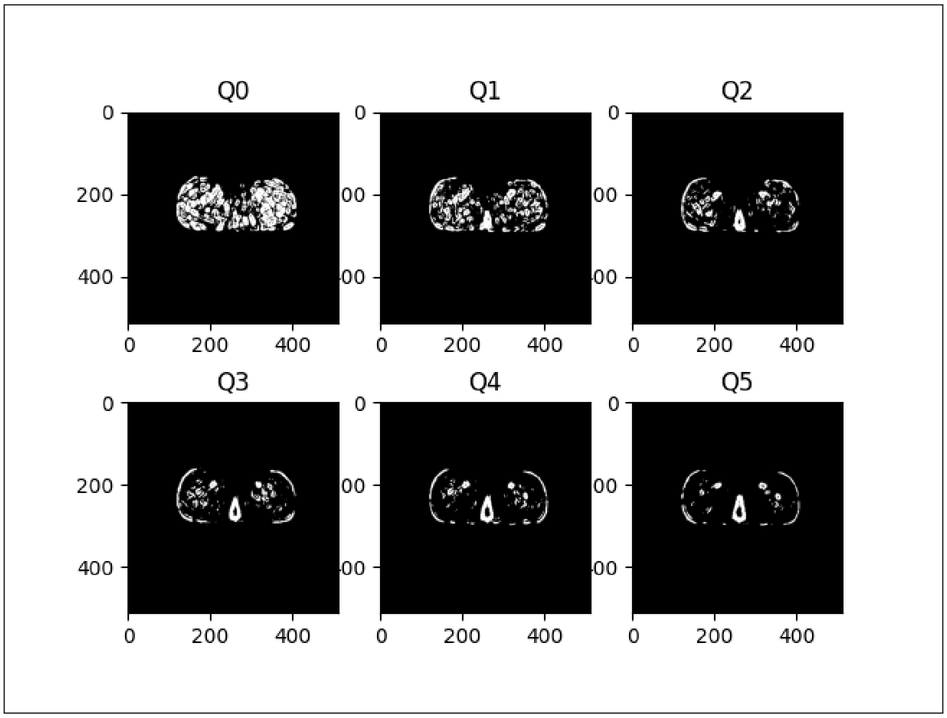

Figure 6.8: Edges of WB_MAC_P690

Table 6.8 contains the edge quality measurements between the frames of the videos with the far right column being the average.

For the 176 resolution images, all of the edge quality measurements were found to be less than 0.100. Match outperforms CALIC for Claire and Foreman with corresponding percent increase of 9.314% and 13.971%. Their average edge quality measurements were calculated to be 0.014 and 0.081 respectively. When compressing Carphone with Match, its resulting compression ratio is 5.015% better than CALIC's, therefore Match slightly outperforms CALIC. Carphone's resulting average edge stability measurement was determined to be 0.069. For the remaining two datasets, Clair(6fps) and Miss_Am, on the other hand, have minor percent differences between the two methods, 2.193% and -1.555% respectively. Therefore, no matter which method is selected to compress these datasets results in near identical compression. Their corresponding average edge stability measurements are 0.031 and 0.019.

Three of the 720 resolution images, Johnny, kristenAndSara, and Mobcal, compressing with Match results in a larger compression ratio than CALIC with percent differences of 13.906%, 15.441%, and 9.684% respectively. Like the 176 resolution images, their corresponding edge quality measurements are less than 0.100 with values of 0.015, 0.012, and 0.064. Parkrun and Shields, on the other hand, have minor percent differences of -2.096% and -1.776% respectively.

Out of the eight 1080 resolution images, only two of the datasets, Controlled_Burn and Life, are best compressed with CALIC with ratio increases of 17.239% and 38.581% respectively. Their corresponding edge stability measurements were calculated to be 0.050 and 0.058, which both are less than 0.100. On the other hand, CALIC compresses better than Match for Station2 and Aspen with percent differences of -6.602%, and -7.774% respectively. It was calculated that Station2 has an average edge stability measurement of 0.105, and Aspen has an average edge stability measurement of 0.047. Therefore, it will be accurately predicted that Station2 should be compressed with CALIC. On the other hand, it is incorrectly predicted that Match should be the method to compress Aspen. The remaining datasets: Blue_Sky, Crowd_Run, Riverbed, and Dinner all have minor percent differences between the two methods of -3.627%, -3.856%, -4.781% and -1.852%. Their corresponding edge stability measurements are 0.127, 0.163, 0.151, and 0.000 respectively. However, due to the minor percent differences, both methods result in near identical compression and thus the predicted method cannot be incorrect.

The 4k resolution datasets are represented without the Netflix portion of their name in Table 6.8. Three of the 4k resolution videos, Netflix_Boat, Netflix_BoxingPractice, and Netflix_Tango, all have minor percent differences between the two methods of 0.373%, 2.319%, and -0.645% respectively. Their corresponding average edge stability measurements were calculated to be 0.118, 0.052, and 0.018;

where the first will be compressed with CALIC and the others with Match. Due to their low percent differences, it doesn't matter which method is predicted to use. Netflix_Narrator, on the other hand, is compressed best with Match with a percent difference of 5.341% and its average edge stability measurement is 0.013, which follows the trend. However, none of these percent increases are large enough to make the computationally difficult method of Match worth it. Therefore, all of these 4k resolution datasets should be compressed with CALIC, and thus it doesn't matter how accurate the edge stability measurement is at predicting which method to use for this resolution.

Resolution	Dataset	1-0	2-1	3-2	4-3	5-4	Average
176	Claire	0.017	0.022	0.011	0.008	0.012	0.014
	Claire (6fps)	0.047	0.034	0.021	0.036	0.017	0.031
	Carphone	0.091	0.061	0.096	0.064	0.034	0.069
	Foreman	0.094	0.087	0.076	0.077	0.072	0.081
	Miss_Am	0.020	0.020	0.017	0.019	0.017	0.019
720	Johnny	0.000	0.018	0.019	0.019	0.019	0.015
	kristenAndSara	0.000	0.015	0.015	0.015	0.016	0.012
	Mobcal	0.069	0.053	0.076	0.075	0.049	0.064
	Parkrun	0.267	0.257	0.265	0.264	0.259	0.262
	Shields	0.217	0.216	0.220	0.221	0.221	0.219
1080	Blue_Sky	0.127	0.128	0.127	0.126	0.125	0.127
	Controlled_Burn	0.051	0.039	0.050	0.047	0.061	0.050
	Crowd_Run	0.181	0.163	0.154	0.153	0.163	0.163
	Riverbed	0.164	0.150	0.150	0.142	0.150	0.151
	Station2	0.106	0.105	0.105	0.104	0.105	0.105
	Aspen	0.049	0.047	0.050	0.048	0.040	0.047
	Dinner	0.000	0.000	0.000	0.001	0.000	0.000
	Life	0.056	0.059	0.058	0.058	0.060	0.058
4K	Boat	0.117	0.116	0.118	0.118	0.118	0.118
	BoxingPractice	0.051	0.052	0.052	0.052	0.053	0.052
	Narrator	0.014	0.014	0.013	0.013	0.013	0.013
	Tango	0.018	0.018	0.019	0.019	0.018	0.018

Table 6.8: Edge Quality Measurement of Videos

The edge quality measurements were calculated for the C.T. scans; however, unlike the video datasets, all of the C.T. scans have measurements between 0.100 and 0.700. This is clearly illustrated in Figure 6.3. If the threshold of 0.100 remains the same, then CALIC would be the method predicted for all of the C.T. datasets. If all of the predictions are CALIC, then the accuracy of the edge quality measurement prediction method would become 89.423% as ten medical datasets: nine C.T. scans and one M.R.I. scan; would be incorrectly predicted. However, if the threshold remains 0.100 for the video datasets, but is adjusted to 0.300 for the medical images, then the accuracy greatly improves.

Table 6.9 contains the edge quality measurements for the miscellaneous C.T. datasets, where the far right column is the average. Of these five datasets, three have percent differences that are less than |5| %: 4D-Lung, CT_FUSION, and WB_MAC_P690. Therefore, the predicted method for each of these datasets will be correct either way. Of these three datasets, only 4D-Lung has an average edge stability measurement less than 0.300 with a value of 0.263 and would thus be compressed with Match. The other two, CT_FUSION and WB_MAC_P690, have corresponding average edge quality measurements of 0.433 and 0.473; therefore CALIC is the predicted method. TCGA-38-4628 is the only dataset where Match outperforms CALIC with a percent increase of 13.895%, however it has an average edge stability of 0.353, which leads to the inaccurate prediction that CALIC would result in better compression. WB_MAC_P690, on the other hand, is best compressed with CALIC as Match under performs by -17.227%. The average edge stability measurement for this dataset was calculated to be 0.519, and therefore the predicted method of CALIC is accurate.

Dataset	1-0	2-1	3-2	4-3	5-4	Average
4D-Lung	0.282	0.267	0.256	0.252	0.256	0.263
TCGA-38-4628	0.352	0.350	0.351	0.360	0.352	0.353
NLST-LSS	0.633	0.223	0.554	0.579	0.609	0.519
CT_FUSION	0.446	0.462	0.436	0.410	0.413	0.433
WB_MAC_P690	0.540	0.591	0.419	0.424	0.390	0.473

Table 6.9: Edge Quality Measurement of Miscellaneous C.T. Datasets

Table 6.10 contains the edge quality measurements for the LIDC-IRDI C.T. datasets. All of these datasets have minor percent differences between the two prediction methods with the largest being 1.789% for LIDC-IRDI-0007. Therefore, either method predicted using the edge quality measurement is accurate. In numerical order, the average edge quality measurements were calculated to be 0.1869, 0.365, 0.150, 0.337, 0.337, 0.180, 0.228, 0.159, 0.363, and 0.297. Applying the threshold of 0.300, datasets

LIDC-IRDI-0001, LIDC-IRDI-0003, LIDC-IRDI-0006, LIDC-IRDI-0007, LIDC-IRDI-0008, and LIDC-IRDI-0010 would be compressed with Match. While the remaining datasets, LIDC-IRDI-0002, LIDC-IRDI-0004, LIDC-IRDI-0005, and LIDC-IRDI-0009 would be compressed with CALIC.

Dataset	1-0	2-1	3-2	4-3	5-4	Average
LIDC-IRDI-0001	0.198	0.187	0.182	0.188	0.187	0.189
LIDC-IRDI-0002	0.353	0.363	0.362	0.369	0.378	0.365
LIDC-IRDI-0003	0.166	0.149	0.142	0.148	0.143	0.150
LIDC-IRDI-0004	0.332	0.351	0.338	0.339	0.325	0.337
LIDC-IRDI-0005	0.332	0.351	0.338	0.339	0.325	0.337
LIDC-IRDI-0006	0.218	0.187	0.161	0.160	0.173	0.180
LIDC-IRDI-0007	0.216	0.240	0.232	0.221	0.230	0.228
LIDC-IRDI-0008	0.149	0.167	0.172	0.159	0.147	0.159
LIDC-IRDI-0009	0.371	0.370	0.363	0.353	0.359	0.363
LIDC-IRDI-0010	0.310	0.282	0.291	0.306	0.295	0.297

Table 6.10: Edge Quality Measurement of LIDC-IRDI C.T. Datasets

The edge quality measurements for the various AMC C.T. datasets are in Table 6.11.

AMC-001 contains the datasets WB_NAC_P690, WB_MAC_P690, CT_FUSION, coronals, and CHST_1.25MM_SHARP. Their corresponding average edge quality measurements are 0.430, 0.473, 0.433, 0.646, and 0.268. Applying the 0.300 threshold results in all but CHST_1.25MM_SHARP being compressed with CALIC. The first dataset, WB_NAC_P690, is accurately predicted as Match under-performs for this dataset by -11.587%. However, the second, third, and fourth datasets, WB_MAC_P690, CT_FUSION, and coronals, have minor percent differences between the two prediction methods of -1.785%, 4.285%, and 0.301% respectively; therefore, either method is an accurate prediction.

AMC-002 is the smallest of the ACM datasets with only two datasets, CHEST_1.0_B45f and coronals. The average edge quality measurement for CHEST_1.0_B45f was calculated to be 0.132 and for coronals was calculated to be 0.410. With the 0.300 threshold, CHEST_1.0_B45f will be compressed with Match while coronals would be compressed with CALIC. Both of these predictions are accurate as the percent difference for CHEST_1.0_B45f is insignificant as its -1.064% and Match slightly under-performs CALIC by -5.774% for coronals.

Seven datasets, CORONAL_MPR_2MM, CT_FUSION, CTAC, THORAX_LUNG_1MM, THORAX_LUNG_2MM, WB_MAC_P690, and WB_NAC_P690, are in ACM-003. The first two of these datasets have average edge quality measurements, 0.512 and 0.335 respectively, that are greater than 0.300 and therefore CALIC is the method that is predicted. For CORONAL_MPR_2MM, Match under-performs CALIC by -7.122%, and therefore the average edge quality measurement prediction is accurate. However, CT_FUSION is best compressed with Match by 5.451%, and therefore the prediction is inaccurate. Match outperforms CALIC by 8.878% for the CTAC dataset. Its average edge stability measurement was calculated to be 0.235, which is less than 0.300 and is therefore accurately predicting that Match is the better compression method. The remaining four datasets, THORAX_LUNG_1MM, THORAX_LUNG_2MM, WB_MAC_P690, and WB_NAC_P690, all have minor percent differences of 0.000%, 0.000%, -2.925%, and -3.579%. Therefore, no matter which way the average edge stability measurement predicts, it's accurate. These datasets are all predicted that Match is the method that should be used with corresponding average edge stability measurements of 0.175, 0.175, 0.108, and 0.168.

Like AMC-001, AMC-004 has five datasets, coronals, CT_FUSION, THORAX_1.0_B45f, WB_MAC_P690, and WB_NAC_P690. All of these datasets have average edge stability measurements greater than 0.300 with values of 0.496, 0.429, 0.332, 0.381, and 0.451 respectively. Therefore, CALIC is the selected method for all of these datasets. The first three datasets, coronals, CT_FUSION, and THORAX_1.0_B45f, have minor percent differences between Match and CALIC of -2.834%, 1.703%, and-1.064%. Therefore, both methods perform nearly identical and thus either method selected would be accurate. However, WB_MAC_P690 and WB_NAC_P690, have larger percent differences of -6.612% and -8.104% respectively. Due to the larger percent differences, CALIC is the method that results in better compression and is thus properly predicted.

AMC-005 also contains five datasets, CHEST_1.0_B45f, CHEST_2.0_coronal, CHEST_2.0_Sagittal, CHEST_5.0_B31f, and CHEST_7.0_MIP_Axia. The first and fourth datasets have minor percent differences of -1.818% and -2.779% and thus whichever method is selected is accurate. CHEST_1.0_B45f has an average edge stability measurement of 0.171 and Match will be chosen. CHEST_5.0_B31f, on the other hand, has an average edge stability measurement of 0.496 and therefore CALIC is selected for this dataset. The second and third datasets, CHEST_2.0_coronal and CHEST_2.0_Sagittal, are both best compressed with CALIC as Match under-performs by -12.838% and -10.936% respectively. Their corresponding average edge stability measurements are 0.596 and 0 312 and therefore it's accurately predicted that CALIC should be the selected method. Unlike the other datasets, CHEST_7.0_MIP_Axia is best compressed with Match as it slightly outperforms CALIC by 6.725%. It's average edge stability measurement was calculated to be 0.157, which is less than 0.300 and thus accurately predicts that Match is the best method for this dataset.

Similar to AMC-003, AMC-006 also has seven datasets, AP_MM, CORONAL_AP, CT_images, LUNG_1MM, PET_BODY_CTAC, PET_BODY_NO_AC, and ST_CAP_5MM. Three of these datasets, AP_1MM, PET_BODY_CTAC, and ST_CAP_5MM all have minor percent differences between Match and CALIC with corresponding percentages of 1.517%, -2.083%, and 0.621%. Therefore, the prediction method for these three datasets will be accurate no matter which method is selected. Their average edge quality measurements were calculated to be 0.236, 0.220, and 0.345 respectively. Since the first two datasets have measurements less than 0.300, they will be compressed with Match and the last dataset will be compressed with CALIC. Match under-performs CALIC for two of the datasets, CORONAL_AP and PET_BODY_NO_AC by -9.179% and -6.828% respectively. Their corresponding average edge quality measurements were calculated to be 0.540 and 0.234; therefore the first method is correctly predicted as the value is greater than 0.300, but the second is incorrect as Match would be the selected method. Match outperforms CALIC for two remaining datasets, CT_images and LUNG_1MM, with corresponding percent increases of 6.758% and 20.365%. The average edge quality measurement for CT_images was calculated to be 0.350; since it's greater than 0.300, it's improperly predicted that CALIC will result in a larger compression ratio. On the other hand, the averages edge quality measurement for LUNT_1MM was calculated to be 0.135 and is therefore accurately predicted.

The remaining ACM dataset, AMC-007, has five datasets: THORAX_1.0_B45f, Thorax_2.0_SPO_cor, Thorax_2.0_SP_sag, Thorax_5.0_B31f, and Thorax_7.0_MIP_ax. Their average edge quality measurements were calculated to be 0.167, 0.699, 0.274, 0.412, and 0.382 respectively. Therefore, Thorax_2.0_SPO_cor, Thorax_5.0_B31f, and Thorax_7.0_MIP_ax will be compressed with CALIC as their measurements are greater than 0.300. This prediction is accurate for all three of theses datasets as

Thorax_2.0_SPO_cor has a percent difference of -8.571% while the other two have minor percent differences of -1.183% and 2.506% respectively. Due to the minor percent difference for the last two datasets, either method selected would be accurate. The remaining two datasets, THORAX_1.0_B45f and Thorax_2.0_SP_sag, have average edge quality measurements less than 0.300, therefore Match would be the selected method. This prediction is inaccurate for Thorax_2.0_SP_sag as Match under-performs CALIC by -8.217%. However, the prediction is accurate for THORAX_1.0_B45f as the percent difference between the two methods is minimal at -2.097%.

Table 6.12 contains the edge quality measurements between each frame in the C.T. datasets under 4D-Lung, where the far right column is the average measurement. For all but one of these datasets, 60, have minor percent differences less than $|5|$%. In order, excluding 60, the percent differences were found to be 4.250%, 3.891%, 3.064%, 3.286%, 3.146%, 3.615%, -3.340%, 4.488%, and 4.731%. All of these datasets but 70 have edge quality measurements less than 0.300 with exact values of 0.199, 0.201, 0.187, 0.194, 0.196, 0.190, 0.190, and 0.193 respectively; therefore, Match is the predicted method for these datasets. Dataset 70, on the other hand, has an average edge quality measurement of 0.301 and therefore would be compressed with CALIC. Match outperforms the remaining dataset, 60, by 22.975% and has an edge quality measurement of 0.150 which accurately predicts that Match is the method that would result in the best compression.

	Dataset	1-0	2-1	3-2	4-3	5-4	Average
1	WB_NAC_P690	0.447	0.586	0.411	0.328	0.376	0.430
	WB_MAC_P690	0.540	0.591	0.419	0.424	0.390	0.473
	CT_FUSION	0.446	0.462	0.436	0.410	0.413	0.433
	coronals	0.648	0.680	0.641	0.624	0.638	0.646
	CHST_1.25MM_SHARP	0.268	0.265	0.271	0.268	0.267	0.268
2	CHEST_1.0_B45f	0.136	0.127	0.131	0.134	0.134	0.132
	coronals	0.369	0.362	0.413	0.467	0.439	0.410
3	CORONAL_MPR_2MM	0.487	0.463	0.488	0.556	0.568	0.512
	CT_FUSION	0.349	0.352	0.322	0.322	0.330	0.335
	CTAC	0.244	0.238	0.231	0.228	0.236	0.235
	THORAX_LUNG_1MM	0.166	0.180	0.174	0.180	0.175	0.175
	THORAX_LUNG_2MM	0.166	0.180	0.174	0.180	0.175	0.175
	WB_MAC_P690	0.103	0.180	0.067	0.077	0.114	0.108
	WB_NAC_P690	0.205	0.289	0.168	0.000	0.180	0.168
4	coronals	0.489	0.499	0.486	0.491	0.516	0.496
	CT_FUSION	0.449	0.432	0.416	0.420	0.428	0.429
	THORAX_1.0_B45f	0.383	0.334	0.306	0.317	0.321	0.332
	WB_MAC_P690	0.181	0.759	0.510	0.119	0.335	0.381
	WB_NAC_P690	0.783	0.546	0.374	0.268	0.285	0.451
5	CHEST_1.0_B45f	0.178	0.167	0.163	0.180	0.165	0.171
	CHEST_2.0_coronal	0.470	0.586	0.593	0.589	0.741	0.596
	CHEST_2.0_Sagittal	0.294	0.322	0.300	0.313	0.328	0.312
	CHEST_5.0_B31f	0.473	0.513	0.494	0.507	0.493	0.496
	CHEST_7.0_MIP_Axia	0.169	0.170	0.161	0.153	0.134	0.157
6	AP_1MM	0.231	0.239	0.230	0.239	0.242	0.236
	CORONAL_AP	0.455	0.348	0.495	0.707	0.698	0.540
	CT_images	0.352	0.356	0.332	0.343	0.365	0.350
	LUNG_1MM	0.168	0.165	0.170	0.000	0.171	0.135
	PET_BODY_CTAC	0.215	0.204	0.197	0.266	0.220	0.220
	PET_BODY_NO_AC	0.287	0.158	0.251	0.276	0.195	0.234
	ST_CAP_5MM	0.335	0.368	0.329	0.352	0.338	0.345
7	THORAX_1.0_B45f	0.175	0.174	0.165	0.164	0.160	0.167
	Thorax_2.0_SPO_cor	0.683	0.717	0.721	0.700	0.674	0.699
	Thorax_2.0_SP_sag	0.294	0.285	0.269	0.263	0.262	0.274
	Thorax_5.0_B31f	0.422	0.434	0.405	0.389	0.407	0.412
	Thorax_7.0_MIP_ax	0.405	0.356	0.390	0.376	0.381	0.382

Table 6.11: Edge Quality Measurement of AMC C.T. Datasets

Dataset	1-0	2-1	3-2	4-3	5-4	Average
0	0.186	0.195	0.209	0.199	0.207	0.199
10	0.189	0.190	0.208	0.208	0.208	0.201
20	0.177	0.181	0.191	0.195	0.192	0.187
30	0.179	0.189	0.197	0.204	0.200	0.194
40	0.188	0.188	0.201	0.204	0.202	0.196
50	0.179	0.181	0.198	0.195	0.195	0.190
60	0.167	0.169	0.226	0.000	0.188	0.150
70	0.177	0.174	0.443	0.349	0.364	0.301
80	0.178	0.182	0.193	0.200	0.199	0.190
90	0.185	0.185	0.191	0.198	0.206	0.193

Table 6.12: Edge Quality Measurement of 4D-Lung C.T. Datasets

The edge quality measurements for the four datasets under CMB-CRC-MSB-02381 are in Table 6.13 and are titled as Body_5.000CE_1, Body_5.000CE_2, Body_5.0CE_1, and Body_5.0CE_2. Each of these datasets have average edge quality measurements that are greater than 0.300 with values of 0.533, 0.427, 0.394, and 0.494 respectively. For this batch of C.T. scans, the average edge quality measurement accurately predicts that all these datastes should be compressed with CALIC. For the first two datasets, Body_5.000CE_1 and Body_5.000CE_2, Match slighly underperforms CALIC by -6.317%, and -5.100%. For the last two datasets, Body_5.0CE_1 and Body_5.0CE_2; on the other hand, Match and CALIC result in nearly identical compression as the percent difference between the two are 1.486%, and -1.295% respectively.

Dataset	1-0	2-1	3-2	4-3	5-4	Average
Body_5.000CE_1	0.540	0.524	0.516	0.548	0.538	0.533
Body_5.000CE_2	0.413	0.394	0.426	0.433	0.467	0.427
Body_5.0CE_1	0.378	0.359	0.372	0.433	0.426	0.394
Body_5.0_CE_2	0.498	0.501	0.487	0.490	0.496	0.494

Table 6.13: Edge Quality Measurement of CMB-CRC-MSB-02381 C.T. Datasets

The final batch of medical images are the 17 M.R.I. datasets; their edge quality measurements between frames as well as the average of those measurements are listed in Table 6.14. All but two of the datasets, ACRIN-6698_DWI_MASK and ISPY2_multiphase384, have edge quality measurements greater than 0.300 and therefore it's predicted that CALIC should be the compression method used. ACRIN-6698_DWI_MASK is the only medical dataset that follows the trend of the video datasets as it's average edge quality measurement is less than 0.100 with a value of 0.086 and Match outperforms CALIC by 39.267%. ISPY2_multiphase384, on the other hand, has an average edge stability measurement of 0.262, however the percent difference between Match and CALIC is minimal at -3.592% and therefore either Match or CALIC could be chosen and the result is accurate. Predicting CALIC for the remaining datasets is accurate for all but one of the datasets, ISPY2_Volser_SER, as it's average edge quality measurement is 0.422 but Match outperforms CALIC by 7.389%. Five of the CALIC predicted datasets, ACRIN-6698_ADC, ISPY2_VOLSER_DCE, ISPY2_VOLSER_PE2, ISPY2_VOLSER_PE6, and ISPY2_Water_T2fseidealarc_BP, have minor percent differences of 0.136%, -3.626%, ,-1.623%, -3.565%, and -4.933% respectively. Their corresponding average edge quality measurements are 0.314, 0.418, 0.370, 0.466, and 0.616. Due to the small percent differences for these datases, either method predicted results in an accurate prediction. Nine of the datasets where CALIC is predicted accurately predict that CALIC will result in the best compression. These datasets are ACRIN-6698_4bval, ACRIN-6698_DWI_TRACE, ISPY2_3_Plane_Scout, ISPY2_Fat_T2fseidealarc_BP, ISPY2_Fieldmap, ISPY2_IP_T2fseidealarc_BP, ISPY2_OP_T2fseidealarc_BP, ISPY2_T2fseidealarc_BP, and ISPY2_WATER_T2_fseidealarc_BP. Their corresponding average edge quality measurements were calculated to be 0.428, 0.588, 0.391, 0.387, 0.781, 0.420, 0.391, 0.611, and 0.606. For each of these ten datsets, Match under-performed CALIC by -7.346%, -5.948%, -10.691%, -7.303%, -14.997%, -7.888%, -5.645%, -5.896%, and -5.896% respectively.

Dataset	1-0	2-1	3-2	4-3	5-4	Average
ACRIN-6698_4bval	0.406	0.409	0.467	0.469	0.391	0.428
ACRIN-6698_ADC	0.329	0.301	0.309	0.321	0.309	0.314
ACRIN-6698_DWI_MASK	0.017	0.076	0.121	0.070	0.144	0.086
ACRIN-6698_DWI_TRACE	0.451	0.581	0.461	0.711	0.736	0.588
ISPY2_3_Plane_Scout	0.420	0.320	0.356	0.414	0.445	0.391
ISPY2_Fat_T2fseidealarc_BP	0.369	0.380	0.393	0.414	0.378	0.387
ISPY2_Fieldmap	0.796	0.718	0.838	0.798	0.754	0.781
ISPY2_IP_T2fseidealarc_BP	0.396	0.406	0.438	0.454	0.404	0.420
ISPY2_multiphase384	0.230	0.234	0.265	0.260	0.321	0.262
ISPY2_OP_T2fseidealarc_BP	0.388	0.373	0.392	0.420	0.384	0.391
ISPY2_T2fseidealarc_BP	0.580	0.593	0.618	0.724	0.540	0.611
ISPY2_VOLSER_DCE	0.454	0.500	0.488	0.493	0.471	0.481
ISPY2_VOLSER_PE2	0.404	0.367	0.254	0.402	0.421	0.370
ISPY2_VOLSER_PE6	0.366	0.400	0.446	0.506	0.613	0.466
ISPY2_Volser_SER	1.000	0.000	0.303	0.431	0.374	0.422
ISPY2_WATER_T2_fseidealarc_BP	0.574	0.736	0.580	0.604	0.537	0.606
ISPY2_Water_T2fseidealarc_BP	0.608	0.711	0.579	0.603	0.580	0.616

Table 6.14: Edge Quality Measurement of M.R.I. Datasets

When using the average edge quality measurement to predict which method, Match or CALIC, results in the best compression, a threshold of 0.100 is best for video datasets while a threshold of 0.300 is needed for the medical images. This is so because the videos have more clean, hard lines, while the medical images have soft edges and thus it's more difficult to find them. When looking at the detected edges for each of the images in Figures 6.7 and 6.8, there's no visible differences between the edge images for Crowd_Run, however there were visible differences in the edge images for WB_MAC_P690. Implementing the two thresholds results in seven of the datasets being improperly predicted, therefore using the average edge quality measurement results in 93.269% accuracy.

Using the structural similarity between the frames of videos, or medical images, to predict which method to use resulted in an accuracy of 86.539% using a threshold of 0.82. On the other hand, using an edge quality measurement between these frames results in an accuracy of 93.269% when using a threshold of 0.100 for the videos and 0.300 for the medical images. If we were to combine the SSIM and average edge quality measurements by using the SSIM to predict the videos and the edge quality to predict the medical images, only one prediction would change: Aspen. This would then raise the prediction accuracy to 94.231%, which isn't worth the added complexity. Therefore, it's best to use the edge quality measurement to determine which method, Match or CALIC, should be used to compress each dataset.

Chapter 7: Conclusion

There are two kinds of compression, lossy and lossless. Lossy compression allows for data to be lost but visually the compressed image looks the same. Lossless, on the other hand, requires that no data be lost to maintain the quality of the image. There are many image compression standards, such as JPEG and Portable Network Graphics (PNG). Typically JPEG is a lossy compression while PNG is a common lossless compression method.

A new lossless video compression technique, Match, was investigated. Match uses the similarity between the frames of a video or the slides of medical images to find a prediction for the current pixel, which makes it a non-linear prediction method. A portion of the previous frame is searched to find a matching context some distance centered on the current location. The best distance to use for each dataset is found experimentally. The matching context refers to the neighborhood of w, nw, n, and ne, where the pixel in the previous frame with the closest matching context becomes the prediction. From the prediction, the error is then calculated, remapped and encoded using adaptive arithmetic encoding. Match's resulting compression ratio is compared to that of CALIC's, where the larger the compression ratio the more efficient the method. Match was used to compress twenty-two video datasets of varying resolutions as well as 65 C.T. scans and 17 M.R.I. scans. Not only was Match used to compress videos as well as medical images, but it was also run on four datsets that had varying resolutions to see how resolution affected Match. Unfortunately, out of the four datasets, there were three different trends, thus there's no clear conclusion to how Match is affected by resolution. It's assumed that as the frame rate increases Match's compression ratio will also increase which is shown in a single dataset. Therefore, no clear conclusion can be stated on how frame rate affects Match. There are three possible results when comparing Match to CALIC: Match outperforms CALIC, CALIC outperforms Match, and both result in nearly identical compression ratios. To determine which method to use, the structural similarity was examined as well as the edge quality measurements. Using the structural similarity with a threshold of 0.820 resulted in 86.538% accuracy. On the other hand, using the edge quality measurement with a threshold of 0.100 for the videos and 0.300 for the medical images resulted in 93.269% accuracy for predicting which method to use.

Match is similar to some of the other compression methods that have been discussed as it encodes the residual errors from the prediction of the pixels. However, Match is a non-linear prediction method that depends on the similarity between frames. Most predictive compression methods depends strictly on the neighboring pixels of the current pixel being encoded while Match depends on a portion of the previous frame to find a matching context within some threshold.

Videos and images contain different qualities that can affect the compression ratio depending on the compression method. With Match, the larger the structural similarity is the better the compression while the smaller the edge quality measurement the larger the compression ratio is. Therefore, it's useful to use external measurements to determine which method should be selected to compress a dataset.

Bibliography

[1] Camera Obscura, https://www.nga.gov/press/exh/2866/camera-obscura.html (accessed Sep. 4, 2023).

[2] "Camera Obscura: Ancestor Of Modern Photography," Encyclopedia.com, https://www.encyclopedia.com/science/encyclopedias-almanacs-transcripts-and-maps/camera-obscura-ancestor-modern-photography (accessed Sep. 13, 2023).

[3] "Camera Obscura," Wikipedia, https://en.wikipedia.org/wiki/Camera_obscura#/media/File:Pinhole-camera.svg (accessed Oct. 24, 2023).

[4] T. Gregory and Carlosmary, "The first camera ever made: A history of cameras," History Cooperative, https://historycooperative.org/first-camera-the-history-of-cameras/ (accessed Sep. 5, 2023).

[5] "The Daguerreotype Medium," The Library of Congress, https://www.loc.gov/collections/daguerreotypes/articles-and-essays/the-daguerreotype-medium/ (accessed Sep. 5, 2023).

[6] "Original Kodak Camera, serial no. 540," National Museum of American History, https://americanhistory.si.edu/collections/search/object/nmah_760118 (accessed Sep. 18, 2023).

[7] "Kodak 35," Wikipedia, https://en.wikipedia.org/wiki/Kodak_35 (accessed Sep. 18, 2023).

[8] The Print Collector/Heritage Images/SCIENCE PHOTO LIBRARY, "Photographic gun designed by Etienne Jules Marey, 1882 - stock image - C042/3142," Science Photo Library, https://www.sciencephoto.com/media/979649/view/photographic-gun-designed-by-etienne-jules-marey-1882 (accessed Sep. 18, 2023).

[9] "Chronophotographic gun," Wikipedia, https://en.wikipedia.org/wiki/Chronophotographic_gun (accessed Sep. 18, 2023).

[10] B. Javier, "#entertainmenttech: The history of the Kinetograph, the world's First Motion Picture Camera," iTech Post, https://www.itechpost.com/articles/109598/20220316/entertainmenttech-history-kinetograph-worlds-first-motion-picture-camera.htm (accessed Sep. 18, 2023).

[11] "The time-travelling camera: A short history of digital photo manipulation," National Science and Media Museum, https://www.scienceandmediamuseum.org.uk/objects-and-stories/digital-photo-manipulation-history (accessed Sep. 27, 2023).

[12] "The evolution of digital camera timeline.," Timetoast timelines, https://www.timetoast.com/timelines/evolution-of-digital-camera (accessed Sep. 29, 2023).

[13] "What is ISO? (And Why ISO Matters in Photography)," ExpertPhotography, https://expertphotography.com/understand-iso-4-simple-steps/ (accessed Sep. 29, 2023).

[14] J. Gray, "Apple's 29-year-old landmark quicktake 100 camera falters in 2023," PetaPixel, https://petapixel.com/2023/03/03/apples-29-year-old-landmark-quicktake-100-camera-falters-in-2023/ (accessed Sep. 29, 2023).

[15] "Quick Review Ricoh RDC-i700 Image Capturing Devices," https://www.imaging-resource.com/PRODS/I700/I70A.HTM (accessed Sep. 29, 2023).

[16] "History of digital camera timeline.," Timetoast timelines, https://www.timetoast.com/timelines/history-of-digital-camera (accessed Oct. 3, 2023).

[17] R. Skibba, "A new 3,200-megapixel camera has astronomers salivating," Wired, https://www.wired.com/story/a-new-3200-megapixel-camera-has-astronomers-salivating/ (accessed Oct. 3, 2023).

[18] P. Rhodes, "At 70 trillion fps, this is the world's fastest camera," RedShark News - Video technology news and analysis, https://www.redsharknews.com/at-70-trillion-fps-this-is-the-worlds-fastest-camera (accessed Oct. 3, 2023).

[19] Amazon.com : Nikon Cookpix B600 Digital Camera (black) (26528..., https://www.amazon.com/Nikon-COOLPIX-Digital-Flexible-Cleaning/dp/B0BZT1PLQB (accessed Oct. 3, 2023).

[20] J. Estrin, "Kodak's first digital moment," The New York Times, https://archive.nytimes.com/lens.blogs.nytimes.com/2015/08/12/kodaks-first-digital-moment/ (accessed Oct. 3, 2023).

[21] "DSLR cameras," Nikon, https://www.nikonusa.com/en/nikon-products/dslr-cameras/index.page (accessed Oct. 3, 2023).

[22] S. Andriani, G. Calvagno, and G. A. Mian, "Lossless video compression using a spatio-temporal optimal predictor," 2005.

[23] Y. Li and K. Sayood, "Lossless video sequence compression using adaptive prediction," IEEE Transactions on Image Processing, vol. 16, no. 4, pp. 997–1007, Apr. 2007, doi: 10.1109/TIP.2006.891336.

[24] K. H. Yang and A. F. Faryar, "A context-based predictive coder for lossless and near-lossless compression of video," in IEEE International Conference on Image Processing, 2000, vol. 1, pp. 144–147. doi: 10.1109/icip.2000.900915.

[25] J. Kim and C. M. Kyung, "A lossless embedded compression using significant bit truncation for hd video coding," IEEE Transactions on Circuits and Systems for Video Technology, vol. 20, no. 6, pp. 848–860, Jun. 2010, doi: 10.1109/TCSVT.2010.2045923.

[26] J. A. Choi and Y. S. Ho, "Efficient residual data coding in CABAC for HEVC lossless video compression," Signal Image Video Process, vol. 9, no. 5, pp. 1055–1066, Jul. 2015, doi: 10.1007/s11760-013-0545-z.

[27] "How medical imaging and scans work: Blog," Loyola Medicine, https://www.loyolamedicine.org/about-us/blog/how-medical-imaging-scans-work (accessed Oct. 24, 2023).

[28] Taquet, J., and Labit, C. (2012). Hierarchical oriented predictions for resolution scalable lossless and near-lossless compression of CT and MRI biomedical images. IEEE Transactions on Image Processing, 21(5), 2641–2652. https://doi.org/10.1109/TIP.2012.2186147

[29] Anusuya, V., Raghavan, V. S., & Kavitha, G. (2014). Lossless Compression on MRI Images Using SWT. Journal of Digital Imaging, 27(5), 594–600. https://doi.org/10.1007/s10278-014-9697-9

[30] Philips, W., Van Assche, S., De Rycke, D., & Denecker, K. (n.d.). State-of-the-art techniques for lossless compression of 3D medical image sets. www.elsevier.com/locate/compmedimag

[31] E. H. Sibley, I. H. Willen, R. M. Neal, and J. G. Cleary, "COMPUTING PRACTICES ARITHMETIC CODING FOR DATA COIUPRESSION."

[32] H. Hu, "A Study of CALIC," 2004.

[33] K. Sayood, "Introduction to Data Compression," 5th ed. 2018. doi: 10.1016/c2015-0-06248-7.

[34] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," IEEE Transactions on Image Processing, vol. 13, no. 4, pp. 600–612, Apr. 2004, doi: 10.1109/TIP.2003.819861.

[35] B. Sankur, "Statistical evaluation of image quality measures," J Electron Imaging, vol. 11, no. 2, p. 206, Apr. 2002, doi: 10.1117/1.1455011.

[36] Babacan, Ş. D., & Sayood, K. (2004). Predictive Image Compression Using Conditional Averages. Data Compression Conference Proceedings, 524. https://doi.org/10.1109/dcc.2004.1281500

[37] Xiph.org video test media [derf's collection]. Xiph.org. (n.d.). Retrieved April 19, 2023, from https://media.xiph.org/video/derf/

[38] "IDC." https://portal.imaging.datacommons.cancer.gov/explore/ (accessed Feb. 12, 2023).

[39] "CT Scan," Mayo Foundation for Medical Education and Research, Jan. 06, 2022. https://www.mayoclinic.org/tests-procedures/ct-scan/about/pac-20393675#: :text=A%20computerized%20tomography%20(CT)%20scan,than%20 plain%20X%2Drays%20do. (accessed Feb. 12, 2023).

[40] U.S. Department of Health and Human Services. (n.d.). Magnetic Resonance Imaging (MRI). National Institute of Biomedical Imaging and Bioengineering. Retrieved March 27, 2023, from https://www.nibib.nih.gov/science-education/science-topics/magnetic-resonance-imaging-mri#:∼:text=projects%20in%20MRI%3F-,What%20is%20MRI%3F ,%2C%20diagnosis%2C%20and%20treatment%20monitoring.

Appendix A: C.T. Scans

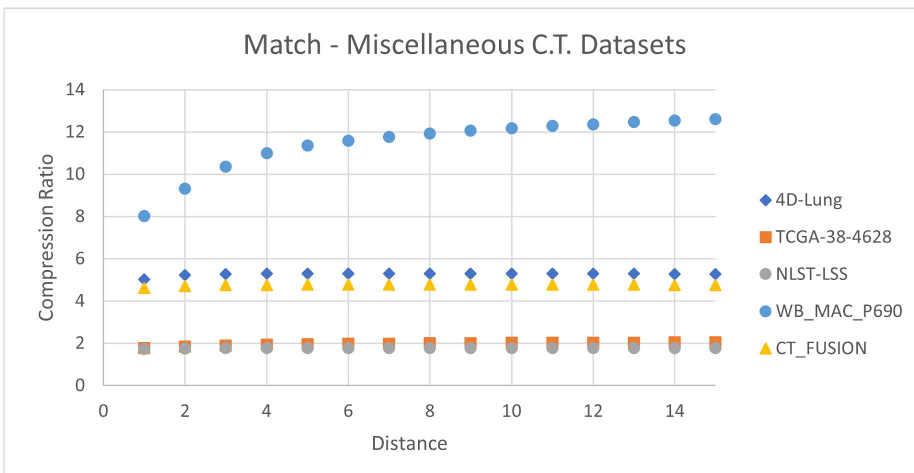

Figure A.1: Match CRs with Varying Distance - Miscellaneous C.T. Scans

Distance	4D_Lung	TCGA	NLST_LSS	CT	WB_MAC	WB_NAC
1	5.026	1.776	1.725	4.607	8.023	3.456
2	5.226	1.847	1.754	4.714	9.316	4.025
3	5.277	1.892	1.766	4.747	10.364	4.504
4	5.294	1.924	1.770	4.756	10.988	4.805
5	5.301	1.950	1.772	4.761	11.347	4.998
6	5.300	1.969	1.773	4.765	11.586	5.139
7	5.300	1.985	1.773	4.769	11.765	5.238
8	5.299	1.998	1.773	4.770	11.927	5.349
9	5.297	2.008	1.773	4.768	12.060	5.416
10	5.295	2.016	1.772	4.767	12.168	5.481
11	5.292	2.023	1.771	4.763	12.280	5.558
12	5.285	2.029	1.770	4.763	12.364	5.589
13	5.283	2.034	1.769	4.761	12.467	5.646
14	5.283	2.038	1.768	4.759	12.544	5.675
15	5.278	2.041	1.768	4.757	12.597	5.692

Table A.1: Match CRs with Varying Distance - Miscellaneous C.T. Scans

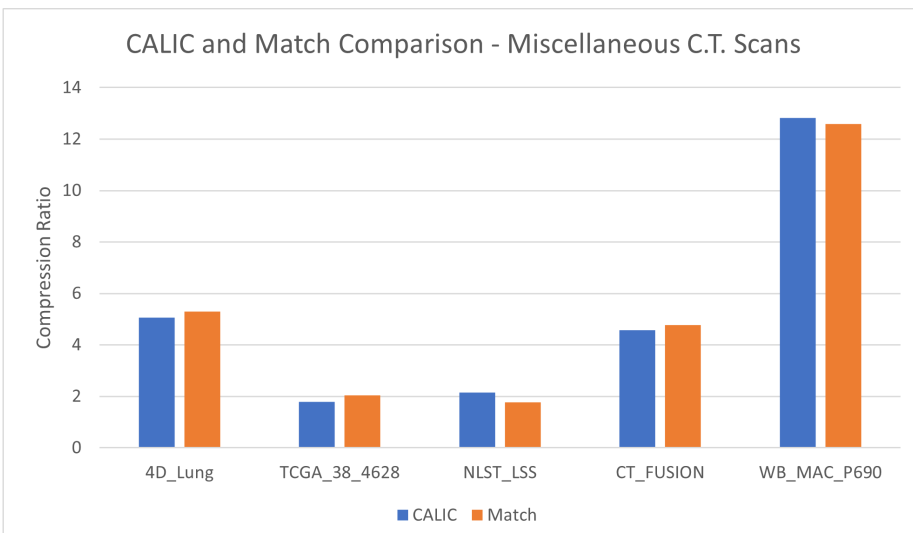

Figure A.2: CALIC's CR Compared to Match's CR - Miscellaneous C.T. Scans

Method	4D_Lung	TCGA	NLST_LSS	CT	WB_MAC
CALIC	5.057	1.792	2.142	4.574	12.826
Match	5.304	2.041	1.773	4.769	12.597
Percent Difference	4.821	13.873	-17.187	4.266	-1.785

Table A.2: CALIC's CR Compared to Match's CR - Miscellaneous C.T. Scans

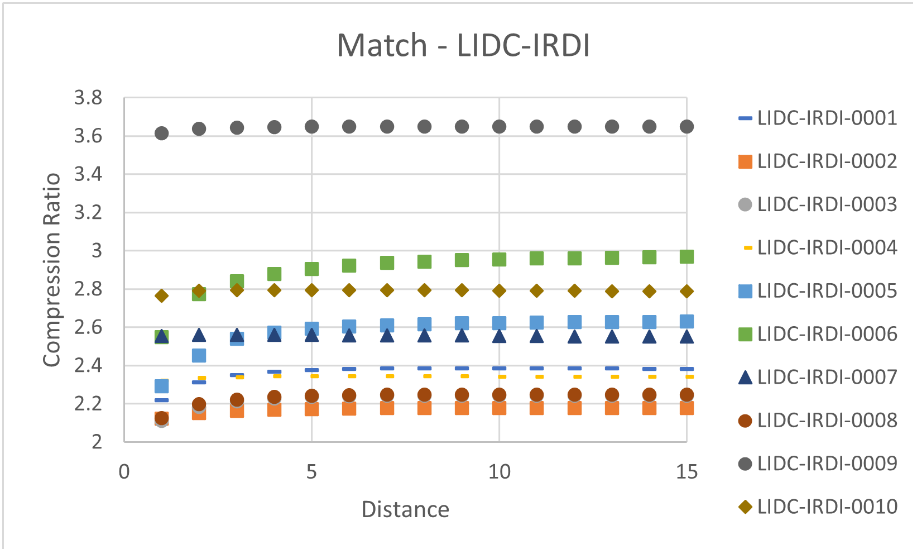

Figure A.3: Match CRs with Varying Distance - LIDC-IRDI C.T. Scans

D	1	2	3	4	5	6	7	8	9	10
1	2.219	2.123	2.111	2.320	2.291	2.548	2.556	2.127	3.615	2.765
2	2.311	2.152	2.183	2.335	2.453	2.773	2.560	2.198	3.639	2.791
3	2.349	2.162	2.214	2.340	2.539	2.842	2.561	2.223	3.644	2.795
4	2.368	2.169	2.229	2.343	2.573	2.878	2.561	2.236	3.648	2.795
5	2.377	2.173	2.236	2.344	2.5922	2.905	2.560	2.242	3.649	2.795
6	2.382	2.175	2.240	2.345	2.604	2.923	2.559	2.246	3.649	2.795
7	2.384	2.177	2.242	2.344	2.611	2.936	2.558	2.248	3.650	2.794
8	2.386	2.178	2.243	2.344	2.616	2.944	2.557	2.248	3.650	2.794
9	2.386	2.179	2.243	2.343	2.621	2.952	2.556	2.248	3.650	2.793
10	2.387	2.179	2.244	2.342	2.623	2.956	2.555	2.249	3.651	2.782
11	2.386	2.179	2.244	2.342	2.625	2.959	2.554	2.249	3.650	2.792
12	2.385	2.179	2.243	2.341	2.627	2.962	2.553	2.248	3.650	2.791
13	2.384	2.178	2.243	2.341	2.629	2.965	2.552	2.248	3.650	2.789
14	2.383	2.178	2.243	2.340	2.629	2.967	2.552	2.247	3.649	2.789
15	2.381	2.177	2.242	2.340	2.630	2.969	2.551	2.247	3.648	2.789

Table A.3: Match CRs with Varying Distance - LIDC-IRDI C.T. Scans

	1	2	3	4	5	6	7	8	9	10
C	2.376	2.182	2.246	2.350	2.637	2.932	2.516	2.241	3.596	2.781
M	2.387	2.179	2.244	2.345	2.630	2.969	2.561	2.249	3.651	2.795
D	0.464	-0.117	-0.090	-0.212	-0.248	1.254	1.784	0.366	1.523	0.502

Table A.4: CALIC's CR Compared to Match's CR - LIDC-IRDI C.T. Scans

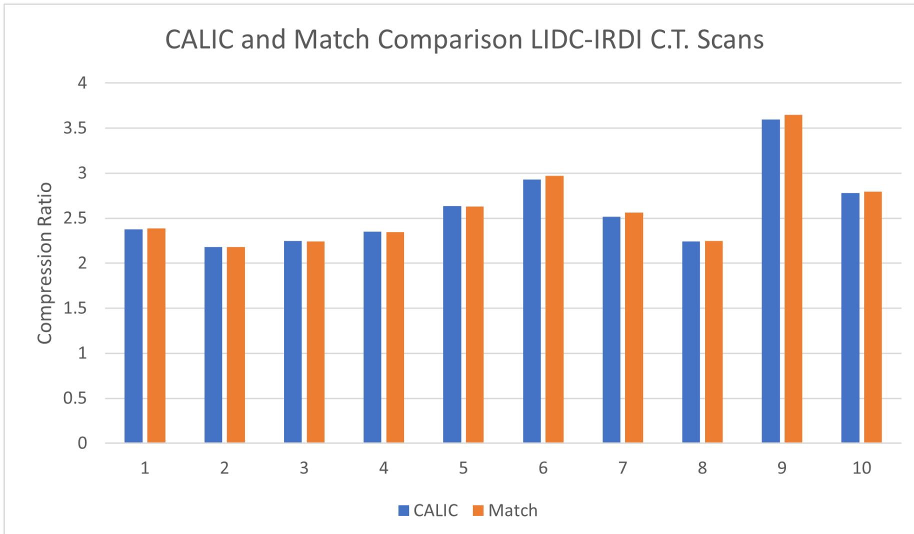

Figure A.4: CALIC's CR Compared to Match's CR - LIDC-IRDI C.T. Scans

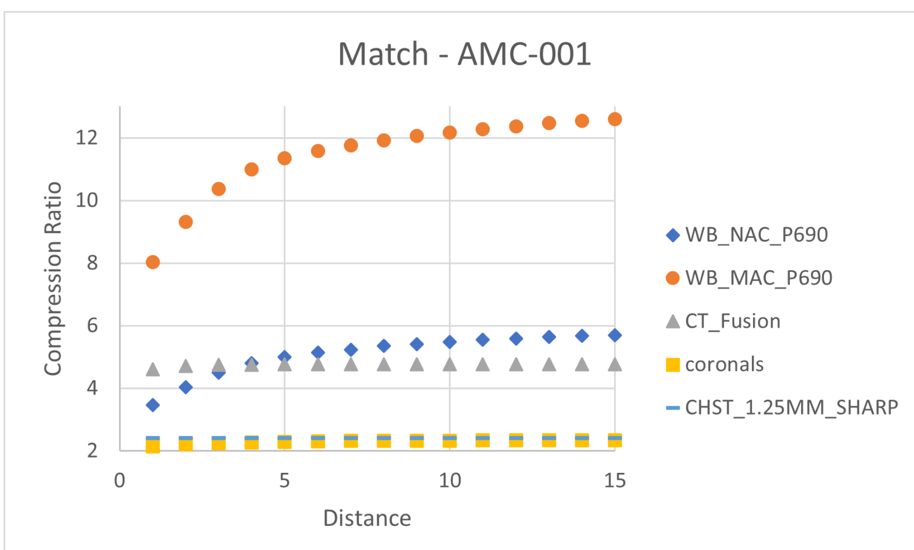

Figure A.5: Match CRs with Varying Distance - AMC-001 C.T. Scans

Distance	WB_NAC_P690	WB_MAC_P690	CT_FUSION	Coronals	CHST
1	3.456	8.023	4.607	2.134	2.402
2	4.025	9.346	4.714	2.206	2.413
3	4.504	10.364	4.747	2.244	2.415
4	4.805	10.988	4.756	2.270	2.418
5	4.998	11.347	4.761	2.291	2.418
6	5.139	11.586	4.765	2.305	2.418
7	5.238	11.765	4.769	2.315	2.417
8	5.416	11.927	4.769	2.321	2.417
9	5.416	12.060	4.768	2.325	4.417
10	5.481	12.168	4.767	2.328	2.416
11	5.558	12.280	4.764	2.331	2.415
12	5.589	12.364	4.763	2.333	2.414
13	5.646	12.467	4.761	2.333	2.414
14	5.675	12.544	4.789	2.334	2.413
15	5.692	12.597	4.757	2.336	2.413

Table A.5: Match CRs with Varying Distance - AMC-001 C.T. Scans

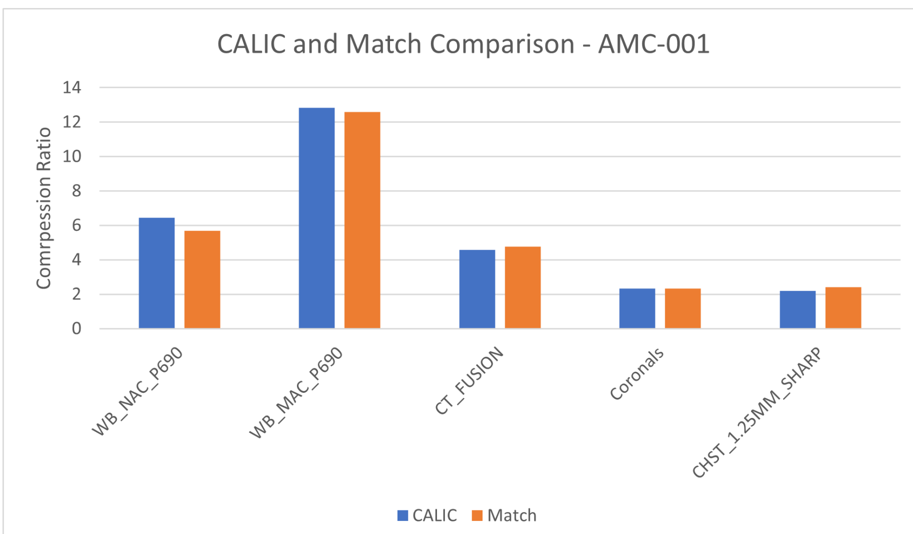

Figure A.6: CALIC's CR Compared to Match's CR - AMC-001 C.T. Scans

Method	WB_NAC_P690	WB_MAC_P690	CT_FUSION	Coronals	CHST
CALIC	6.438	12.826	4.574	2.329	2.195
Match	5.692	12.597	4.770	2.336	2.418
Percent Difference	-11.587	-1.785	4.266	0.285	10.181

Table A.6: CALIC's CR Compared to Match's CR - AMC-001 C.T. Scans

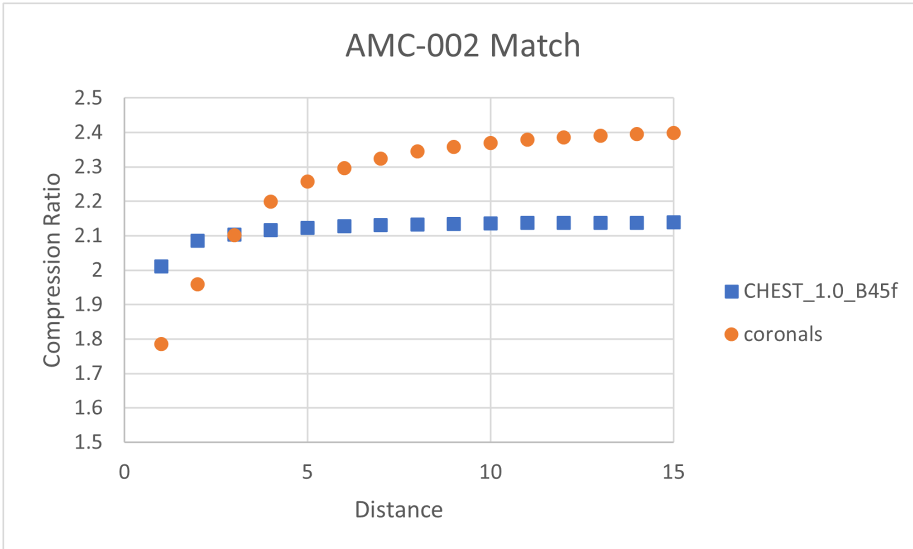

Figure A.7: Match CRs with Varying Distance - AMC-002 C.T. Scans

Distance	CHEST_1.0_B45f	Coronals
1	2.010	1.785
2	2.085	1.960
3	2.103	2.101
4	2.116	2.198
5	2.123	2.257
6	2.128	2.296
7	2.130	2.324
8	2.133	2.344
9	2.134	2.359
10	2.136	2.370
11	2.137	2.378
12	2.137	2.386
13	2.138	2.391
14	2.138	2.395
15	2.139	2.399

Table A.7: Match CRs with Varying Distance - AMC-002 C.T. Scans

Method	CHEST_1.0_B45f	Coronals
CALIC	2.162	2.546
Match	2.139	2.399
Percent Difference	-1.096	-5.809

Table A.8: CALIC's CR Compared to Match's CR - AMC-002 C.T. Scans

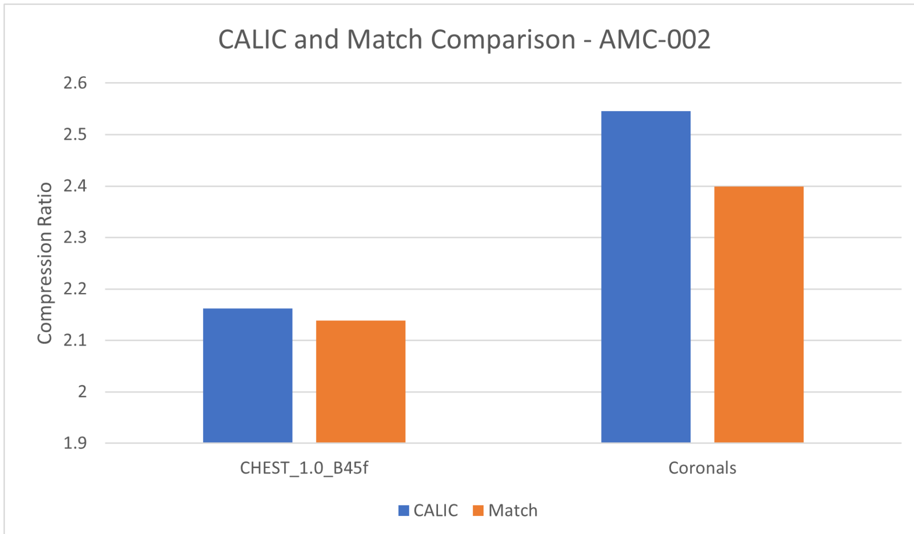

Figure A.8: CALIC's CR Compared to Match's CR - AMC-002 C.T. Scans

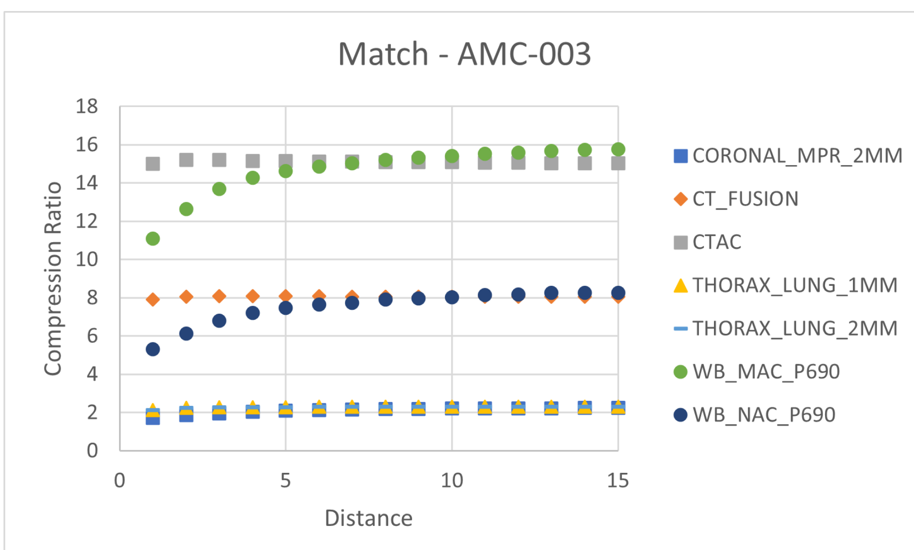

Figure A.9: Match CRs with Varying Distance - AMC-003 C.T. Scans

Distance	CORONAL	CT_FUSION	CTAC	1MM	2MM	MAC	NAC
1	1.722	7.903	15.000	2.139	2.139	11.087	5.300
2	1.855	8.058	15.220	2.242	2.242	12.630	6.123
3	1.960	8.084	15.209	2.267	2.267	13.694	6.813
4	2.043	8.087	15.161	2.282	2.282	14.284	7.222
5	2.102	8.077	15.153	2.292	2.292	14.6231	7.472
6	2.137	8.074	15.134	2.298	2.298	14.862	7.641
7	2.162	8.071	15.123	2.302	2.302	15.037	7.746
8	2.181	8.069	15.108	2.305	2.305	15.207	7.905
9	2.195	8.068	15.098	2.308	2.308	15.321	7.971
10	2.207	8.066	15.084	2.309	2.309	15.418	8.041
11	2.217	8.604	15.073	2.311	2.311	15.120	8.152
12	2.226	8.055	15.062	2.311	2.311	15.595	8.172
13	2.232	8.052	15.045	2.312	2.312	15.686	8.255
14	2.238	8.050	15.038	2.313	2.313	15.727	8.270
15	2.243	8.047	15.027	2.314	2.314	15.766	8.271

Table A.9: Match CRs with Varying Distance - AMC-003 C.T. Scans

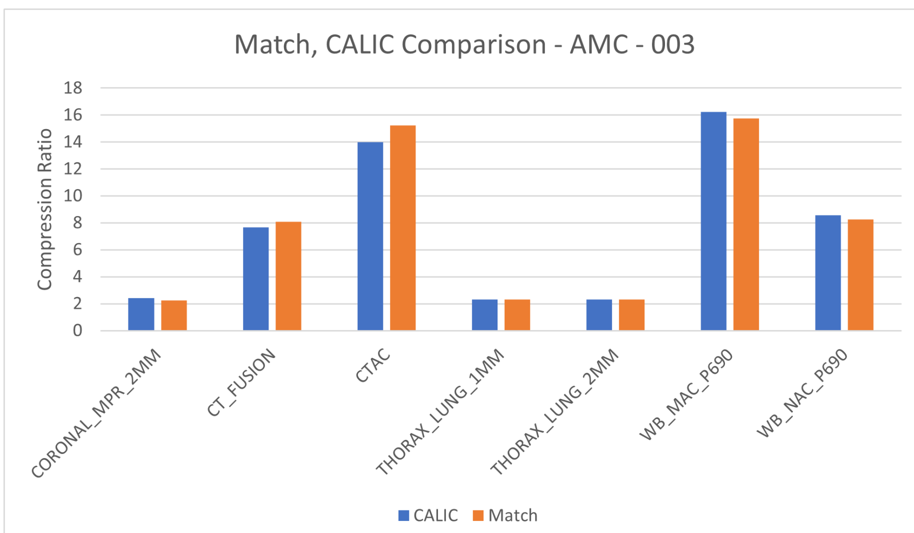

Figure A.10: CALIC's CR Compared to Match's CR - AMC-003 C.T. Scans

Method	CORONAL	CT_ FUSION	CTAC	1MM	2MM	MAC	NAC
CALIC	2.415	7.669	13.979	2.314	2.314	16.241	8.578
Match	2.243	8.087	15.220	2.314	2.314	15.766	8.271
%D	-7.117	5.445	8.879	0.001	0.001	-2.925	-3.574

Table A.10: CALIC's CR Compared to Match's CR - AMC-003 C.T. Scans

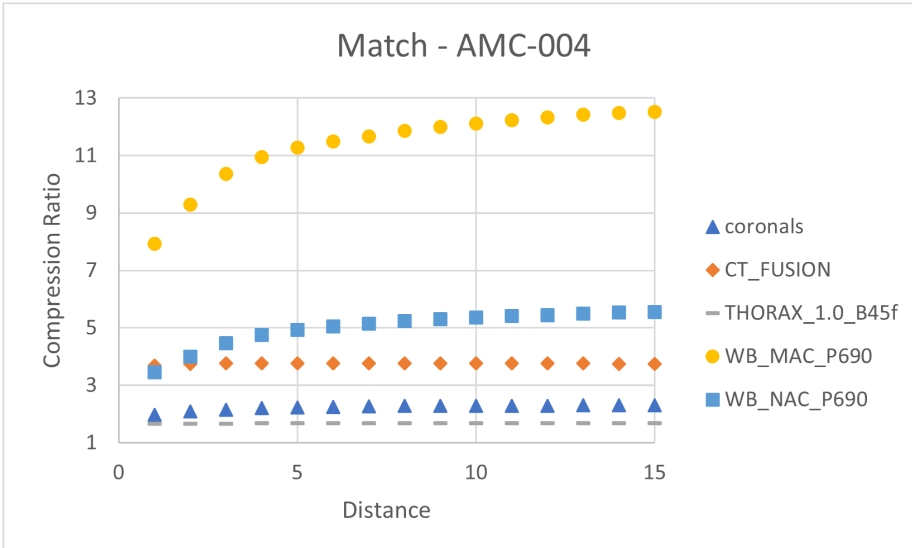

Figure A.11: Match CRs with Varying Distance - AMC-004 C.T. Scans

D	Coronals	CT_FUSION	THORAX	WB_MAC_P690	WB_NAC_P690
1	1.969	3.686	1.655	7.936	3.446
2	2.082	3.739	1.668	9.300	4.006
3	2.157	3.758	1.670	10.355	4.468
4	2.201	3.762	1.672	10.989	4.756
5	2.231	3.762	1.673	11.275	4.933
6	2.251	3.763	1.673	11.488	5.058
7	2.265	3.763	1.673	11.672	5.146
8	2.275	3.762	1.673	11.859	5.244
9	2.282	3.761	1.673	11.995	5.302
10	2.288	3.759	1.673	12.106	5.354
11	2.291	3.758	1.673	12.226	5.418
12	2.293	3.756	1.673	12.321	5.447
13	2.294	3.755	1.672	12.424	5.499
14	2.295	3.754	1.672	12.479	5.529
15	2.297	3.753	1.672	12.528	5.545

Table A.11: Match CRs with Varying Distance - AMC-004 C.T. Scans

Method	Coronals	FUSION	THORAX	WB_MAC_P690	WB_NAC_P690
CALIC	2.364	3.700	1.691	13.415	6.034
Match	2.297	3.763	1.673	12.528	5.545
Percent Difference	-2.851	1.690	-1.044	-6.613	-8.107

Table A.12: CALIC's CR Compared to Match's CR - AMC-004 C.T. Scans

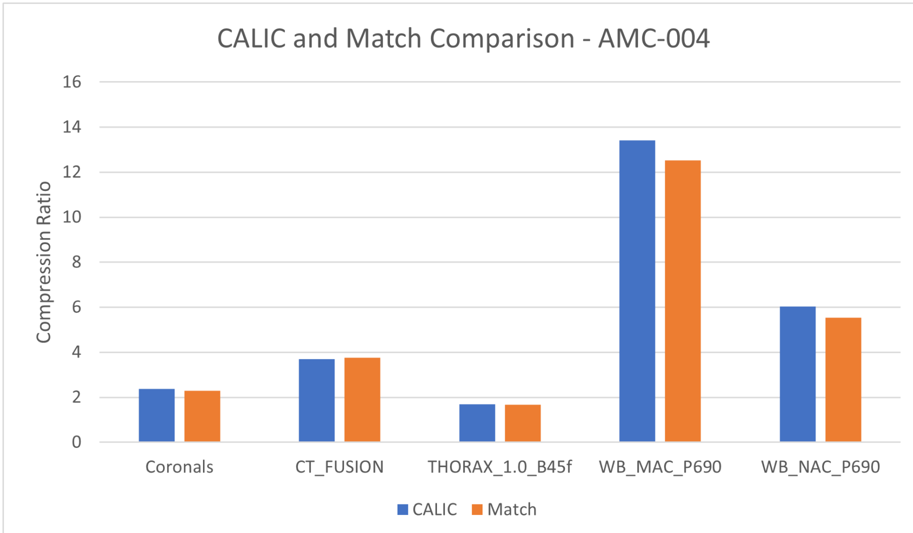

Figure A.12: CALIC's CR Compared to Match's CR - AMC-004 C.T. Scans

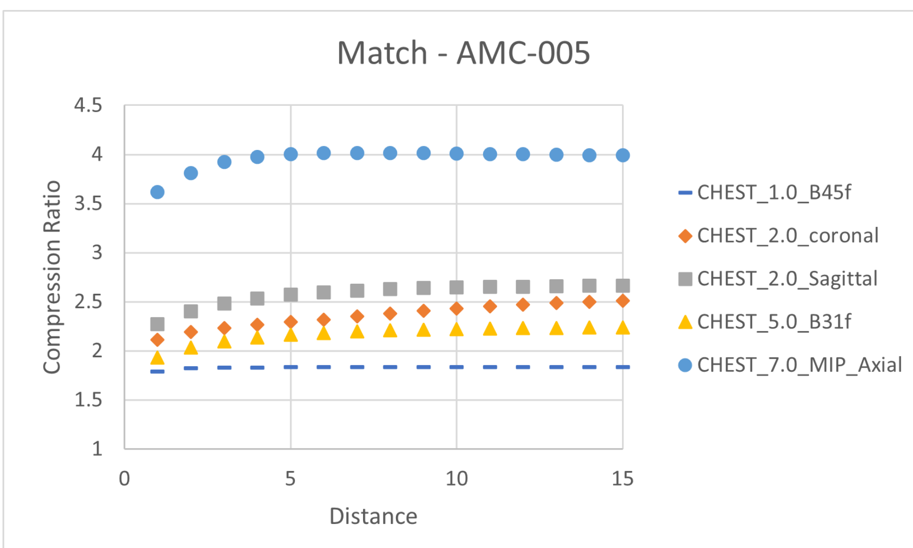

Figure A.13: Match CRs with Varying Distance - AMC-005 C.T. Scans

Distance	B45f	Coronal	Saggital	B31f	Axial
1	1.791	2.111	2.272	1.933	3.615
2	1.822	2.190	2.400	2.035	3.811
3	1.828	2.233	2.481	2.096	3.923
4	1.831	2.265	2.534	2.134	3.977
5	1.834	2.293	2.571	2.161	4.004
6	1.8.5	2.319	2.596	2.182	4.013
7	1.836	2.349	2.614	2.197	4.017
8	1.835	2.379	2.629	2.209	4.015
9	1.836	2.407	2.638	2.217	4.012
10	1.836	2.431	2.644	2.223	4.009
11	1.836	2.451	2.650	2.228	4.005
12	1.836	2.470	2.655	2.232	4.002
13	1.836	2.486	2.658	2.235	3.998
14	1.836	2.500	2.661	2.237	3.994
15	1.835	2.512	2.663	2.239	3.990

Table A.13: Match CRs with Varying Distance - AMC-005 C.T. Scans

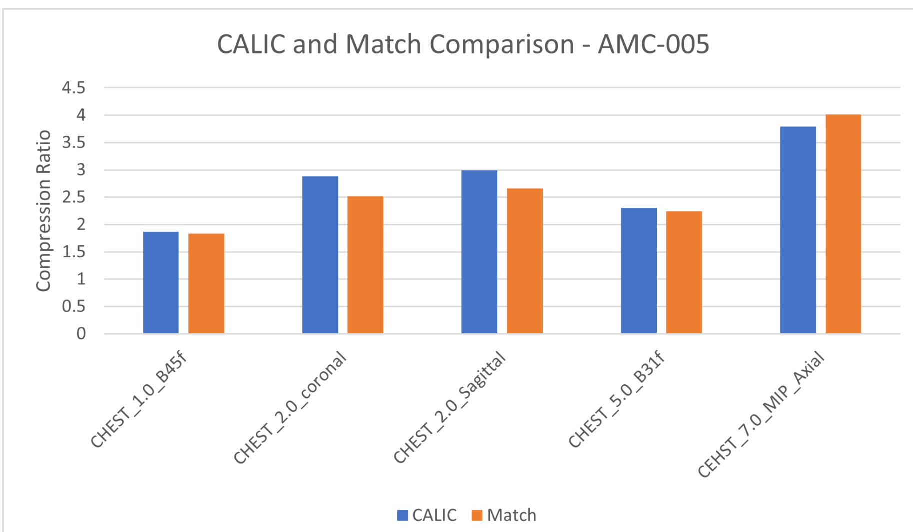

Figure A.14: CALIC's CR Compared to Match's CR - AMC-005 C.T. Scans

Method	B45f	Coronal	Saggital	B31f	Axial
CALIC	1.870	2.882	2.990	2.303	3.792
Match	1.836	2.512	2.663	2.239	4.017
Percent Difference	-1.788	-12.835	-10.937	-2.750	5.938

Table A.14: CALIC's CR Compared to Match's CR - AMC-005 C.T. Scans

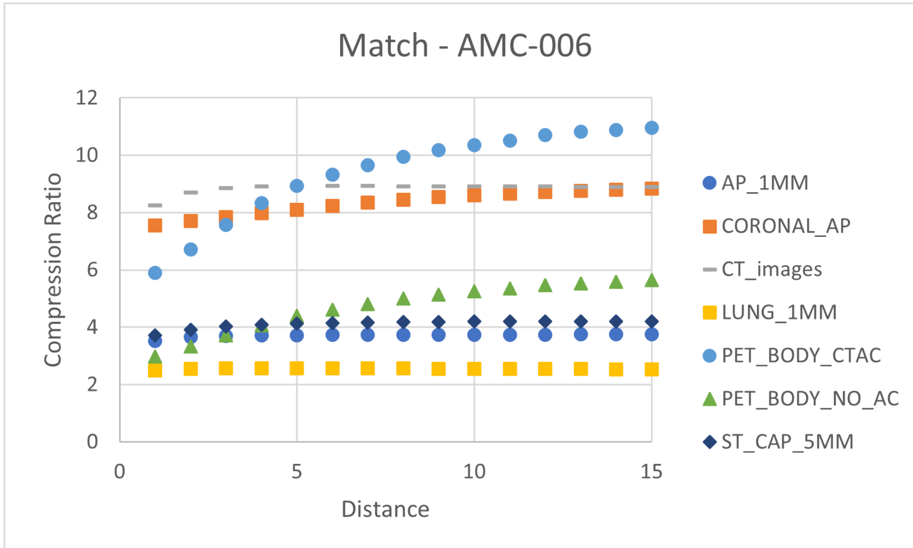

Figure A.15: Match CRs with Varying Distance - AMC-006 C.T. Scans

Distance	AP	CORONAL	CT	1MM	CTAC	NO_AC	ST_CAP
1	3.514	7.555	8.254	2.500	5.899	2.972	3.718
2	3.660	7.714	8.695	2.557	6.715	3.330	3.918
3	3.696	7.845	8.855	2.566	7.570	3.715	4.020
4	3.710	7.972	8.914	2.571	8.338	4.075	4.079
5	3.721	8.096	8.920	2.570	8.939	4.390	4.119
6	3.729	8.228	8.926	2.568	9.320	4.621	4.147
7	3.734	8.343	8.926	2.564	9.649	4.813	4.167
8	3.738	8.446	8.921	2.561	9.948	5.007	4.180
9	3.741	8.536	8.919	5.557	10.182	5.144	4.189
10	3.743	8.608	8.918	2.554	10.353	5.254	4.195
11	3.745	8.670	8.915	2.550	10.508	5.343	4.200
12	3.746	8.715	8.907	2.546	10.906	5.463	4.205
13	3.747	8.764	8.904	2.542	10.813	5.536	4.208
14	3.747	8.803	8.900	2.539	10.884	5.594	4.211
15	3.748	8.836	8.900	2.536	10.954	5.636	4.213

Table A.15: Match CRs with Varying Distance - AMC-006 C.T. Scans

Method	AP	CORONAL	CT	1MM	CTAC	NO_AC	ST_CAP
CALIC	3.692	9.729	8.361	2.136	11.187	6.049	4.187
Match	3.748	8.836	8.926	2.571	10.954	5.636	4.213
Percent Difference	1.518	-9.189	6.751	20.363	-2.076	-6.825	0.631

Table A.16: CALIC's CR Compared to Match's CR - AMC-006 C.T. Scans

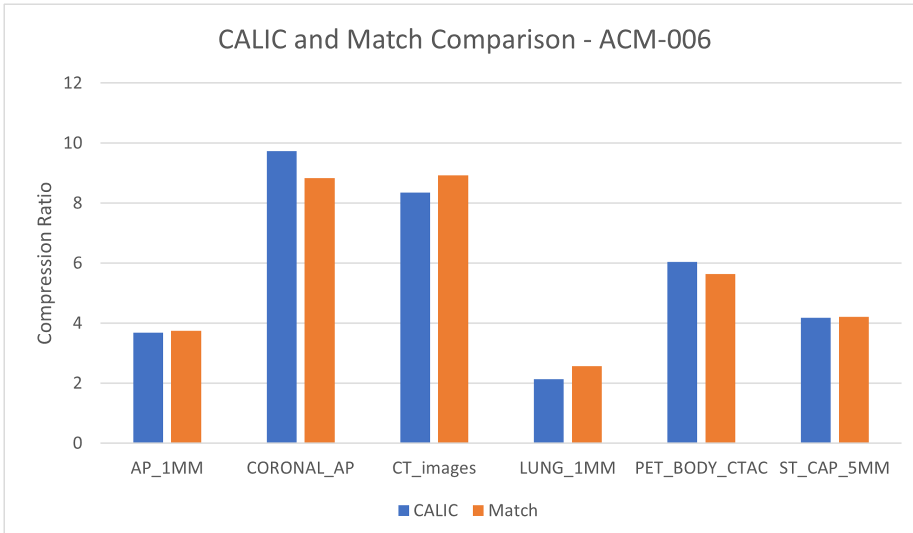

Figure A.16: CALIC's CR Compared to Match's CR - AMC-006 C.T. Scans

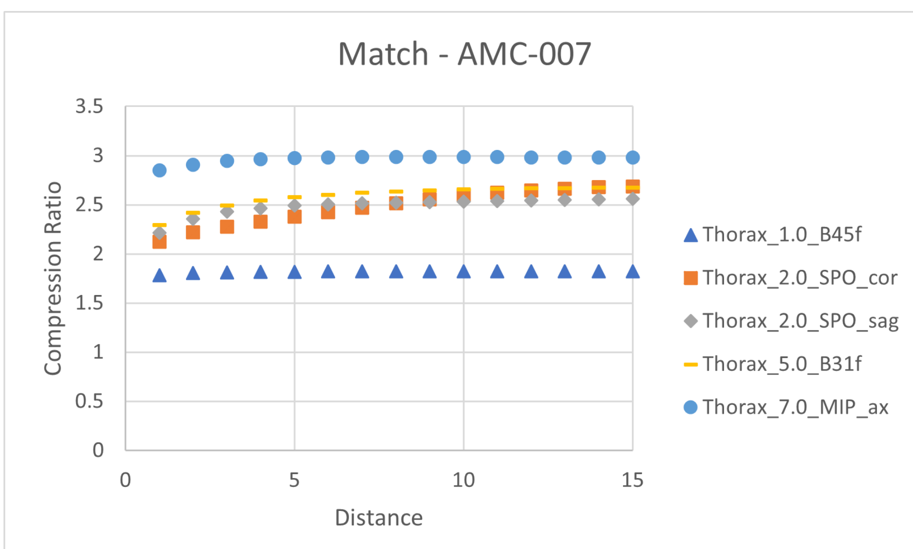

Figure A.17: Match CRs with Varying Distance - AMC-007 C.T. Scans

Distance	B45f	SPO_cor	sag	B31f	ax
1	1.782	2.125	2.216	2.296	2.848
2	1.805	2.219	2.359	2.418	2.907
3	1.811	2.278	2.430	2.493	2.945
4	1.816	2.329	2.466	2.542	2.966
5	1.819	2.378	2.490	2.576	2.976
6	1.820	2.423	2.504	2.602	2.980
7	1.821	2.468	2.514	2.622	2.985
8	1.822	2.514	2.522	2.636	2.986
9	1.822	2.556	2.528	2.647	2.985
10	1.822	2.593	2.534	2.655	2.985
11	1.822	2.623	2.540	2.661	2.984
12	1.822	2.647	2.545	2.666	2.982
13	1.822	2.665	2.550	2.669	2.981
14	1.821	2.678	2.554	2.672	2.980
15	1.821	2.688	2.558	2.673	2.978

Table A.17: Match CRs with Varying Distance - AMC-007 C.T. Scans

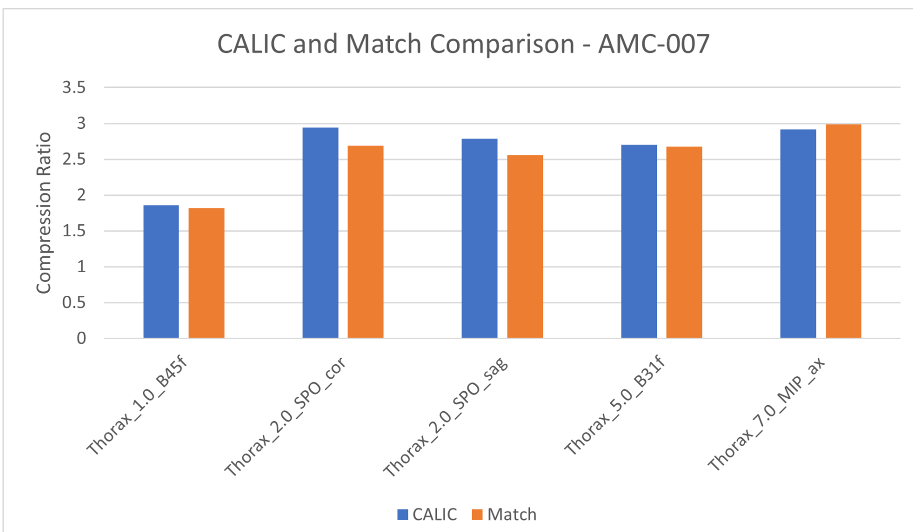

Figure A.18: CALIC's CR Compared to Match's CR - AMC-007 C.T. Scans

Distance	B45f	SPO_cor	sag	B31f	ax
CALIC	1.860	2.940	2.787	2.705	2.913
Match	1.822	2.688	2.558	2.673	2.986
Percent Difference	-2.026	-8.581	-8.225	-1.205	2.493

Table A.18: CALIC's CR Compared to Match's CR - AMC-007 C.T. Scans

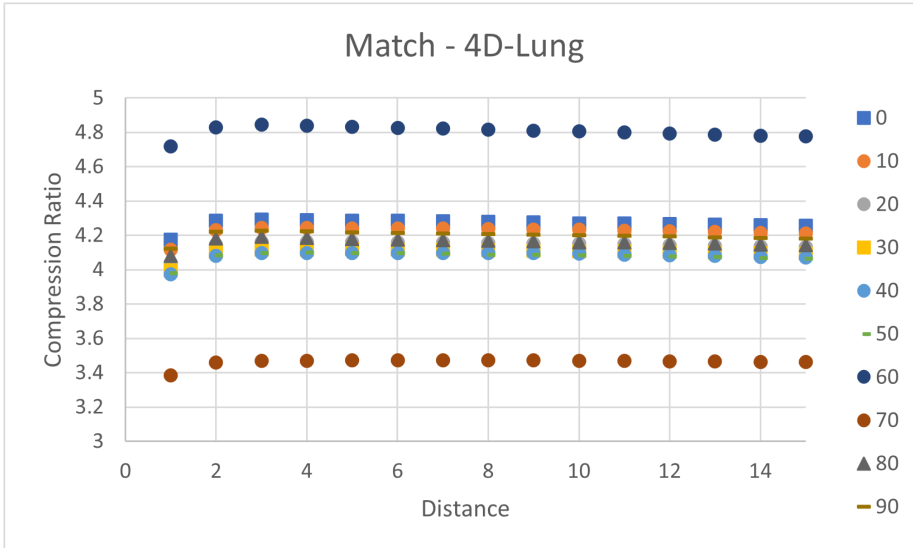

Figure A.19: Match CRs with Varying Distance - 4D-Lung C.T. Scans

D	0	10	20	30	40	50	60	70	80	90
1	4.174	1.448	4.037	3.998	3.974	3.979	4.717	3.385	4.082	4.122
2	4.287	4.232	4.147	4.102	4.082	4.084	4.829	3.459	4.181	4.222
3	4.293	4.244	4.162	4.117	4.096	4.097	4.844	3.468	4.191	4.228
4	4.290	4.456	4.162	4.117	4.098	4.099	4.839	3.470	4.187	4.222
5	4.287	4.242	4.161	4.118	4.098	4.096	4.831	3.471	4.177	4.217
6	4.285	4.239	4.159	4.116	4.098	4.096	4.826	3.472	4.175	4.214
7	4.283	4.239	4.158	4.115	4.098	4.093	4.824	3.473	4.173	4.210
8	4.278	4.237	4.155	4.113	4.097	4.089	4.817	3.473	4.168	4.207
9	4.275	4.235	4.152	4.111	4.096	4.086	4.810	3.471	4.164	4.204
10	4.270	4.233	4.151	4.108	4.092	4.084	4.806	3.470	4.159	4.202
11	4.269	4.228	4.147	4.106	4.088	4.081	4.800	3.468	4.158	4.198
12	4.265	4.222	4.144	4.103	4.084	4.077	4.783	3.467	4.154	4.195
13	4.263	4.220	4.139	4.100	4.079	4.074	4.786	3.465	4.151	4.190
14	4.259	4.215	4.137	4.098	4.075	4.069	4.780	3.464	4.147	4.186
15	4.257	4.211	4.133	4.04	4.073	4.065	4.776	3.462	4.144	4.181

Table A.19: Match CRs with Varying Distance - 4D-Lung C.T. Scans

Method	0	10	20	30	40
CALIC	4.118	4.086	4.014	3.987	3.973
Match	4.293	4.245	4.137	4.118	4.098
Percent Difference	4.244	3.890	3.698	3.285	3.143

Table A.20: CALIC's CR Compared to Match's CR - 4D-Lung C.T. Scans

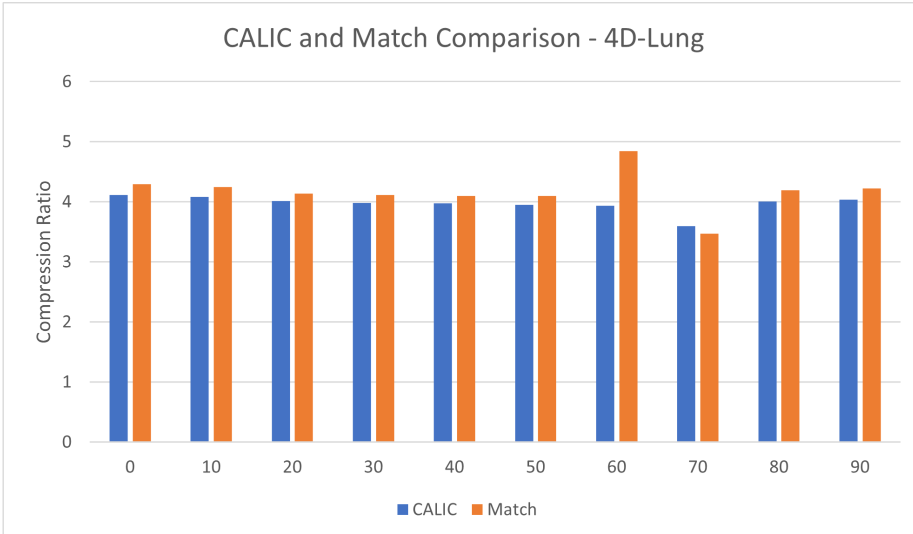

Figure A.20: CALIC's CR Compared to Match's CR - 4D-Lung C.T. Scans

Method	50	60	70	80	90
CALIC	3.956	3.939	3.563	4.011	4.037
Match	4.099	4.844	3.473	4.191	4.228
Percent Difference	3.624	22.981	-3.341	4.472	4.715

Table A.21: CALIC's CR Compared to Match's CR - 4D-Lung C.T. Scans

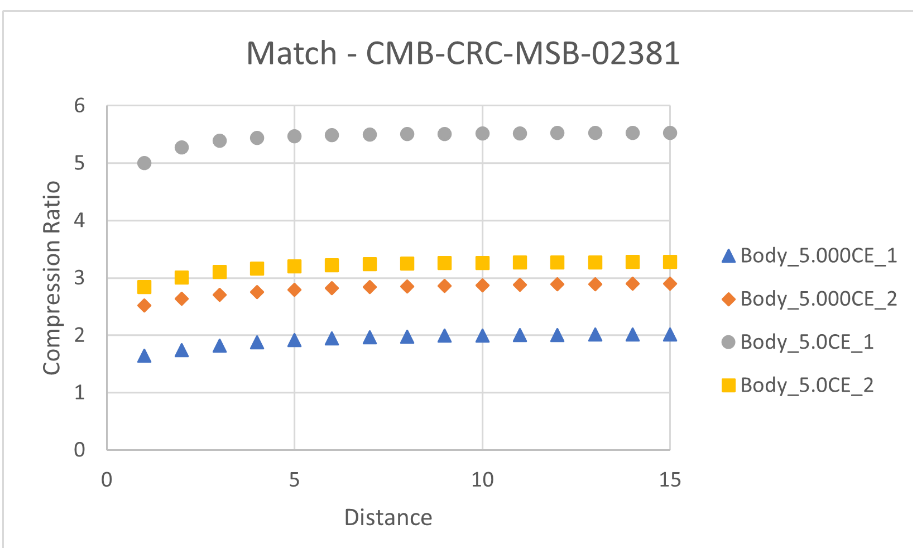

Figure A.21: Match CRs with Varying Distance - CMB-CRC-MSB-02381 C.T. Scans

Distance	Body_5.000CE_1	Body_5.000CE_2	Body_5.0CE_1	Body_5.0CE_2
1	1.645	2.525	5.002	2.846
2	1.744	2.638	5.274	3.005
3	1.820	2.708	5.388	3.108
4	1.876	2.757	5.436	3.166
5	1.915	2.792	5.467	3.201
6	1.943	2.819	5.486	3.225
7	1.962	2.839	5.498	3.240
8	1.976	2.855	5.507	3.251
9	1.994	2.867	5.513	3.258
10	1.994	2.875	5.519	3.262
11	2.000	2.884	5.523	3.2267
12	2.005	2.889	5.525	3.271
13	2.010	2.895	5.527	3.274
14	2.013	2.899	5.528	3.275
15	2.017	2.903	5.531	3.277

Table A.22: Match CRs with Varying Distance - CMB-CRC-MSB-02381 C.T. Scans

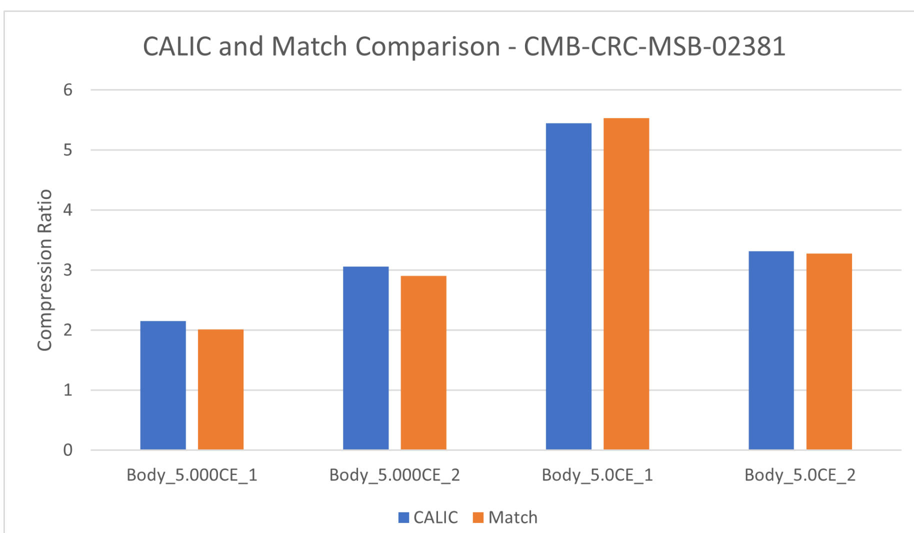

Figure A.22: CALIC's CR Compared to Match's CR - CMB-CRC-MSB-02381 C.T. Scans

Method	Body_5.000CE_1	Body_5.000CE_2	Body_5.0CE_1	Body_5.0CE_2
CALIC	2.153	3.059	5.450	3.320
Match	2.017	2.903	5.531	3.277
Percent Difference	-6.353	-5.122	1.487	-1.280

Table A.23: CALIC's CR Compared to Match's CR - CMB-CRC-MSB-02381 C.T. Scans

Appendix B: Code

```
#include <stdio.h>
#include <stdlib.h>

#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/opencv.hpp>
#include <iostream>

#include <tiffio.h>
#include <tiff.h>

#include <math.h>

#include "encoder.cpp"
#include "decoder.cpp"
#include "calic.cpp"

using namespace cv;
using namespace std;

uint32_t h, w, x, y;
uint8_t z,c;
Mat decodedImage, prevDecodedImage, decodedImage0, decodedImage1,
    decodedImage2, decodedImage3, decodedImage4, decodedImage5;
uint64_t fileLocation;
uint64_t fileSize;
uint64_t maxFileLocation;
```

```
uint32_t fileByteRW = 32768;
uint32_t maxCount = 16383;

uint32_t matrix[256][3001] = {0};
FILE * statistics = NULL;


int main(){
  int64_t i = 0, j = 0, k = 0, l = 0, m = 0;
  uint16_t symbol;
  Mat image, image0, image1, image2, image3, image4, image5;
  Vec3b processing;
  int process;
  Vec3b NorthNorth, NorthNorthEast, NorthWest, North, NorthEast, WestWest,
      West, Pixel;
  Vec3b prevNorthNorth, prevNorthNorthEast, prevNorthWest, prevNorth,
      prevNorthEast, prevWestWest, prevWest, prevPixel;

  //Images
  image0 = imread("/path/to/file/image0.png", IMREAD_UNCHANGED);
  image1 = imread("/path/to/file/image1.png", IMREAD_UNCHANGED);
  image2 = imread("/path/to/file/image2.png", IMREAD_UNCHANGED);
  image3 = imread("/path/to/file/image3.png", IMREAD_UNCHANGED);
  image4 = imread("/path/to/file/image4.png", IMREAD_UNCHANGED);
  image5 = imread("/path/to/file/image5.png", IMREAD_UNCHANGED);

  if(image0.empty() || image1.empty() || image2.empty() || image3.empty() ||
      image4.empty() || image5.empty()){
    perror("Error with imread");
    return -1;
  }
  else{
    printf("The image has been successfully opened. \n");
  }

  Mat difference1;
  absdiff(image1, image0, difference1);
```

```
imwrite("miss_am_difference.jpg", difference1);

//Now to find the width and height of the image using opencv stuff
Size s = image0.size();
h = s.height;
w = s.width;
int c;
c = image0.channels();
printf("The size of the image is: [%d, %d, %d]\n", h, w, c);

//Initialize Encoder stuff
output_array_e = (uint8_t *) malloc(fileByteRW*sizeof(uint8_t)); //Right now this is h*w*3

//open the output file
encodedFile = fopen("encoded.bin", "w");

//initialize the tables that translate between symbol indexes and characters
for(i = 0; i < 256; i++){
  pix_to_index_e[i] = i+1;
  index_to_pix_e[i + 1] = i;
}

//initialize the symbol counts and cummulative counts
for(i = 0; i <= 256; i ++){
  symbol_count_e[i] = 1;
  cum_count_e[i] = 256 - i;
}

symbol_count_e[0] = 0; //count[0] must not be the same as count[1]

//Onto Match
uint8_t threshold = 0;
int16_t initialPrediction;
int16_t error;
uint8_t remap;
uint16_t distance = 1;
```

```
Mat img1, img2;
img1.create(h,w,image0.type());
img2.create(h,w,image0.type());
Mat errors_pos, errors_neg, errors;
uint8_t flag = 0;
int16_t Xdiff, Ydiff;
uint16_t Xadd, Yadd;
int16_t offset;

//encode image0 using CALIC
for(y = 0; y < h; y++){
  for(x = 0; x < 2; x++){
    for(z = 0; z < c; z++){
      processing = image0.at<Vec3b>(y,x);
      process = processing.val[z];
      symbol = pix_to_index_e[process]; //translate to an index
      encode(symbol);
    }
  }
}

for(x = 2; x < w; x++){
  for(y = 0; y < 2; y++){
    for(z = 0; z < c; z++){
      processing = image0.at<Vec3b>(y,x);
      process = processing.val[z];
      symbol = pix_to_index_e[process]; //translate to an index
      encode(symbol);
    }
  }
}

for(y = 2; y < h; y++){
  for(x = 2; x < w; x++){
    for(z = 0; z < c; z++){
      NorthNorth = image0.at<Vec3b>(y - 2,x);
      NorthWest = image0.at<Vec3b>(y - 1,x - 1);
```

```
            North = image0.at<Vec3b>(y-1,x);
            WestWest = image0.at<Vec3b>(y,x - 2);
            West = image0.at<Vec3b>(y,x - 1);
            Pixel = image0.at<Vec3b>(y,x);
            if (x == w-1){
              NorthNorthEast = 0;
              NorthEast = 0;
            }
            else{
              NorthNorthEast = image0.at<Vec3b>(y - 2,x + 1);
              NorthEast = image0.at<Vec3b>(y - 1,x + 1);
            }

            initialPrediction = uint16_t(initially_predict(NorthNorth.val[z],
                NorthNorthEast.val[z],NorthWest.val[z],North.val[z],NorthEast.val[
                z],WestWest.val[z],West.val[z]));
            error = uint16_t(Pixel.val[z]) - uint16_t(initialPrediction);

            //Remap so that all the error values are positive and within the range
                0-255
            remap = Remap(error, initialPrediction);
            symbol = pix_to_index_e[remap];
            encode(symbol);
        }
      }
    }

    //Encode the rest of the images using only Match
    Mat errorImg;
    errorImg.create(h,w,image0.type());

    for(int imageCount = 1; imageCount < 6; imageCount++){//6
      printf("ImageCount: %d\n", imageCount);
      if(imageCount == 1){
        img1 = image0;
        img2 = image1;
      }
```

```
    else if(imageCount == 2){
      img1 = image1;
      img2 = image2;
    }
    else if(imageCount == 3){
      img1 = image2;
      img2 = image3;
    }
    else if(imageCount == 4){
      img1 = image3;
      img2 = image4;
    }
    else if(imageCount == 5){
      img1 = image4;
      img2 = image5;
    }

    //encode first two rows and columns
    uint16_t symbol;
    for(y = 0; y < h; y++){
      for(x = 0; x < 2; x++){
        for(z = 0; z < c; z++){
          processing = img2.at<Vec3b>(y,x);
          process = processing.val[z];
          symbol = pix_to_index_e[process]; //translate to an index
          encode(symbol);
        }
      }
    }

    for(x = 2; x < w; x++){
      for(y = 0; y < 2; y++){
        for(z = 0; z < c; z++){
          processing = img2.at<Vec3b>(y,x);
          process = processing.val[z];
          symbol = pix_to_index_e[process]; //translate to an index
          encode(symbol);
```

```
            }
        }
    }

    //onto Match
    for(y = 2; y < h; y++){
        for(x = 2; x < w; x++){
            for(z = 0; z < c; z++){
                NorthNorth = img2.at<Vec3b>(y - 2,x);
                NorthWest = img2.at<Vec3b>(y - 1,x - 1);
                North = img2.at<Vec3b>(y-1,x);
                WestWest = img2.at<Vec3b>(y,x - 2);
                West = img2.at<Vec3b>(y,x - 1);
                Pixel = img2.at<Vec3b>(y,x);
                if (x == w-1){
                    NorthNorthEast = 0;
                    NorthEast = 0;
                }
                else{
                    NorthNorthEast = img2.at<Vec3b>(y - 2,x + 1);
                    NorthEast = img2.at<Vec3b>(y - 1,x + 1);
                }

                //look for an exact match in a fram around the image
                Ydiff = y - distance;
                Xdiff = x - distance;
                Yadd = y + distance;
                Xadd = x + distance;

                if(Ydiff < 2 && Xdiff < 2 && Yadd < h && Xadd < w){
                    for(j = 2; j < Yadd; j++){
                        for(i = 2; i < Xadd; i++){
                            prevNorthNorth = img1.at<Vec3b>(j - 2,i);
                            prevNorthWest = img1.at<Vec3b>(j - 1,i - 1);
                            prevNorth = img1.at<Vec3b>(j-1,i);
                            prevWestWest = img1.at<Vec3b>(j,i - 2);
                            prevWest = img1.at<Vec3b>(j,i - 1);
```

```
                prevPixel = img1.at<Vec3b>(j,i);
                if (x == w-1){
                  prevNorthNorthEast = 0;
                  prevNorthEast = 0;
                }
                else{
                  prevNorthNorthEast = img1.at<Vec3b>(j - 2,i + 1);
                  prevNorthEast = img1.at<Vec3b>(j - 1,i + 1);
                }
                if(abs(prevWest.val[z] - West.val[z]) == threshold && abs(
                    prevNorthWest.val[z] - NorthWest.val[z]) == threshold &&
                    abs(prevNorth.val[z] - North.val[z]) == threshold && abs(
                    prevNorthEast.val[z] - NorthEast.val[z]) == threshold){
                  initialPrediction = prevPixel.val[z];
                  flag = 1;
                  goto finish1;
                }
              }
            }
          }
          else if(Ydiff < 2 && Xdiff < 2 && Yadd <= h && Xadd >= w){
            for(j = 2; j < Yadd; j++){
              for(i = 2; i < w; i++){
                prevNorthNorth = img1.at<Vec3b>(j - 2,i);
                prevNorthWest = img1.at<Vec3b>(j - 1,i - 1);
                prevNorth = img1.at<Vec3b>(j-1,i);
                prevWestWest = img1.at<Vec3b>(j,i - 2);
                prevWest = img1.at<Vec3b>(j,i - 1);
                prevPixel = img1.at<Vec3b>(j,i);
                if (x == w-1){
                  prevNorthNorthEast = 0;
                  prevNorthEast = 0;
                }
                else{
                  prevNorthNorthEast = img1.at<Vec3b>(j - 2,i + 1);
                  prevNorthEast = img1.at<Vec3b>(j - 1,i + 1);
                }
```

```
                if(abs(prevWest.val[z] - West.val[z]) == threshold && abs(
                    prevNorthWest.val[z] - NorthWest.val[z]) == threshold &&
                    abs(prevNorth.val[z] - North.val[z]) == threshold && abs(
                    prevNorthEast.val[z] - NorthEast.val[z]) == threshold){
                  initialPrediction = prevPixel.val[z];
                  flag = 1;
                  goto finish1;
                }
              }
            }
          }
          else if(Ydiff < 2 && Xdiff < 2 && Yadd >= h && Xadd < w){
            for(j = 2; j < h; j++){
              for(i = 2; i < Xadd; i++){
                prevNorthNorth = img1.at<Vec3b>(j - 2,i);
                prevNorthWest = img1.at<Vec3b>(j - 1,i - 1);
                prevNorth = img1.at<Vec3b>(j-1,i);
                prevWestWest = img1.at<Vec3b>(j,i - 2);
                prevWest = img1.at<Vec3b>(j,i - 1);
                prevPixel = img1.at<Vec3b>(j,i);
                if (x == w-1){
                  prevNorthNorthEast = 0;
                  prevNorthEast = 0;
                }
                else{
                  prevNorthNorthEast = img1.at<Vec3b>(j - 2,i + 1);
                  prevNorthEast = img1.at<Vec3b>(j - 1,i + 1);
                }
                if(abs(prevWest.val[z] - West.val[z]) == threshold && abs(
                    prevNorthWest.val[z] - NorthWest.val[z]) == threshold &&
                    abs(prevNorth.val[z] - North.val[z]) == threshold && abs(
                    prevNorthEast.val[z] - NorthEast.val[z]) == threshold){
                  initialPrediction = prevPixel.val[z];
                  flag = 1;
                  goto finish1;
                }
              }
```

```
    }
  }
  else if(Ydiff < 2 && Xdiff < 2 && Yadd >= h && Xadd >= w){
    for(j = 2; j < h; j++){
      for(i = 2; i < w; i++){
        prevNorthNorth = img1.at<Vec3b>(j - 2,i);
        prevNorthWest = img1.at<Vec3b>(j - 1,i - 1);
        prevNorth = img1.at<Vec3b>(j-1,i);
        prevWestWest = img1.at<Vec3b>(j,i - 2);
        prevWest = img1.at<Vec3b>(j,i - 1);
        prevPixel = img1.at<Vec3b>(j,i);
        if (x == w-1){
          prevNorthNorthEast = 0;
          prevNorthEast = 0;
        }
        else{
          prevNorthNorthEast = img1.at<Vec3b>(j - 2,i + 1);
          prevNorthEast = img1.at<Vec3b>(j - 1,i + 1);
        }
        if(abs(prevWest.val[z] - West.val[z]) == threshold && abs(
            prevNorthWest.val[z] - NorthWest.val[z]) == threshold &&
            abs(prevNorth.val[z] - North.val[z]) == threshold && abs(
            prevNorthEast.val[z] - NorthEast.val[z]) == threshold){
          initialPrediction = prevPixel.val[z];
          flag = 1;
          goto finish1;
        }
      }
    }
  }
  else if(Ydiff < 2 && Xdiff >= 2 && Yadd < h && Xadd < w){
    for(j = 2; j < Yadd; j++){
      for(i = Xdiff; i < x + distance; i++){
        prevNorthNorth = img1.at<Vec3b>(j - 2,i);
        prevNorthWest = img1.at<Vec3b>(j - 1,i - 1);
        prevNorth = img1.at<Vec3b>(j-1,i);
        prevWestWest = img1.at<Vec3b>(j,i - 2);
```

```
      prevWest = img1.at<Vec3b>(j,i - 1);
      prevPixel = img1.at<Vec3b>(j,i);
      if (x == w-1){
        prevNorthNorthEast = 0;
        prevNorthEast = 0;
      }
      else{
        prevNorthNorthEast = img1.at<Vec3b>(j - 2,i + 1);
        prevNorthEast = img1.at<Vec3b>(j - 1,i + 1);
      }
      if(abs(prevWest.val[z] - West.val[z]) == threshold && abs(
          prevNorthWest.val[z] - NorthWest.val[z]) == threshold &&
          abs(prevNorth.val[z] - North.val[z]) == threshold && abs(
          prevNorthEast.val[z] - NorthEast.val[z]) == threshold){
        initialPrediction = prevPixel.val[z];
        flag = 1;
        goto finish1;
      }
    }
  }
}
else if(Ydiff < 2 && Xdiff >= 2 && Yadd < h && Xadd >= w){
  for(j = 2; j < Yadd; j++){
    for(i = Xdiff; i < w; i++){
      prevNorthNorth = img1.at<Vec3b>(j - 2,i);
      prevNorthWest = img1.at<Vec3b>(j - 1,i - 1);
      prevNorth = img1.at<Vec3b>(j-1,i);
      prevWestWest = img1.at<Vec3b>(j,i - 2);
      prevWest = img1.at<Vec3b>(j,i - 1);
      prevPixel = img1.at<Vec3b>(j,i);
      if (x == w-1){
        prevNorthNorthEast = 0;
        prevNorthEast = 0;
      }
      else{
        prevNorthNorthEast = img1.at<Vec3b>(j - 2,i + 1);
        prevNorthEast = img1.at<Vec3b>(j - 1,i + 1);
```

```
                }
                if(abs(prevWest.val[z] - West.val[z]) == threshold && abs(prevNorthWest.val[z] - NorthWest.val[z]) == threshold && abs(prevNorth.val[z] - North.val[z]) == threshold && abs(prevNorthEast.val[z] - NorthEast.val[z]) == threshold){
                  initialPrediction = prevPixel.val[z];
                  flag = 1;
                  goto finish1;
                }
              }
            }
          }
          else if(Ydiff < 2 && Xdiff >= 2 && Yadd >= h && Xadd < w){
            for(j = 2; j < h; j++){
              for(i = Xdiff; i < Xadd; i++){
                prevNorthNorth = img1.at<Vec3b>(j - 2,i);
                prevNorthWest = img1.at<Vec3b>(j - 1,i - 1);
                prevNorth = img1.at<Vec3b>(j-1,i);
                prevWestWest = img1.at<Vec3b>(j,i - 2);
                prevWest = img1.at<Vec3b>(j,i - 1);
                prevPixel = img1.at<Vec3b>(j,i);
                if (x == w-1){
                  prevNorthNorthEast = 0;
                  prevNorthEast = 0;
                }
                else{
                  prevNorthNorthEast = img1.at<Vec3b>(j - 2,i + 1);
                  prevNorthEast = img1.at<Vec3b>(j - 1,i + 1);
                }
                if(abs(prevWest.val[z] - West.val[z]) == threshold && abs(prevNorthWest.val[z] - NorthWest.val[z]) == threshold && abs(prevNorth.val[z] - North.val[z]) == threshold && abs(prevNorthEast.val[z] - NorthEast.val[z]) == threshold){
                  initialPrediction = prevPixel.val[z];
                  flag = 1;
                  goto finish1;
                }
```

```
                    }
                }
            }
            else if(Ydiff < 2 && Xdiff >= 2 && Yadd >= h && Xadd >= w){
                for(j = 2; j < h; j++){
                    for(i = Xdiff; i < w; i++){
                        prevNorthNorth = img1.at<Vec3b>(j - 2,i);
                        prevNorthWest = img1.at<Vec3b>(j - 1,i - 1);
                        prevNorth = img1.at<Vec3b>(j-1,i);
                        prevWestWest = img1.at<Vec3b>(j,i - 2);
                        prevWest = img1.at<Vec3b>(j,i - 1);
                        prevPixel = img1.at<Vec3b>(j,i);
                        if (x == w-1){
                            prevNorthNorthEast = 0;
                            prevNorthEast = 0;
                        }
                        else{
                            prevNorthNorthEast = img1.at<Vec3b>(j - 2,i + 1);
                            prevNorthEast = img1.at<Vec3b>(j - 1,i + 1);
                        }
                        if(abs(prevWest.val[z] - West.val[z]) == threshold && abs(
                            prevNorthWest.val[z] - NorthWest.val[z]) == threshold &&
                            abs(prevNorth.val[z] - North.val[z]) == threshold && abs(
                            prevNorthEast.val[z] - NorthEast.val[z]) == threshold){
                            initialPrediction = prevPixel.val[z];
                            flag = 1;
                            goto finish1;
                        }
                    }
                }
            }
            else if(Ydiff >= 2 && Xdiff < 2 && Yadd < h && Xadd < w){
                for(j = Ydiff; j < Yadd; j++){
                    for(i = 2; i < Xadd; i++){
                        prevNorthNorth = img1.at<Vec3b>(j - 2,i);
                        prevNorthWest = img1.at<Vec3b>(j - 1,i - 1);
                        prevNorth = img1.at<Vec3b>(j-1,i);
```

```
            prevWestWest = img1.at<Vec3b>(j,i - 2);
            prevWest = img1.at<Vec3b>(j,i - 1);
            prevPixel = img1.at<Vec3b>(j,i);
            if (x == w-1){
              prevNorthNorthEast = 0;
              prevNorthEast = 0;
            }
            else{
              prevNorthNorthEast = img1.at<Vec3b>(j - 2,i + 1);
              prevNorthEast = img1.at<Vec3b>(j - 1,i + 1);
            }
            if(abs(prevWest.val[z] - West.val[z]) == threshold && abs(
                prevNorthWest.val[z] - NorthWest.val[z]) == threshold &&
                abs(prevNorth.val[z] - North.val[z]) == threshold && abs(
                prevNorthEast.val[z] - NorthEast.val[z]) == threshold){
              initialPrediction = prevPixel.val[z];
              flag = 1;
              goto finish1;
            }
          }
        }
      }
      else if(Ydiff >= 2 && Xdiff < 2 && Yadd < h && Xadd >= w){
        for(j = Ydiff; j < Yadd; j++){
          for(i = 2; i < w; i++){
            prevNorthNorth = img1.at<Vec3b>(j - 2,i);
            prevNorthWest = img1.at<Vec3b>(j - 1,i - 1);
            prevNorth = img1.at<Vec3b>(j-1,i);
            prevWestWest = img1.at<Vec3b>(j,i - 2);
            prevWest = img1.at<Vec3b>(j,i - 1);
            prevPixel = img1.at<Vec3b>(j,i);
            if (x == w-1){
              prevNorthNorthEast = 0;
              prevNorthEast = 0;
            }
            else{
              prevNorthNorthEast = img1.at<Vec3b>(j - 2,i + 1);
```

```
                    prevNorthEast = img1.at<Vec3b>(j - 1,i + 1);
                  }
                  if(abs(prevWest.val[z] - West.val[z]) == threshold && abs(
                      prevNorthWest.val[z] - NorthWest.val[z]) == threshold &&
                      abs(prevNorth.val[z] - North.val[z]) == threshold && abs(
                      prevNorthEast.val[z] - NorthEast.val[z]) == threshold){
                    initialPrediction = prevPixel.val[z];
                    flag = 1;
                    goto finish1;
                  }
                }
              }
            }
            else if(Ydiff >= 2 && Xdiff < 2 && Yadd >= h && Xadd < w){
              for(j = Ydiff; j < h; j++){
                for(i = 2; i < Xadd; i++){
                  prevNorthNorth = img1.at<Vec3b>(j - 2,i);
                  prevNorthWest = img1.at<Vec3b>(j - 1,i - 1);
                  prevNorth = img1.at<Vec3b>(j-1,i);
                  prevWestWest = img1.at<Vec3b>(j,i - 2);
                  prevWest = img1.at<Vec3b>(j,i - 1);
                  prevPixel = img1.at<Vec3b>(j,i);
                  if (x == w-1){
                    prevNorthNorthEast = 0;
                    prevNorthEast = 0;
                  }
                  else{
                    prevNorthNorthEast = img1.at<Vec3b>(j - 2,i + 1);
                    prevNorthEast = img1.at<Vec3b>(j - 1,i + 1);
                  }
                  if(abs(prevWest.val[z] - West.val[z]) == threshold && abs(
                      prevNorthWest.val[z] - NorthWest.val[z]) == threshold &&
                      abs(prevNorth.val[z] - North.val[z]) == threshold && abs(
                      prevNorthEast.val[z] - NorthEast.val[z]) == threshold){
                    initialPrediction = prevPixel.val[z];
                    flag = 1;
                    goto finish1;
```

```
                    }
                }
            }
        }
        else if(Ydiff >= 2 && Xdiff < 2 && Yadd >= h && Xadd >= w){
            for(j = Ydiff; j < h; j++){
                for(i = 2; i < w; i++){
                    prevNorthNorth = img1.at<Vec3b>(j - 2,i);
                    prevNorthWest = img1.at<Vec3b>(j - 1,i - 1);
                    prevNorth = img1.at<Vec3b>(j-1,i);
                    prevWestWest = img1.at<Vec3b>(j,i - 2);
                    prevWest = img1.at<Vec3b>(j,i - 1);
                    prevPixel = img1.at<Vec3b>(j,i);
                    if (x == w-1){
                        prevNorthNorthEast = 0;
                        prevNorthEast = 0;
                    }
                    else{
                        prevNorthNorthEast = img1.at<Vec3b>(j - 2,i + 1);
                        prevNorthEast = img1.at<Vec3b>(j - 1,i + 1);
                    }
                    if(abs(prevWest.val[z] - West.val[z]) == threshold && abs(
                        prevNorthWest.val[z] - NorthWest.val[z]) == threshold &&
                        abs(prevNorth.val[z] - North.val[z]) == threshold && abs(
                        prevNorthEast.val[z] - NorthEast.val[z]) == threshold){
                        initialPrediction = prevPixel.val[z];
                        flag = 1;
                        goto finish1;
                    }
                }
            }
        }
        else if(Ydiff >= 2 && Xdiff >= 2 && Yadd < h && Xadd < w){
            for(j = Ydiff; j < Yadd; j++){
                for(i = Xdiff; i < Xadd; i++){
                    prevNorthNorth = img1.at<Vec3b>(j - 2,i);
                    prevNorthWest = img1.at<Vec3b>(j - 1,i - 1);
```

```
                    prevNorth = img1.at<Vec3b>(j-1,i);
                    prevWestWest = img1.at<Vec3b>(j,i - 2);
                    prevWest = img1.at<Vec3b>(j,i - 1);
                    prevPixel = img1.at<Vec3b>(j,i);
                    if (x == w-1){
                      prevNorthNorthEast = 0;
                      prevNorthEast = 0;
                    }
                    else{
                      prevNorthNorthEast = img1.at<Vec3b>(j - 2,i + 1);
                      prevNorthEast = img1.at<Vec3b>(j - 1,i + 1);
                    }
                    if(abs(prevWest.val[z] - West.val[z]) == threshold && abs(
                        prevNorthWest.val[z] - NorthWest.val[z]) == threshold &&
                        abs(prevNorth.val[z] - North.val[z]) == threshold && abs(
                        prevNorthEast.val[z] - NorthEast.val[z]) == threshold){
                      initialPrediction = prevPixel.val[z];
                      flag = 1;
                      goto finish1;
                    }
                  }
                }
              }
              else if(Ydiff >= 2 && Xdiff >= 2 && Yadd < h && Xadd >= w){
                for(j = Ydiff; j < Yadd; j++){
                  for(i = Xdiff; i < w; i++){
                    prevNorthNorth = img1.at<Vec3b>(j - 2,i);
                    prevNorthWest = img1.at<Vec3b>(j - 1,i - 1);
                    prevNorth = img1.at<Vec3b>(j-1,i);
                    prevWestWest = img1.at<Vec3b>(j,i - 2);
                    prevWest = img1.at<Vec3b>(j,i - 1);
                    prevPixel = img1.at<Vec3b>(j,i);
                    if (x == w-1){
                      prevNorthNorthEast = 0;
                      prevNorthEast = 0;
                    }
                    else{
```

```
        prevNorthNorthEast = img1.at<Vec3b>(j - 2,i + 1);
        prevNorthEast = img1.at<Vec3b>(j - 1,i + 1);
      }
      if(abs(prevWest.val[z] - West.val[z]) == threshold && abs(prevNorthWest.val[z] - NorthWest.val[z]) == threshold && abs(prevNorth.val[z] - North.val[z]) == threshold && abs(prevNorthEast.val[z] - NorthEast.val[z]) == threshold){
        initialPrediction = prevPixel.val[z];
        flag = 1;
        goto finish1;
      }
    }
  }
}
else if(Ydiff >= 2 && Xdiff >= 2 && Yadd >= h && Xadd < w){
  for(j = Ydiff; j < h; j++){
    for(i = Xdiff; i < Xadd; i++){
      prevNorthNorth = img1.at<Vec3b>(j - 2,i);
      prevNorthWest = img1.at<Vec3b>(j - 1,i - 1);
      prevNorth = img1.at<Vec3b>(j-1,i);
      prevWestWest = img1.at<Vec3b>(j,i - 2);
      prevWest = img1.at<Vec3b>(j,i - 1);
      prevPixel = img1.at<Vec3b>(j,i);
      if (x == w-1){
        prevNorthNorthEast = 0;
        prevNorthEast = 0;
      }
      else{
        prevNorthNorthEast = img1.at<Vec3b>(j - 2,i + 1);
        prevNorthEast = img1.at<Vec3b>(j - 1,i + 1);
      }
      if(abs(prevWest.val[z] - West.val[z]) == threshold && abs(prevNorthWest.val[z] - NorthWest.val[z]) == threshold && abs(prevNorth.val[z] - North.val[z]) == threshold && abs(prevNorthEast.val[z] - NorthEast.val[z]) == threshold){
        initialPrediction = prevPixel.val[z];
        flag = 1;
```

```
                goto finish1;
              }
            }
          }
        }
        else{ //if(Ydiff >= 0; Xdiff >= 0; Yadd >= h; Xadd >=w)
          for(j = Ydiff; j < h; j++){
            for(i = Xdiff; i < w; i++){
              prevNorthNorth = img1.at<Vec3b>(j - 2,i);
              prevNorthWest = img1.at<Vec3b>(j - 1,i - 1);
              prevNorth = img1.at<Vec3b>(j-1,i);
              prevWestWest = img1.at<Vec3b>(j,i - 2);
              prevWest = img1.at<Vec3b>(j,i - 1);
              prevPixel = img1.at<Vec3b>(j,i);
              if (x == w-1){
                prevNorthNorthEast = 0;
                prevNorthEast = 0;
              }
              else{
                prevNorthNorthEast = img1.at<Vec3b>(j - 2,i + 1);
                prevNorthEast = img1.at<Vec3b>(j - 1,i + 1);
              }
              if(abs(prevWest.val[z] - West.val[z]) == threshold && abs(
                  prevNorthWest.val[z] - NorthWest.val[z]) == threshold &&
                  abs(prevNorth.val[z] - North.val[z]) == threshold && abs(
                  prevNorthEast.val[z] - NorthEast.val[z]) == threshold){
                initialPrediction = prevPixel.val[z];
                flag = 1;
                goto finish1;
              }
            }
          }
        }
        finish1:

        //No match is found, so we have to go through and find the best
            match
```

```
            if(flag == 0){
              threshold ++;
              while(1){
                if(Ydiff < 2 && Xdiff < 2 && Yadd < h && Xadd < w){
                  for(j = 2; j < Yadd; j++){
                    for(i = 2; i < Xadd; i++){
                      prevNorthNorth = img1.at<Vec3b>(j - 2,i);
                      prevNorthWest = img1.at<Vec3b>(j - 1,i - 1);
                      prevNorth = img1.at<Vec3b>(j-1,i);
                      prevWestWest = img1.at<Vec3b>(j,i - 2);
                      prevWest = img1.at<Vec3b>(j,i - 1);
                      prevPixel = img1.at<Vec3b>(j,i);
                      if (x == w-1){
                        prevNorthNorthEast = 0;
                        prevNorthEast = 0;
                      }
                      else{
                        prevNorthNorthEast = img1.at<Vec3b>(j - 2,i + 1);
                        prevNorthEast = img1.at<Vec3b>(j - 1,i + 1);
                      }
                      if(abs(prevWest.val[z] - West.val[z]) <= threshold && abs(
                          prevNorthWest.val[z] - NorthWest.val[z]) <= threshold
                          && abs(prevNorth.val[z] - North.val[z]) <= threshold
                          && abs(prevNorthEast.val[z] - NorthEast.val[z]) <=
                          threshold){
                        initialPrediction = prevPixel.val[z];
                        goto finish2;
                      }
                    }
                  }
                }
                else if(Ydiff < 2 && Xdiff < 2 && Yadd <= h && Xadd >= w){
                  for(j = 2; j < Yadd; j++){
                    for(i = 2; i < w; i++){
                      prevNorthNorth = img1.at<Vec3b>(j - 2,i);
                      prevNorthWest = img1.at<Vec3b>(j - 1,i - 1);
                      prevNorth = img1.at<Vec3b>(j-1,i);
```

```
      prevWestWest = img1.at<Vec3b>(j,i - 2);
      prevWest = img1.at<Vec3b>(j,i - 1);
      prevPixel = img1.at<Vec3b>(j,i);
      if (x == w-1){
        prevNorthNorthEast = 0;
        prevNorthEast = 0;
      }
      else{
        prevNorthNorthEast = img1.at<Vec3b>(j - 2,i + 1);
        prevNorthEast = img1.at<Vec3b>(j - 1,i + 1);
      }
      if(abs(prevWest.val[z] - West.val[z]) <= threshold && abs(
          prevNorthWest.val[z] - NorthWest.val[z]) <= threshold
          && abs(prevNorth.val[z] - North.val[z]) <= threshold
          && abs(prevNorthEast.val[z] - NorthEast.val[z]) <=
          threshold){
        initialPrediction = prevPixel.val[z];
        goto finish2;
      }
    }
  }
}
else if(Ydiff < 2 && Xdiff < 2 && Yadd >= h && Xadd < w){
  for(j = 2; j < h; j++){
    for(i = 2; i < Xadd; i++){
      prevNorthNorth = img1.at<Vec3b>(j - 2,i);
      prevNorthWest = img1.at<Vec3b>(j - 1,i - 1);
      prevNorth = img1.at<Vec3b>(j-1,i);
      prevWestWest = img1.at<Vec3b>(j,i - 2);
      prevWest = img1.at<Vec3b>(j,i - 1);
      prevPixel = img1.at<Vec3b>(j,i);
      if (x == w-1){
        prevNorthNorthEast = 0;
        prevNorthEast = 0;
      }
      else{
        prevNorthNorthEast = img1.at<Vec3b>(j - 2,i + 1);
```

```
                    prevNorthEast = img1.at<Vec3b>(j - 1,i + 1);
                  }
                  if(abs(prevWest.val[z] - West.val[z]) <= threshold && abs(
                      prevNorthWest.val[z] - NorthWest.val[z]) <= threshold
                      && abs(prevNorth.val[z] - North.val[z]) <= threshold
                      && abs(prevNorthEast.val[z] - NorthEast.val[z]) <=
                      threshold){
                    initialPrediction = prevPixel.val[z];
                    goto finish2;
                  }
                }
              }
            }
            else if(Ydiff < 2 && Xdiff < 2 && Yadd >= h && Xadd >= w){
              for(j = 2; j < h; j++){
                for(i = 2; i < w; i++){
                  prevNorthNorth = img1.at<Vec3b>(j - 2,i);
                  prevNorthWest = img1.at<Vec3b>(j - 1,i - 1);
                  prevNorth = img1.at<Vec3b>(j-1,i);
                  prevWestWest = img1.at<Vec3b>(j,i - 2);
                  prevWest = img1.at<Vec3b>(j,i - 1);
                  prevPixel = img1.at<Vec3b>(j,i);
                  if (x == w-1){
                    prevNorthNorthEast = 0;
                    prevNorthEast = 0;
                  }
                  else{
                    prevNorthNorthEast = img1.at<Vec3b>(j - 2,i + 1);
                    prevNorthEast = img1.at<Vec3b>(j - 1,i + 1);
                  }
                  if(abs(prevWest.val[z] - West.val[z]) <= threshold && abs(
                      prevNorthWest.val[z] - NorthWest.val[z]) <= threshold
                      && abs(prevNorth.val[z] - North.val[z]) <= threshold
                      && abs(prevNorthEast.val[z] - NorthEast.val[z]) <=
                      threshold){
                    initialPrediction = prevPixel.val[z];
                    goto finish2;
```

```
            }
        }
    }
}
else if(Ydiff < 2 && Xdiff >= 2 && Yadd < h && Xadd < w){
    for(j = 2; j < Yadd; j++){
        for(i = Xdiff; i < x + distance; i++){
            prevNorthNorth = img1.at<Vec3b>(j - 2,i);
            prevNorthWest = img1.at<Vec3b>(j - 1,i - 1);
            prevNorth = img1.at<Vec3b>(j-1,i);
            prevWestWest = img1.at<Vec3b>(j,i - 2);
            prevWest = img1.at<Vec3b>(j,i - 1);
            prevPixel = img1.at<Vec3b>(j,i);
            if (x == w-1){
                prevNorthNorthEast = 0;
                prevNorthEast = 0;
            }
            else{
                prevNorthNorthEast = img1.at<Vec3b>(j - 2,i + 1);
                prevNorthEast = img1.at<Vec3b>(j - 1,i + 1);
            }
            if(abs(prevWest.val[z] - West.val[z]) <= threshold && abs(prevNorthWest.val[z] - NorthWest.val[z]) <= threshold && abs(prevNorth.val[z] - North.val[z]) <= threshold && abs(prevNorthEast.val[z] - NorthEast.val[z]) <= threshold){
                initialPrediction = prevPixel.val[z];
                goto finish2;
            }
        }
    }
}
else if(Ydiff < 2 && Xdiff >= 2 && Yadd < h && Xadd >= w){
    for(j = 2; j < Yadd; j++){
        for(i = Xdiff; i < w; i++){
            prevNorthNorth = img1.at<Vec3b>(j - 2,i);
            prevNorthWest = img1.at<Vec3b>(j - 1,i - 1);
```

```
                    prevNorth = img1.at<Vec3b>(j-1,i);
                    prevWestWest = img1.at<Vec3b>(j, i - 2);
                    prevWest = img1.at<Vec3b>(j, i - 1);
                    prevPixel = img1.at<Vec3b>(j, i);
                    if (x == w-1){
                      prevNorthNorthEast = 0;
                      prevNorthEast = 0;
                    }
                    else{
                      prevNorthNorthEast = img1.at<Vec3b>(j - 2,i + 1);
                      prevNorthEast = img1.at<Vec3b>(j - 1,i + 1);
                    }
                    if(abs(prevWest.val[z] - West.val[z]) <= threshold && abs(prevNorthWest.val[z] - NorthWest.val[z]) <= threshold && abs(prevNorth.val[z] - North.val[z]) <= threshold && abs(prevNorthEast.val[z] - NorthEast.val[z]) <= threshold){
                      initialPrediction = prevPixel.val[z];
                      goto finish2;
                    }
                  }
                }
              }
              else if(Ydiff < 2 && Xdiff >= 2 && Yadd >= h && Xadd < w){
                for(j = 2; j < h; j++){
                  for(i = Xdiff; i < Xadd; i++){
                    prevNorthNorth = img1.at<Vec3b>(j - 2,i);
                    prevNorthWest = img1.at<Vec3b>(j - 1,i - 1);
                    prevNorth = img1.at<Vec3b>(j-1,i);
                    prevWestWest = img1.at<Vec3b>(j, i - 2);
                    prevWest = img1.at<Vec3b>(j, i - 1);
                    prevPixel = img1.at<Vec3b>(j, i);
                    if (x == w-1){
                      prevNorthNorthEast = 0;
                      prevNorthEast = 0;
                    }
                    else{
```

```
        prevNorthNorthEast = img1.at<Vec3b>(j - 2,i + 1);
        prevNorthEast = img1.at<Vec3b>(j - 1,i + 1);
      }
      if(abs(prevWest.val[z] - West.val[z]) <= threshold && abs(prevNorthWest.val[z] - NorthWest.val[z]) <= threshold && abs(prevNorth.val[z] - North.val[z]) <= threshold && abs(prevNorthEast.val[z] - NorthEast.val[z]) <= threshold){
        initialPrediction = prevPixel.val[z];
        goto finish2;
      }
    }
  }
}
else if(Ydiff < 2 && Xdiff >= 2 && Yadd >= h && Xadd >= w){
  for(j = 2; j < h; j++){
    for(i = Xdiff; i < w; i++){
      prevNorthNorth = img1.at<Vec3b>(j - 2,i);
      prevNorthWest = img1.at<Vec3b>(j - 1,i - 1);
      prevNorth = img1.at<Vec3b>(j-1,i);
      prevWestWest = img1.at<Vec3b>(j,i - 2);
      prevWest = img1.at<Vec3b>(j,i - 1);
      prevPixel = img1.at<Vec3b>(j,i);
      if (x == w-1){
        prevNorthNorthEast = 0;
        prevNorthEast = 0;
      }
      else{
        prevNorthNorthEast = img1.at<Vec3b>(j - 2,i + 1);
        prevNorthEast = img1.at<Vec3b>(j - 1,i + 1);
      }
      if(abs(prevWest.val[z] - West.val[z]) <= threshold && abs(prevNorthWest.val[z] - NorthWest.val[z]) <= threshold && abs(prevNorth.val[z] - North.val[z]) <= threshold && abs(prevNorthEast.val[z] - NorthEast.val[z]) <= threshold){
        initialPrediction = prevPixel.val[z];
```

```
                goto finish2;
              }
            }
          }
        }
        else if(Ydiff >= 2 && Xdiff < 2 && Yadd < h && Xadd < w){
          for(j = Ydiff; j < Yadd; j++){
            for(i = 2; i < Xadd; i++){
              prevNorthNorth = img1.at<Vec3b>(j - 2,i);
              prevNorthWest = img1.at<Vec3b>(j - 1,i - 1);
              prevNorth = img1.at<Vec3b>(j-1,i);
              prevWestWest = img1.at<Vec3b>(j,i - 2);
              prevWest = img1.at<Vec3b>(j,i - 1);
              prevPixel = img1.at<Vec3b>(j,i);
              if (x == w-1){
                prevNorthNorthEast = 0;
                prevNorthEast = 0;
              }
              else{
                prevNorthNorthEast = img1.at<Vec3b>(j - 2,i + 1);
                prevNorthEast = img1.at<Vec3b>(j - 1,i + 1);
              }
              if(abs(prevWest.val[z] - West.val[z]) <= threshold && abs(
                  prevNorthWest.val[z] - NorthWest.val[z]) <= threshold
                  && abs(prevNorth.val[z] - North.val[z]) <= threshold
                  && abs(prevNorthEast.val[z] - NorthEast.val[z]) <=
                  threshold){
                initialPrediction = prevPixel.val[z];
                goto finish2;
              }
            }
          }
        }
        else if(Ydiff >= 2 && Xdiff < 2 && Yadd < h && Xadd >= w){
          for(j = Ydiff; j < Yadd; j++){
            for(i = 2; i < w; i++){
              prevNorthNorth = img1.at<Vec3b>(j - 2,i);
```

```
      prevNorthWest = img1.at<Vec3b>(j - 1,i - 1);
      prevNorth = img1.at<Vec3b>(j-1,i);
      prevWestWest = img1.at<Vec3b>(j,i - 2);
      prevWest = img1.at<Vec3b>(j,i - 1);
      prevPixel = img1.at<Vec3b>(j,i);
      if (x == w-1){
        prevNorthNorthEast = 0;
        prevNorthEast = 0;
      }
      else{
        prevNorthNorthEast = img1.at<Vec3b>(j - 2,i + 1);
        prevNorthEast = img1.at<Vec3b>(j - 1,i + 1);
      }
      if(abs(prevWest.val[z] - West.val[z]) <= threshold && abs(
          prevNorthWest.val[z] - NorthWest.val[z]) <= threshold
          && abs(prevNorth.val[z] - North.val[z]) <= threshold
          && abs(prevNorthEast.val[z] - NorthEast.val[z]) <=
          threshold){
        initialPrediction = prevPixel.val[z];
        goto finish2;
      }
    }
  }
}
else if(Ydiff >= 2 && Xdiff < 2 && Yadd >= h && Xadd < w){
  for(j = Ydiff; j < h; j++){
    for(i = 2; i < Xadd; i++){
      prevNorthNorth = img1.at<Vec3b>(j - 2,i);
      prevNorthWest = img1.at<Vec3b>(j - 1,i - 1);
      prevNorth = img1.at<Vec3b>(j-1,i);
      prevWestWest = img1.at<Vec3b>(j,i - 2);
      prevWest = img1.at<Vec3b>(j,i - 1);
      prevPixel = img1.at<Vec3b>(j,i);
      if (x == w-1){
        prevNorthNorthEast = 0;
        prevNorthEast = 0;
      }
```

```
                    else{
                      prevNorthNorthEast = img1.at<Vec3b>(j - 2,i + 1);
                      prevNorthEast = img1.at<Vec3b>(j - 1,i + 1);
                    }
                    if(abs(prevWest.val[z] - West.val[z]) <= threshold && abs(
                        prevNorthWest.val[z] - NorthWest.val[z]) <= threshold
                        && abs(prevNorth.val[z] - North.val[z]) <= threshold
                        && abs(prevNorthEast.val[z] - NorthEast.val[z]) <=
                        threshold){
                      initialPrediction = prevPixel.val[z];
                      goto finish2;
                    }
                  }
                }
              }
              else if(Ydiff >= 2 && Xdiff < 2 && Yadd >= h && Xadd >= w){
                for(j = Ydiff; j < h; j++){
                  for(i = 2; i < w; i++){
                    prevNorthNorth = img1.at<Vec3b>(j - 2,i);
                    prevNorthWest = img1.at<Vec3b>(j - 1,i - 1);
                    prevNorth = img1.at<Vec3b>(j-1,i);
                    prevWestWest = img1.at<Vec3b>(j,i - 2);
                    prevWest = img1.at<Vec3b>(j,i - 1);
                    prevPixel = img1.at<Vec3b>(j,i);
                    if (x == w-1){
                      prevNorthNorthEast = 0;
                      prevNorthEast = 0;
                    }
                    else{
                      prevNorthNorthEast = img1.at<Vec3b>(j - 2,i + 1);
                      prevNorthEast = img1.at<Vec3b>(j - 1,i + 1);
                    }
                    if(abs(prevWest.val[z] - West.val[z]) <= threshold && abs(
                        prevNorthWest.val[z] - NorthWest.val[z]) <= threshold
                        && abs(prevNorth.val[z] - North.val[z]) <= threshold
                        && abs(prevNorthEast.val[z] - NorthEast.val[z]) <=
                        threshold){
```

```
                initialPrediction = prevPixel.val[z];
                goto finish2;
              }
            }
          }
        }
        else if(Ydiff >= 2 && Xdiff >= 2 && Yadd < h && Xadd < w){
          for(j = Ydiff; j < Yadd; j++){
            for(i = Xdiff; i < Xadd; i++){
              prevNorthNorth = img1.at<Vec3b>(j - 2,i);
              prevNorthWest = img1.at<Vec3b>(j - 1,i - 1);
              prevNorth = img1.at<Vec3b>(j-1,i);
              prevWestWest = img1.at<Vec3b>(j,i - 2);
              prevWest = img1.at<Vec3b>(j,i - 1);
              prevPixel = img1.at<Vec3b>(j,i);
              if (x == w-1){
                prevNorthNorthEast = 0;
                prevNorthEast = 0;
              }
              else{
                prevNorthNorthEast = img1.at<Vec3b>(j - 2,i + 1);
                prevNorthEast = img1.at<Vec3b>(j - 1,i + 1);
              }
              if(abs(prevWest.val[z] - West.val[z]) <= threshold && abs(
                  prevNorthWest.val[z] - NorthWest.val[z]) <= threshold
                  && abs(prevNorth.val[z] - North.val[z]) <= threshold
                  && abs(prevNorthEast.val[z] - NorthEast.val[z]) <=
                  threshold){
                initialPrediction = prevPixel.val[z];
                goto finish2;
              }
            }
          }
        }
        else if(Ydiff >= 2 && Xdiff >= 2 && Yadd < h && Xadd >= w){
          for(j = Ydiff; j < Yadd; j++){
            for(i = Xdiff; i < w; i++){
```

```
      prevNorthNorth = img1.at<Vec3b>(j - 2,i);
      prevNorthWest = img1.at<Vec3b>(j - 1,i - 1);
      prevNorth = img1.at<Vec3b>(j-1,i);
      prevWestWest = img1.at<Vec3b>(j,i - 2);
      prevWest = img1.at<Vec3b>(j,i - 1);
      prevPixel = img1.at<Vec3b>(j,i);
      if (x == w-1){
        prevNorthNorthEast = 0;
        prevNorthEast = 0;
      }
      else{
        prevNorthNorthEast = img1.at<Vec3b>(j - 2,i + 1);
        prevNorthEast = img1.at<Vec3b>(j - 1,i + 1);
      }
      if(abs(prevWest.val[z] - West.val[z]) <= threshold && abs(prevNorthWest.val[z] - NorthWest.val[z]) <= threshold && abs(prevNorth.val[z] - North.val[z]) <= threshold && abs(prevNorthEast.val[z] - NorthEast.val[z]) <= threshold){
        initialPrediction = prevPixel.val[z];
        goto finish2;
      }
    }
  }
}
else if(Ydiff >= 2 && Xdiff >= 2 && Yadd >= h && Xadd < w){
  for(j = Ydiff; j < h; j++){
    for(i = Xdiff; i < Xadd; i++){
      prevNorthNorth = img1.at<Vec3b>(j - 2,i);
      prevNorthWest = img1.at<Vec3b>(j - 1,i - 1);
      prevNorth = img1.at<Vec3b>(j-1,i);
      prevWestWest = img1.at<Vec3b>(j,i - 2);
      prevWest = img1.at<Vec3b>(j,i - 1);
      prevPixel = img1.at<Vec3b>(j,i);
      if (x == w-1){
        prevNorthNorthEast = 0;
        prevNorthEast = 0;
```

```
      }
      else{
        prevNorthNorthEast = img1.at<Vec3b>(j - 2,i + 1);
        prevNorthEast = img1.at<Vec3b>(j - 1,i + 1);
      }
      if(abs(prevWest.val[z] - West.val[z]) <= threshold && abs(prevNorthWest.val[z] - NorthWest.val[z]) <= threshold && abs(prevNorth.val[z] - North.val[z]) <= threshold && abs(prevNorthEast.val[z] - NorthEast.val[z]) <= threshold){
        initialPrediction = prevPixel.val[z];
        goto finish2;
      }
    }
  }
}
else{ //if(Ydiff >= 2; Xdiff >= 2; Yadd >= h; Xadd >=w)
  for(j = Ydiff; j < h; j++){
    for(i = Xdiff; i < w; i++){
      prevNorthNorth = img1.at<Vec3b>(j - 2,i);
      prevNorthWest = img1.at<Vec3b>(j - 1,i - 1);
      prevNorth = img1.at<Vec3b>(j-1,i);
      prevWestWest = img1.at<Vec3b>(j,i - 2);
      prevWest = img1.at<Vec3b>(j,i - 1);
      prevPixel = img1.at<Vec3b>(j,i);
      if (x == w-1){
        prevNorthNorthEast = 0;
        prevNorthEast = 0;
      }
      else{
        prevNorthNorthEast = img1.at<Vec3b>(j - 2,i + 1);
        prevNorthEast = img1.at<Vec3b>(j - 1,i + 1);
      }
      if(abs(prevWest.val[z] - West.val[z]) <= threshold && abs(prevNorthWest.val[z] - NorthWest.val[z]) <= threshold && abs(prevNorth.val[z] - North.val[z]) <= threshold && abs(prevNorthEast.val[z] - NorthEast.val[z]) <=
```

```
                        threshold){
                        initialPrediction = prevPixel.val[z];
                        goto finish2;
                    }
                }
            }
        }
        threshold ++;
      }
    }
    finish2:

    error = int16_t(Pixel.val[z]) - int16_t(initialPrediction);
    //Remap so that all the error values are positive and within the range 0-255
    remap = Remap(error, initialPrediction);
    symbol = pix_to_index_e[remap];
    encode(symbol);

    flag = 0;
    threshold = 0;
   }
  }
 }
}

sendLowerLimit();

//Free up the encoded allocated memory
free(output_array_e);
printf("The file has been encoded\n");
cout << "The final output bit location is : " << outputBitLocation_e << endl;
fclose(encodedFile);

//Now we need to open and read from the encoded file little by little
encodedFile = fopen("encoded.bin","rb");
```

```

//find the size of the file
fseek(encodedFile,0,SEEK_END);
fileSize = ftell(encodedFile);
maxFileLocation = fileSize / fileByteRW;
rewind(encodedFile);

fileLocation = 1;
fread(toDecode,1,fileByteRW,encodedFile);
tag = (toDecode[0] << 8) | toDecode[1];
fseek(encodedFile,fileByteRW*fileLocation,SEEK_SET);
fileLocation++;

//initialize decodedImages
prevDecodedImage.create(h,w,image0.type());
decodedImage.create(h,w,image0.type());
decodedImage0.create(h,w,image0.type());
decodedImage1.create(h,w,image1.type());
decodedImage2.create(h,w,image2.type());
decodedImage3.create(h,w,image3.type());
decodedImage4.create(h,w,image4.type());
decodedImage5.create(h,w,image5.type());

//initialize the tables that translate between symbol indexes and characters
for(int i = 0; i < 256; i++){
  pix_to_index_d[i] = i + 1;
  index_to_pix_d[i + 1] = i;
}

//initialize the symbol counts and cummulative counts
for(int i = 0; i <= 256; i ++){
  symbol_count_d[i] = 1;
  cum_count_d[i] = 256 - i;
}
symbol_count_d[0] = 0; //count[0] must not be the same as count[1]

decode(h, w, c);
```

```

int16_t decodedPrediction;
uint8_t pixelVal;
int16_t e;
Vec3b decodedImagePixel;

//first image was encoded using CALIC.
for(y = 2; y < h; y++){
  for(x = 2; x < w; x++){
    for(z = 0; z < c; z++){
      NorthNorth = decodedImage0.at<Vec3b>(y - 2,x);
      NorthWest = decodedImage0.at<Vec3b>(y - 1,x - 1);
      North = decodedImage0.at<Vec3b>(y-1,x);
      WestWest = decodedImage0.at<Vec3b>(y,x - 2);
      West = decodedImage0.at<Vec3b>(y,x - 1);

      if (x == w-1){
        NorthNorthEast = 0;
        NorthEast = 0;
      }
      else{
        NorthNorthEast = decodedImage0.at<Vec3b>(y - 2,x + 1);
        NorthEast = decodedImage0.at<Vec3b>(y - 1,x + 1);
      }

      decodedImagePixel = decodedImage0.at<Vec3b>(y,x);
      decodedPrediction = initially_predict(NorthNorth.val[z],NorthNorthEast
          .val[z],NorthWest.val[z],North.val[z],NorthEast.val[z],WestWest.
          val[z],West.val[z]);
      e = (signed int16_t)undoRemapping(decodedImagePixel.val[z],
          decodedPrediction);
      decodedImage0.at<Vec3b>(y,x)[z] = decodedPrediction + e;
    }
  }
}

for(int imageCount = 1; imageCount < 6; imageCount++){
```

```
    if(imageCount == 1){
      prevDecodedImage = decodedImage0;
      decodedImage = decodedImage1;
    }
    else if(imageCount == 2){
      prevDecodedImage = decodedImage1;
      decodedImage = decodedImage2;
    }
    else if(imageCount == 3){
      prevDecodedImage = decodedImage2;
      decodedImage = decodedImage3;
    }
    else if(imageCount == 4){
      prevDecodedImage = decodedImage3;
      decodedImage = decodedImage4;
    }
    else if(imageCount == 5){
      prevDecodedImage = decodedImage4;
      decodedImage = decodedImage5;
    }


    for(y = 2; y < h; y++){
      for(x = 2; x < w; x++){
        for(z = 0; z < c; z++){
          NorthNorth = decodedImage.at<Vec3b>(y - 2,x);
          NorthWest = decodedImage.at<Vec3b>(y - 1,x - 1);
          North = decodedImage.at<Vec3b>(y-1,x);
          WestWest = decodedImage.at<Vec3b>(y,x - 2);
          West = decodedImage.at<Vec3b>(y,x - 1);

          if (x == w-1){
            NorthNorthEast = 0;
            NorthEast = 0;
          }
          else{
            NorthNorthEast = decodedImage.at<Vec3b>(y - 2,x + 1);
```

```
            NorthEast = decodedImage.at<Vec3b>(y - 1,x + 1);
          }

          decodedImagePixel = decodedImage.at<Vec3b>(y,x);

          //look for an exact match in a fram around the image
          Ydiff = y - distance;
          Xdiff = x - distance;
          Yadd = y + distance;
          Xadd = x + distance;

          if(Ydiff < 2 && Xdiff < 2 && Yadd < h && Xadd < w){
            for(j = 2; j < Yadd; j++){
              for(i = 2; i < Xadd; i++){
                prevNorthNorth = prevDecodedImage.at<Vec3b>(j - 2,i);
                prevNorthWest = prevDecodedImage.at<Vec3b>(j - 1,i - 1);
                prevNorth = prevDecodedImage.at<Vec3b>(j-1,i);
                prevWestWest = prevDecodedImage.at<Vec3b>(j,i - 2);
                prevWest = prevDecodedImage.at<Vec3b>(j,i - 1);
                prevPixel = prevDecodedImage.at<Vec3b>(j,i);
                if (x == w-1){
                  prevNorthNorthEast = 0;
                  prevNorthEast = 0;
                }
                else{
                  prevNorthNorthEast = prevDecodedImage.at<Vec3b>(j - 2,i + 1);
                  prevNorthEast = prevDecodedImage.at<Vec3b>(j - 1,i + 1);
                }
                if(abs(prevWest.val[z] - West.val[z]) == threshold && abs(prevNorthWest.val[z] - NorthWest.val[z]) == threshold && abs(prevNorth.val[z] - North.val[z]) == threshold && abs(prevNorthEast.val[z] - NorthEast.val[z]) == threshold){
                  decodedPrediction = prevPixel.val[z];
                  flag = 1;
                  goto finish3;
                }
```

```
                }
            }
        }
        else if(Ydiff < 2 && Xdiff < 2 && Yadd <= h && Xadd >= w){
            for(j = 2; j < Yadd; j++){
                for(i = 2; i < w; i++){
                    prevNorthNorth = prevDecodedImage.at<Vec3b>(j - 2,i);
                    prevNorthWest = prevDecodedImage.at<Vec3b>(j - 1,i - 1);
                    prevNorth = prevDecodedImage.at<Vec3b>(j-1,i);
                    prevWestWest = prevDecodedImage.at<Vec3b>(j,i - 2);
                    prevWest = prevDecodedImage.at<Vec3b>(j,i - 1);
                    prevPixel = prevDecodedImage.at<Vec3b>(j,i);
                    if (x == w-1){
                        prevNorthNorthEast = 0;
                        prevNorthEast = 0;
                    }
                    else{
                        prevNorthNorthEast = prevDecodedImage.at<Vec3b>(j - 2,i + 1)
                            ;
                        prevNorthEast = prevDecodedImage.at<Vec3b>(j - 1,i + 1);
                    }
                    if(abs(prevWest.val[z] - West.val[z]) == threshold && abs(
                        prevNorthWest.val[z] - NorthWest.val[z]) == threshold &&
                        abs(prevNorth.val[z] - North.val[z]) == threshold && abs(
                        prevNorthEast.val[z] - NorthEast.val[z]) == threshold){
                        decodedPrediction = prevPixel.val[z];
                        flag = 1;
                        goto finish3;
                    }
                }
            }
        }
        else if(Ydiff < 2 && Xdiff < 2 && Yadd >= h && Xadd < w){
            for(j = 2; j < h; j++){
                for(i = 2; i < Xadd; i++){
                    prevNorthNorth = prevDecodedImage.at<Vec3b>(j - 2,i);
                    prevNorthWest = prevDecodedImage.at<Vec3b>(j - 1,i - 1);
```

```
            prevNorth = prevDecodedImage.at<Vec3b>(j-1,i);
            prevWestWest = prevDecodedImage.at<Vec3b>(j, i - 2);
            prevWest = prevDecodedImage.at<Vec3b>(j, i - 1);
            prevPixel = prevDecodedImage.at<Vec3b>(j, i);
            if (x == w-1){
              prevNorthNorthEast = 0;
              prevNorthEast = 0;
            }
            else{
              prevNorthNorthEast = prevDecodedImage.at<Vec3b>(j - 2,i + 1);
              prevNorthEast = prevDecodedImage.at<Vec3b>(j - 1,i + 1);
            }
            if(abs(prevWest.val[z] - West.val[z]) == threshold && abs(prevNorthWest.val[z] - NorthWest.val[z]) == threshold && abs(prevNorth.val[z] - North.val[z]) == threshold && abs(prevNorthEast.val[z] - NorthEast.val[z]) == threshold){
                decodedPrediction = prevPixel.val[z];
                flag = 1;
                goto finish3;
            }
          }
        }
      }
      else if(Ydiff < 2 && Xdiff < 2 && Yadd >= h && Xadd >= w){
        for(j = 2; j < h; j++){
          for(i = 2; i < w; i++){
            prevNorthNorth = prevDecodedImage.at<Vec3b>(j - 2,i);
            prevNorthWest = prevDecodedImage.at<Vec3b>(j - 1,i - 1);
            prevNorth = prevDecodedImage.at<Vec3b>(j-1,i);
            prevWestWest = prevDecodedImage.at<Vec3b>(j, i - 2);
            prevWest = prevDecodedImage.at<Vec3b>(j, i - 1);
            prevPixel = prevDecodedImage.at<Vec3b>(j, i);
            if (x == w-1){
              prevNorthNorthEast = 0;
              prevNorthEast = 0;
            }
```

```
          else{
            prevNorthNorthEast = prevDecodedImage.at<Vec3b>(j - 2,i + 1);
            prevNorthEast = prevDecodedImage.at<Vec3b>(j - 1,i + 1);
          }
          if(abs(prevWest.val[z] - West.val[z]) == threshold && abs(prevNorthWest.val[z] - NorthWest.val[z]) == threshold && abs(prevNorth.val[z] - North.val[z]) == threshold && abs(prevNorthEast.val[z] - NorthEast.val[z]) == threshold){
            decodedPrediction = prevPixel.val[z];
            flag = 1;
            goto finish3;
          }
        }
      }
    }
    else if(Ydiff < 2 && Xdiff >= 2 && Yadd < h && Xadd < w){
      for(j = 2; j < Yadd; j++){
        for(i = Xdiff; i < x + distance; i++){
          prevNorthNorth = prevDecodedImage.at<Vec3b>(j - 2,i);
          prevNorthWest = prevDecodedImage.at<Vec3b>(j - 1,i - 1);
          prevNorth = prevDecodedImage.at<Vec3b>(j-1,i);
          prevWestWest = prevDecodedImage.at<Vec3b>(j,i - 2);
          prevWest = prevDecodedImage.at<Vec3b>(j,i - 1);
          prevPixel = prevDecodedImage.at<Vec3b>(j,i);
          if (x == w-1){
            prevNorthNorthEast = 0;
            prevNorthEast = 0;
          }
          else{
            prevNorthNorthEast = prevDecodedImage.at<Vec3b>(j - 2,i + 1);
            prevNorthEast = prevDecodedImage.at<Vec3b>(j - 1,i + 1);
          }
          if(abs(prevWest.val[z] - West.val[z]) == threshold && abs(prevNorthWest.val[z] - NorthWest.val[z]) == threshold && abs(prevNorth.val[z] - North.val[z]) == threshold && abs(
```

```
                    prevNorthEast.val[z] - NorthEast.val[z]) == threshold){
                    decodedPrediction = prevPixel.val[z];
                    flag = 1;
                    goto finish3;
                }
            }
        }
    }
    else if(Ydiff < 2 && Xdiff >= 2 && Yadd < h && Xadd >= w){
        for(j = 2; j < Yadd; j++){
            for(i = Xdiff; i < w; i++){
                prevNorthNorth = prevDecodedImage.at<Vec3b>(j - 2,i);
                prevNorthWest = prevDecodedImage.at<Vec3b>(j - 1,i - 1);
                prevNorth = prevDecodedImage.at<Vec3b>(j-1,i);
                prevWestWest = prevDecodedImage.at<Vec3b>(j,i - 2);
                prevWest = prevDecodedImage.at<Vec3b>(j,i - 1);
                prevPixel = prevDecodedImage.at<Vec3b>(j,i);
                if (x == w-1){
                    prevNorthNorthEast = 0;
                    prevNorthEast = 0;
                }
                else{
                    prevNorthNorthEast = prevDecodedImage.at<Vec3b>(j - 2,i + 1)
                        ;
                    prevNorthEast = prevDecodedImage.at<Vec3b>(j - 1,i + 1);
                }
                if(abs(prevWest.val[z] - West.val[z]) == threshold && abs(
                    prevNorthWest.val[z] - NorthWest.val[z]) == threshold &&
                    abs(prevNorth.val[z] - North.val[z]) == threshold && abs(
                    prevNorthEast.val[z] - NorthEast.val[z]) == threshold){
                    decodedPrediction = prevPixel.val[z];
                    flag = 1;
                    goto finish3;
                }
            }
        }
    }
```

```
        else if(Ydiff < 2 && Xdiff >= 2 && Yadd >= h && Xadd < w){
          for(j = 2; j < h; j++){
            for(i = Xdiff; i < Xadd; i++){
              prevNorthNorth = prevDecodedImage.at<Vec3b>(j - 2,i);
              prevNorthWest = prevDecodedImage.at<Vec3b>(j - 1,i - 1);
              prevNorth = prevDecodedImage.at<Vec3b>(j-1,i);
              prevWestWest = prevDecodedImage.at<Vec3b>(j,i - 2);
              prevWest = prevDecodedImage.at<Vec3b>(j,i - 1);
              prevPixel = prevDecodedImage.at<Vec3b>(j,i);
              if (x == w-1){
                prevNorthNorthEast = 0;
                prevNorthEast = 0;
              }
              else{
                prevNorthNorthEast = prevDecodedImage.at<Vec3b>(j - 2,i + 1)
                    ;
                prevNorthEast = prevDecodedImage.at<Vec3b>(j - 1,i + 1);
              }
              if(abs(prevWest.val[z] - West.val[z]) == threshold && abs(
                  prevNorthWest.val[z] - NorthWest.val[z]) == threshold &&
                  abs(prevNorth.val[z] - North.val[z]) == threshold && abs(
                  prevNorthEast.val[z] - NorthEast.val[z]) == threshold){
                decodedPrediction = prevPixel.val[z];
                flag = 1;
                goto finish3;
              }
            }
          }
        }
        else if(Ydiff < 2 && Xdiff >= 2 && Yadd >= h && Xadd >= w){
          for(j = 2; j < h; j++){
            for(i = Xdiff; i < w; i++){
              prevNorthNorth = prevDecodedImage.at<Vec3b>(j - 2,i);
              prevNorthWest = prevDecodedImage.at<Vec3b>(j - 1,i - 1);
              prevNorth = prevDecodedImage.at<Vec3b>(j-1,i);
              prevWestWest = prevDecodedImage.at<Vec3b>(j,i - 2);
              prevWest = prevDecodedImage.at<Vec3b>(j,i - 1);
```

```
                prevPixel = prevDecodedImage.at<Vec3b>(j,i);
                if (x == w-1){
                  prevNorthNorthEast = 0;
                  prevNorthEast = 0;
                }
                else{
                  prevNorthNorthEast = prevDecodedImage.at<Vec3b>(j - 2,i + 1);
                  prevNorthEast = prevDecodedImage.at<Vec3b>(j - 1,i + 1);
                }
                if(abs(prevWest.val[z] - West.val[z]) == threshold && abs(prevNorthWest.val[z] - NorthWest.val[z]) == threshold && abs(prevNorth.val[z] - North.val[z]) == threshold && abs(prevNorthEast.val[z] - NorthEast.val[z]) == threshold){
                    decodedPrediction = prevPixel.val[z];
                    flag = 1;
                    goto finish3;
                }
              }
            }
          }
          else if(Ydiff >= 2 && Xdiff < 2 && Yadd < h && Xadd < w){
            for(j = Ydiff; j < Yadd; j++){
              for(i = 2; i < Xadd; i++){
                prevNorthNorth = prevDecodedImage.at<Vec3b>(j - 2,i);
                prevNorthWest = prevDecodedImage.at<Vec3b>(j - 1,i - 1);
                prevNorth = prevDecodedImage.at<Vec3b>(j-1,i);
                prevWestWest = prevDecodedImage.at<Vec3b>(j,i - 2);
                prevWest = prevDecodedImage.at<Vec3b>(j,i - 1);
                prevPixel = prevDecodedImage.at<Vec3b>(j,i);
                if (x == w-1){
                  prevNorthNorthEast = 0;
                  prevNorthEast = 0;
                }
                else{
                  prevNorthNorthEast = prevDecodedImage.at<Vec3b>(j - 2,i + 1);
```

```
                prevNorthEast = prevDecodedImage.at<Vec3b>(j - 1,i + 1);
              }
              if(abs(prevWest.val[z] - West.val[z]) == threshold && abs(prevNorthWest.val[z] - NorthWest.val[z]) == threshold && abs(prevNorth.val[z] - North.val[z]) == threshold && abs(prevNorthEast.val[z] - NorthEast.val[z]) == threshold){
                decodedPrediction = prevPixel.val[z];
                flag = 1;
                goto finish3;
              }
            }
          }
        }
        else if(Ydiff >= 2 && Xdiff < 2 && Yadd < h && Xadd >= w){
          for(j = Ydiff; j < Yadd; j++){
            for(i = 2; i < w; i++){
              prevNorthNorth = prevDecodedImage.at<Vec3b>(j - 2,i);
              prevNorthWest = prevDecodedImage.at<Vec3b>(j - 1,i - 1);
              prevNorth = prevDecodedImage.at<Vec3b>(j-1,i);
              prevWestWest = prevDecodedImage.at<Vec3b>(j,i - 2);
              prevWest = prevDecodedImage.at<Vec3b>(j,i - 1);
              prevPixel = prevDecodedImage.at<Vec3b>(j,i);
              if (x == w-1){
                prevNorthNorthEast = 0;
                prevNorthEast = 0;
              }
              else{
                prevNorthNorthEast = prevDecodedImage.at<Vec3b>(j - 2,i + 1);
                prevNorthEast = prevDecodedImage.at<Vec3b>(j - 1,i + 1);
              }
              if(abs(prevWest.val[z] - West.val[z]) == threshold && abs(prevNorthWest.val[z] - NorthWest.val[z]) == threshold && abs(prevNorth.val[z] - North.val[z]) == threshold && abs(prevNorthEast.val[z] - NorthEast.val[z]) == threshold){
                decodedPrediction = prevPixel.val[z];
                flag = 1;
```

```
                goto finish3;
              }
            }
          }
        }
        else if(Ydiff >= 2 && Xdiff < 2 && Yadd >= h && Xadd < w){
          for(j = Ydiff; j < h; j++){
            for(i = 2; i < Xadd; i++){
              prevNorthNorth = prevDecodedImage.at<Vec3b>(j - 2,i);
              prevNorthWest = prevDecodedImage.at<Vec3b>(j - 1,i - 1);
              prevNorth = prevDecodedImage.at<Vec3b>(j-1,i);
              prevWestWest = prevDecodedImage.at<Vec3b>(j,i - 2);
              prevWest = prevDecodedImage.at<Vec3b>(j,i - 1);
              prevPixel = prevDecodedImage.at<Vec3b>(j,i);
              if (x == w-1){
                prevNorthNorthEast = 0;
                prevNorthEast = 0;
              }
              else{
                prevNorthNorthEast = prevDecodedImage.at<Vec3b>(j - 2,i + 1)
                    ;
                prevNorthEast = prevDecodedImage.at<Vec3b>(j - 1,i + 1);
              }
              if(abs(prevWest.val[z] - West.val[z]) == threshold && abs(
                  prevNorthWest.val[z] - NorthWest.val[z]) == threshold &&
                  abs(prevNorth.val[z] - North.val[z]) == threshold && abs(
                  prevNorthEast.val[z] - NorthEast.val[z]) == threshold){
                decodedPrediction = prevPixel.val[z];
                flag = 1;
                goto finish3;
              }
            }
          }
        }
        else if(Ydiff >= 2 && Xdiff < 2 && Yadd >= h && Xadd >= w){
          for(j = Ydiff; j < h; j++){
            for(i = 2; i < w; i++){
```

```
      prevNorthNorth = prevDecodedImage.at<Vec3b>(j - 2,i);
      prevNorthWest = prevDecodedImage.at<Vec3b>(j - 1,i - 1);
      prevNorth = prevDecodedImage.at<Vec3b>(j-1,i);
      prevWestWest = prevDecodedImage.at<Vec3b>(j,i - 2);
      prevWest = prevDecodedImage.at<Vec3b>(j,i - 1);
      prevPixel = prevDecodedImage.at<Vec3b>(j,i);
      if (x == w-1){
        prevNorthNorthEast = 0;
        prevNorthEast = 0;
      }
      else{
        prevNorthNorthEast = prevDecodedImage.at<Vec3b>(j - 2,i + 1);
        prevNorthEast = prevDecodedImage.at<Vec3b>(j - 1,i + 1);
      }
      if(abs(prevWest.val[z] - West.val[z]) == threshold && abs(prevNorthWest.val[z] - NorthWest.val[z]) == threshold && abs(prevNorth.val[z] - North.val[z]) == threshold && abs(prevNorthEast.val[z] - NorthEast.val[z]) == threshold){
          decodedPrediction = prevPixel.val[z];
          flag = 1;
          goto finish3;
      }
    }
  }
}
else if(Ydiff >= 2 && Xdiff >= 2 && Yadd < h && Xadd < w){
  for(j = Ydiff; j < Yadd; j++){
    for(i = Xdiff; i < Xadd; i++){
      prevNorthNorth = prevDecodedImage.at<Vec3b>(j - 2,i);
      prevNorthWest = prevDecodedImage.at<Vec3b>(j - 1,i - 1);
      prevNorth = prevDecodedImage.at<Vec3b>(j-1,i);
      prevWestWest = prevDecodedImage.at<Vec3b>(j,i - 2);
      prevWest = prevDecodedImage.at<Vec3b>(j,i - 1);
      prevPixel = prevDecodedImage.at<Vec3b>(j,i);
      if (x == w-1){
        prevNorthNorthEast = 0;
```

```
                prevNorthEast = 0;
              }
              else{
                prevNorthNorthEast = prevDecodedImage.at<Vec3b>(j - 2,i + 1)
                    ;
                prevNorthEast = prevDecodedImage.at<Vec3b>(j - 1,i + 1);
              }
              if(abs(prevWest.val[z] - West.val[z]) == threshold && abs(
                  prevNorthWest.val[z] - NorthWest.val[z]) == threshold &&
                  abs(prevNorth.val[z] - North.val[z]) == threshold && abs(
                  prevNorthEast.val[z] - NorthEast.val[z]) == threshold){
                decodedPrediction = prevPixel.val[z];
                flag = 1;
                goto finish3;
              }
            }
          }
        }
        else if(Ydiff >= 2 && Xdiff >= 2 && Yadd < h && Xadd >= w){
          for(j = Ydiff; j < Yadd; j++){
            for(i = Xdiff; i < w; i++){
              prevNorthNorth = prevDecodedImage.at<Vec3b>(j - 2,i);
              prevNorthWest = prevDecodedImage.at<Vec3b>(j - 1,i - 1);
              prevNorth = prevDecodedImage.at<Vec3b>(j-1,i);
              prevWestWest = prevDecodedImage.at<Vec3b>(j,i - 2);
              prevWest = prevDecodedImage.at<Vec3b>(j,i - 1);
              prevPixel = prevDecodedImage.at<Vec3b>(j,i);
              if (x == w-1){
                prevNorthNorthEast = 0;
                prevNorthEast = 0;
              }
              else{
                prevNorthNorthEast = prevDecodedImage.at<Vec3b>(j - 2,i + 1)
                    ;
                prevNorthEast = prevDecodedImage.at<Vec3b>(j - 1,i + 1);
              }
```

```
                if(abs(prevWest.val[z] - West.val[z]) == threshold && abs(prevNorthWest.val[z] - NorthWest.val[z]) == threshold && abs(prevNorth.val[z] - North.val[z]) == threshold && abs(prevNorthEast.val[z] - NorthEast.val[z]) == threshold){
                    decodedPrediction = prevPixel.val[z];
                    flag = 1;
                    goto finish3;
                }
            }
        }
    }
    else if(Ydiff >= 2 && Xdiff >= 2 && Yadd >= h && Xadd < w){
        for(j = Ydiff; j < h; j++){
            for(i = Xdiff; i < Xadd; i++){
                prevNorthNorth = prevDecodedImage.at<Vec3b>(j - 2,i);
                prevNorthWest = prevDecodedImage.at<Vec3b>(j - 1,i - 1);
                prevNorth = prevDecodedImage.at<Vec3b>(j-1,i);
                prevWestWest = prevDecodedImage.at<Vec3b>(j,i - 2);
                prevWest = prevDecodedImage.at<Vec3b>(j,i - 1);
                prevPixel = prevDecodedImage.at<Vec3b>(j,i);
                if (x == w-1){
                    prevNorthNorthEast = 0;
                    prevNorthEast = 0;
                }
                else{
                    prevNorthNorthEast = prevDecodedImage.at<Vec3b>(j - 2,i + 1);
                    prevNorthEast = prevDecodedImage.at<Vec3b>(j - 1,i + 1);
                }
                if(abs(prevWest.val[z] - West.val[z]) == threshold && abs(prevNorthWest.val[z] - NorthWest.val[z]) == threshold && abs(prevNorth.val[z] - North.val[z]) == threshold && abs(prevNorthEast.val[z] - NorthEast.val[z]) == threshold){
                    decodedPrediction = prevPixel.val[z];
                    flag = 1;
                    goto finish3;
                }
```

```
                }
            }
        }
        else{ //if(Ydiff >= 2; Xdiff >= 2; Yadd >= h; Xadd >=w)
            for(j = Ydiff; j < h; j++){
                for(i = Xdiff; i < w; i++){
                    prevNorthNorth = prevDecodedImage.at<Vec3b>(j - 2,i);
                    prevNorthWest = prevDecodedImage.at<Vec3b>(j - 1,i - 1);
                    prevNorth = prevDecodedImage.at<Vec3b>(j-1,i);
                    prevWestWest = prevDecodedImage.at<Vec3b>(j,i - 2);
                    prevWest = prevDecodedImage.at<Vec3b>(j,i - 1);
                    prevPixel = prevDecodedImage.at<Vec3b>(j,i);
                    if (x == w-1){
                        prevNorthNorthEast = 0;
                        prevNorthEast = 0;
                    }
                    else{
                        prevNorthNorthEast = prevDecodedImage.at<Vec3b>(j - 2,i + 1);
                        prevNorthEast = prevDecodedImage.at<Vec3b>(j - 1,i + 1);
                    }
                    if(abs(prevWest.val[z] - West.val[z]) == threshold && abs(prevNorthWest.val[z] - NorthWest.val[z]) == threshold && abs(prevNorth.val[z] - North.val[z]) == threshold && abs(prevNorthEast.val[z] - NorthEast.val[z]) == threshold){
                        decodedPrediction = prevPixel.val[z];
                        flag = 1;
                        goto finish3;
                    }
                }
            }
        }
        finish3:

        //No match is found, so we have to go through and find the best match
        if(flag == 0){
```

```
threshold ++;
while(1){
  if(Ydiff < 2 && Xdiff < 2 && Yadd < h && Xadd < w){
    for(j = 2; j < Yadd; j++){
      for(i = 2; i < Xadd; i++){
        prevNorthNorth = prevDecodedImage.at<Vec3b>(j - 2,i);
        prevNorthWest = prevDecodedImage.at<Vec3b>(j - 1,i - 1);
        prevNorth = prevDecodedImage.at<Vec3b>(j-1,i);
        prevWestWest = prevDecodedImage.at<Vec3b>(j,i - 2);
        prevWest = prevDecodedImage.at<Vec3b>(j,i - 1);
        prevPixel = prevDecodedImage.at<Vec3b>(j,i);
        if (x == w-1){
          prevNorthNorthEast = 0;
          prevNorthEast = 0;
        }
        else{
          prevNorthNorthEast = prevDecodedImage.at<Vec3b>(j - 2,i + 1);
          prevNorthEast = prevDecodedImage.at<Vec3b>(j - 1,i + 1);
        }
        if(abs(prevWest.val[z] - West.val[z]) <= threshold && abs(prevNorthWest.val[z] - NorthWest.val[z]) <= threshold && abs(prevNorth.val[z] - North.val[z]) <= threshold && abs(prevNorthEast.val[z] - NorthEast.val[z]) <= threshold){
          decodedPrediction = prevPixel.val[z];
          flag = 1;
          goto finish4;
        }
      }
    }
  }
  else if(Ydiff < 2 && Xdiff < 2 && Yadd <= h && Xadd >= w){
    for(j = 2; j < Yadd; j++){
      for(i = 2; i < w; i++){
        prevNorthNorth = prevDecodedImage.at<Vec3b>(j - 2,i);
        prevNorthWest = prevDecodedImage.at<Vec3b>(j - 1,i - 1);
```

```
                    prevNorth = prevDecodedImage.at<Vec3b>(j-1,i);
                    prevWestWest = prevDecodedImage.at<Vec3b>(j, i - 2);
                    prevWest = prevDecodedImage.at<Vec3b>(j, i - 1);
                    prevPixel = prevDecodedImage.at<Vec3b>(j, i);
                    if (x == w-1){
                      prevNorthNorthEast = 0;
                      prevNorthEast = 0;
                    }
                    else{
                      prevNorthNorthEast = prevDecodedImage.at<Vec3b>(j - 2,i + 1);
                      prevNorthEast = prevDecodedImage.at<Vec3b>(j - 1,i + 1);
                    }
                    if(abs(prevWest.val[z] - West.val[z]) <= threshold && abs(prevNorthWest.val[z] - NorthWest.val[z]) <= threshold && abs(prevNorth.val[z] - North.val[z]) <= threshold && abs(prevNorthEast.val[z] - NorthEast.val[z]) <= threshold){
                      decodedPrediction = prevPixel.val[z];
                      flag = 1;
                      goto finish4;
                    }
                  }
                }
              }
              else if(Ydiff < 2 && Xdiff < 2 && Yadd >= h && Xadd < w){
                for(j = 2; j < h; j++){
                  for(i = 2; i < Xadd; i++){
                    prevNorthNorth = prevDecodedImage.at<Vec3b>(j - 2,i);
                    prevNorthWest = prevDecodedImage.at<Vec3b>(j - 1,i - 1);
                    prevNorth = prevDecodedImage.at<Vec3b>(j-1,i);
                    prevWestWest = prevDecodedImage.at<Vec3b>(j, i - 2);
                    prevWest = prevDecodedImage.at<Vec3b>(j, i - 1);
                    prevPixel = prevDecodedImage.at<Vec3b>(j, i);
                    if (x == w-1){
                      prevNorthNorthEast = 0;
                      prevNorthEast = 0;
```

```
                    }
                    else{
                      prevNorthNorthEast = prevDecodedImage.at<Vec3b>(j - 2,i
                          + 1);
                      prevNorthEast = prevDecodedImage.at<Vec3b>(j - 1,i + 1);
                    }
                    if(abs(prevWest.val[z] - West.val[z]) <= threshold && abs(
                        prevNorthWest.val[z] - NorthWest.val[z]) <= threshold
                        && abs(prevNorth.val[z] - North.val[z]) <= threshold
                        && abs(prevNorthEast.val[z] - NorthEast.val[z]) <=
                        threshold){
                      decodedPrediction = prevPixel.val[z];
                      flag = 1;
                      goto finish4;
                    }
                  }
                }
              }
              else if(Ydiff < 2 && Xdiff < 2 && Yadd >= h && Xadd >= w){
                for(j = 2; j < h; j++){
                  for(i = 2; i < w; i++){
                    prevNorthNorth = prevDecodedImage.at<Vec3b>(j - 2,i);
                    prevNorthWest = prevDecodedImage.at<Vec3b>(j - 1,i - 1);
                    prevNorth = prevDecodedImage.at<Vec3b>(j-1,i);
                    prevWestWest = prevDecodedImage.at<Vec3b>(j,i - 2);
                    prevWest = prevDecodedImage.at<Vec3b>(j,i - 1);
                    prevPixel = prevDecodedImage.at<Vec3b>(j,i);
                    if (x == w-1){
                      prevNorthNorthEast = 0;
                      prevNorthEast = 0;
                    }
                    else{
                      prevNorthNorthEast = prevDecodedImage.at<Vec3b>(j - 2,i
                          + 1);
                      prevNorthEast = prevDecodedImage.at<Vec3b>(j - 1,i + 1);
                    }
```

```
                        if(abs(prevWest.val[z] - West.val[z]) <= threshold && abs(
                            prevNorthWest.val[z] - NorthWest.val[z]) <= threshold &&
                             abs(prevNorth.val[z] - North.val[z]) <= threshold &&
                            abs(prevNorthEast.val[z] - NorthEast.val[z]) <=
                            threshold){
                            decodedPrediction = prevPixel.val[z];
                            flag = 1;
                            goto finish4;
                        }
                    }
                }
            }
            else if(Ydiff < 2 && Xdiff >= 2 && Yadd < h && Xadd < w){
                for(j = 2; j < Yadd; j++){
                    for(i = Xdiff; i < x + distance; i++){
                        prevNorthNorth = prevDecodedImage.at<Vec3b>(j - 2,i);
                        prevNorthWest = prevDecodedImage.at<Vec3b>(j - 1,i - 1);
                        prevNorth = prevDecodedImage.at<Vec3b>(j-1,i);
                        prevWestWest = prevDecodedImage.at<Vec3b>(j,i - 2);
                        prevWest = prevDecodedImage.at<Vec3b>(j,i - 1);
                        prevPixel = prevDecodedImage.at<Vec3b>(j,i);
                        if (x == w-1){
                            prevNorthNorthEast = 0;
                            prevNorthEast = 0;
                        }
                        else{
                            prevNorthNorthEast = prevDecodedImage.at<Vec3b>(j - 2,i
                                + 1);
                            prevNorthEast = prevDecodedImage.at<Vec3b>(j - 1,i + 1);
                        }
                        if(abs(prevWest.val[z] - West.val[z]) <= threshold && abs(
                            prevNorthWest.val[z] - NorthWest.val[z]) <= threshold
                            && abs(prevNorth.val[z] - North.val[z]) <= threshold
                            && abs(prevNorthEast.val[z] - NorthEast.val[z]) <=
                            threshold){
                            decodedPrediction = prevPixel.val[z];
                            flag = 1;
```

```
                goto finish4;
              }
            }
          }
        }
        else if(Ydiff < 2 && Xdiff >= 2 && Yadd < h && Xadd >= w){
          for(j = 2; j < Yadd; j++){
            for(i = Xdiff; i < w; i++){
              prevNorthNorth = prevDecodedImage.at<Vec3b>(j - 2,i);
              prevNorthWest = prevDecodedImage.at<Vec3b>(j - 1,i - 1);
              prevNorth = prevDecodedImage.at<Vec3b>(j-1,i);
              prevWestWest = prevDecodedImage.at<Vec3b>(j,i - 2);
              prevWest = prevDecodedImage.at<Vec3b>(j,i - 1);
              prevPixel = prevDecodedImage.at<Vec3b>(j,i);
              if (x == w-1){
                prevNorthNorthEast = 0;
                prevNorthEast = 0;
              }
              else{
                prevNorthNorthEast = prevDecodedImage.at<Vec3b>(j - 2,i
                    + 1);
                prevNorthEast = prevDecodedImage.at<Vec3b>(j - 1,i + 1);
              }
              if(abs(prevWest.val[z] - West.val[z]) <= threshold && abs(
                  prevNorthWest.val[z] - NorthWest.val[z]) <= threshold
                  && abs(prevNorth.val[z] - North.val[z]) <= threshold
                  && abs(prevNorthEast.val[z] - NorthEast.val[z]) <=
                  threshold){
                decodedPrediction = prevPixel.val[z];
                flag = 1;
                goto finish4;
              }
            }
          }
        }
        else if(Ydiff < 2 && Xdiff >= 2 && Yadd >= h && Xadd < w){
          for(j = 2; j < h; j++){
```

```
                for(i = Xdiff; i < Xadd; i++){
                  prevNorthNorth = prevDecodedImage.at<Vec3b>(j - 2,i);
                  prevNorthWest = prevDecodedImage.at<Vec3b>(j - 1,i - 1);
                  prevNorth = prevDecodedImage.at<Vec3b>(j-1,i);
                  prevWestWest = prevDecodedImage.at<Vec3b>(j,i - 2);
                  prevWest = prevDecodedImage.at<Vec3b>(j,i - 1);
                  prevPixel = prevDecodedImage.at<Vec3b>(j,i);
                  if (x == w-1){
                    prevNorthNorthEast = 0;
                    prevNorthEast = 0;
                  }
                  else{
                    prevNorthNorthEast = prevDecodedImage.at<Vec3b>(j - 2,i
                        + 1);
                    prevNorthEast = prevDecodedImage.at<Vec3b>(j - 1,i + 1);
                  }
                  if(abs(prevWest.val[z] - West.val[z]) <= threshold && abs(
                      prevNorthWest.val[z] - NorthWest.val[z]) <= threshold
                      && abs(prevNorth.val[z] - North.val[z]) <= threshold
                      && abs(prevNorthEast.val[z] - NorthEast.val[z]) <=
                      threshold){
                    decodedPrediction = prevPixel.val[z];
                    flag = 1;
                    goto finish4;
                  }
                }
              }
            }
            else if(Ydiff < 2 && Xdiff >= 2 && Yadd >= h && Xadd >= w){
              for(j = 2; j < h; j++){
                for(i = Xdiff; i < w; i++){
                  prevNorthNorth = prevDecodedImage.at<Vec3b>(j - 2,i);
                  prevNorthWest = prevDecodedImage.at<Vec3b>(j - 1,i - 1);
                  prevNorth = prevDecodedImage.at<Vec3b>(j-1,i);
                  prevWestWest = prevDecodedImage.at<Vec3b>(j,i - 2);
                  prevWest = prevDecodedImage.at<Vec3b>(j,i - 1);
                  prevPixel = prevDecodedImage.at<Vec3b>(j,i);
```

```
                    if (x == w-1){
                        prevNorthNorthEast = 0;
                        prevNorthEast = 0;
                    }
                    else{
                        prevNorthNorthEast = prevDecodedImage.at<Vec3b>(j - 2,i + 1);
                        prevNorthEast = prevDecodedImage.at<Vec3b>(j - 1,i + 1);
                    }
                    if(abs(prevWest.val[z] - West.val[z]) <= threshold && abs(prevNorthWest.val[z] - NorthWest.val[z]) <= threshold && abs(prevNorth.val[z] - North.val[z]) <= threshold && abs(prevNorthEast.val[z] - NorthEast.val[z]) <= threshold){
                        decodedPrediction = prevPixel.val[z];
                        flag = 1;
                        goto finish4;
                    }
                }
            }
        }
        else if(Ydiff >= 2 && Xdiff < 2 && Yadd < h && Xadd < w){
            for(j = Ydiff; j < Yadd; j++){
                for(i = 2; i < Xadd; i++){
                    prevNorthNorth = prevDecodedImage.at<Vec3b>(j - 2,i);
                    prevNorthWest = prevDecodedImage.at<Vec3b>(j - 1,i - 1);
                    prevNorth = prevDecodedImage.at<Vec3b>(j-1,i);
                    prevWestWest = prevDecodedImage.at<Vec3b>(j,i - 2);
                    prevWest = prevDecodedImage.at<Vec3b>(j,i - 1);
                    prevPixel = prevDecodedImage.at<Vec3b>(j,i);
                    if (x == w-1){
                        prevNorthNorthEast = 0;
                        prevNorthEast = 0;
                    }
                    else{
                        prevNorthNorthEast = prevDecodedImage.at<Vec3b>(j - 2,i + 1);
```

```
                    prevNorthEast = prevDecodedImage.at<Vec3b>(j - 1,i + 1);
                  }
                  if(abs(prevWest.val[z] - West.val[z]) <= threshold && abs(
                      prevNorthWest.val[z] - NorthWest.val[z]) <= threshold
                      && abs(prevNorth.val[z] - North.val[z]) <= threshold
                      && abs(prevNorthEast.val[z] - NorthEast.val[z]) <=
                      threshold){
                    decodedPrediction = prevPixel.val[z];
                    flag = 1;
                    goto finish4;
                  }
                }
              }
            }
            else if(Ydiff >= 2 && Xdiff < 2 && Yadd < h && Xadd >= w){
              for(j = Ydiff; j < Yadd; j++){
                for(i = 2; i < w; i++){
                  prevNorthNorth = prevDecodedImage.at<Vec3b>(j - 2,i);
                  prevNorthWest = prevDecodedImage.at<Vec3b>(j - 1,i - 1);
                  prevNorth = prevDecodedImage.at<Vec3b>(j-1,i);
                  prevWestWest = prevDecodedImage.at<Vec3b>(j,i - 2);
                  prevWest = prevDecodedImage.at<Vec3b>(j,i - 1);
                  prevPixel = prevDecodedImage.at<Vec3b>(j,i);
                  if (x == w-1){
                    prevNorthNorthEast = 0;
                    prevNorthEast = 0;
                  }
                  else{
                    prevNorthNorthEast = prevDecodedImage.at<Vec3b>(j - 2,i
                        + 1);
                    prevNorthEast = prevDecodedImage.at<Vec3b>(j - 1,i + 1);
                  }
                  if(abs(prevWest.val[z] - West.val[z]) <= threshold && abs(
                      prevNorthWest.val[z] - NorthWest.val[z]) <= threshold
                      && abs(prevNorth.val[z] - North.val[z]) <= threshold
                      && abs(prevNorthEast.val[z] - NorthEast.val[z]) <=
                      threshold){
```

```
                        decodedPrediction = prevPixel.val[z];
                        flag = 1;
                        goto finish4;
                    }
                }
            }
        }
        else if(Ydiff >= 2 && Xdiff < 2 && Yadd >= h && Xadd < w){
            for(j = Ydiff; j < h; j++){
                for(i = 2; i < Xadd; i++){
                    prevNorthNorth = prevDecodedImage.at<Vec3b>(j - 2,i);
                    prevNorthWest = prevDecodedImage.at<Vec3b>(j - 1,i - 1);
                    prevNorth = prevDecodedImage.at<Vec3b>(j-1,i);
                    prevWestWest = prevDecodedImage.at<Vec3b>(j,i - 2);
                    prevWest = prevDecodedImage.at<Vec3b>(j,i - 1);
                    prevPixel = prevDecodedImage.at<Vec3b>(j,i);
                    if (x == w-1){
                        prevNorthNorthEast = 0;
                        prevNorthEast = 0;
                    }
                    else{
                        prevNorthNorthEast = prevDecodedImage.at<Vec3b>(j - 2,i
                            + 1);
                        prevNorthEast = prevDecodedImage.at<Vec3b>(j - 1,i + 1);
                    }
                    if(abs(prevWest.val[z] - West.val[z]) <= threshold && abs(
                        prevNorthWest.val[z] - NorthWest.val[z]) <= threshold
                        && abs(prevNorth.val[z] - North.val[z]) <= threshold
                        && abs(prevNorthEast.val[z] - NorthEast.val[z]) <=
                        threshold){
                        decodedPrediction = prevPixel.val[z];
                        flag = 1;
                        goto finish4;
                    }
                }
            }
        }
```

```
        else if(Ydiff >= 2 && Xdiff < 2 && Yadd >= h && Xadd >= w){
          for(j = Ydiff; j < h; j++){
            for(i = 2; i < w; i++){
              prevNorthNorth = prevDecodedImage.at<Vec3b>(j - 2,i);
              prevNorthWest = prevDecodedImage.at<Vec3b>(j - 1,i - 1);
              prevNorth = prevDecodedImage.at<Vec3b>(j-1,i);
              prevWestWest = prevDecodedImage.at<Vec3b>(j,i - 2);
              prevWest = prevDecodedImage.at<Vec3b>(j,i - 1);
              prevPixel = prevDecodedImage.at<Vec3b>(j,i);
              if (x == w-1){
                prevNorthNorthEast = 0;
                prevNorthEast = 0;
              }
              else{
                prevNorthNorthEast = prevDecodedImage.at<Vec3b>(j - 2,i
                    + 1);
                prevNorthEast = prevDecodedImage.at<Vec3b>(j - 1,i + 1);
              }
              if(abs(prevWest.val[z] - West.val[z]) <= threshold && abs(
                  prevNorthWest.val[z] - NorthWest.val[z]) <= threshold
                  && abs(prevNorth.val[z] - North.val[z]) <= threshold
                  && abs(prevNorthEast.val[z] - NorthEast.val[z]) <=
                  threshold){
                decodedPrediction = prevPixel.val[z];
                flag = 1;
                goto finish4;
              }
            }
          }
        }
        else if(Ydiff >= 2 && Xdiff >= 2 && Yadd < h && Xadd < w){
          for(j = Ydiff; j < Yadd; j++){
            for(i = Xdiff; i < Xadd; i++){
              prevNorthNorth = prevDecodedImage.at<Vec3b>(j - 2,i);
              prevNorthWest = prevDecodedImage.at<Vec3b>(j - 1,i - 1);
              prevNorth = prevDecodedImage.at<Vec3b>(j-1,i);
              prevWestWest = prevDecodedImage.at<Vec3b>(j,i - 2);
```

```
          prevWest = prevDecodedImage.at<Vec3b>(j, i - 1);
          prevPixel = prevDecodedImage.at<Vec3b>(j, i);
          if (x == w-1){
            prevNorthNorthEast = 0;
            prevNorthEast = 0;
          }
          else{
            prevNorthNorthEast = prevDecodedImage.at<Vec3b>(j - 2, i + 1);
            prevNorthEast = prevDecodedImage.at<Vec3b>(j - 1, i + 1);
          }
          if(abs(prevWest.val[z] - West.val[z]) <= threshold && abs(prevNorthWest.val[z] - NorthWest.val[z]) <= threshold && abs(prevNorth.val[z] - North.val[z]) <= threshold && abs(prevNorthEast.val[z] - NorthEast.val[z]) <= threshold){
            decodedPrediction = prevPixel.val[z];
            flag = 1;
            goto finish4;
          }
        }
      }
    }
    else if(Ydiff >= 2 && Xdiff >= 2 && Yadd < h && Xadd >= w){
      for(j = Ydiff; j < Yadd; j++){
        for(i = Xdiff; i < w; i++){
          prevNorthNorth = prevDecodedImage.at<Vec3b>(j - 2, i);
          prevNorthWest = prevDecodedImage.at<Vec3b>(j - 1, i - 1);
          prevNorth = prevDecodedImage.at<Vec3b>(j-1, i);
          prevWestWest = prevDecodedImage.at<Vec3b>(j, i - 2);
          prevWest = prevDecodedImage.at<Vec3b>(j, i - 1);
          prevPixel = prevDecodedImage.at<Vec3b>(j, i);
          if (x == w-1){
            prevNorthNorthEast = 0;
            prevNorthEast = 0;
          }
          else{
```

```
                prevNorthNorthEast = prevDecodedImage.at<Vec3b>(j - 2,i + 1);
                prevNorthEast = prevDecodedImage.at<Vec3b>(j - 1,i + 1);
            }
            if(abs(prevWest.val[z] - West.val[z]) <= threshold && abs(prevNorthWest.val[z] - NorthWest.val[z]) <= threshold && abs(prevNorth.val[z] - North.val[z]) <= threshold && abs(prevNorthEast.val[z] - NorthEast.val[z]) <= threshold){
                decodedPrediction = prevPixel.val[z];
                flag = 1;
                goto finish4;
            }
        }
    }
}
else if(Ydiff >= 2 && Xdiff >= 2 && Yadd >= h && Xadd < w){
    for(j = Ydiff; j < h; j++){
        for(i = Xdiff; i < Xadd; i++){
            prevNorthNorth = prevDecodedImage.at<Vec3b>(j - 2,i);
            prevNorthWest = prevDecodedImage.at<Vec3b>(j - 1,i - 1);
            prevNorth = prevDecodedImage.at<Vec3b>(j-1,i);
            prevWestWest = prevDecodedImage.at<Vec3b>(j,i - 2);
            prevWest = prevDecodedImage.at<Vec3b>(j,i - 1);
            prevPixel = prevDecodedImage.at<Vec3b>(j,i);
            if (x == w-1){
                prevNorthNorthEast = 0;
                prevNorthEast = 0;
            }
            else{
                prevNorthNorthEast = prevDecodedImage.at<Vec3b>(j - 2,i + 1);
                prevNorthEast = prevDecodedImage.at<Vec3b>(j - 1,i + 1);
            }
            if(abs(prevWest.val[z] - West.val[z]) <= threshold && abs(prevNorthWest.val[z] - NorthWest.val[z]) <= threshold && abs(prevNorth.val[z] - North.val[z]) <= threshold
```

```
                    && abs(prevNorthEast.val[z] - NorthEast.val[z]) <= threshold){
                    decodedPrediction = prevPixel.val[z];
                    flag = 1;
                    goto finish4;
                }
            }
        }
    }
    else{ //if(Ydiff >= 2; Xdiff >= 2; Yadd >= h; Xadd >=w)
        for(j = Ydiff; j < h; j++){
            for(i = Xdiff; i < w; i++){
                prevNorthNorth = prevDecodedImage.at<Vec3b>(j - 2,i);
                prevNorthWest = prevDecodedImage.at<Vec3b>(j - 1,i - 1);
                prevNorth = prevDecodedImage.at<Vec3b>(j-1,i);
                prevWestWest = prevDecodedImage.at<Vec3b>(j,i - 2);
                prevWest = prevDecodedImage.at<Vec3b>(j,i - 1);
                prevPixel = prevDecodedImage.at<Vec3b>(j,i);
                if (x == w-1){
                    prevNorthNorthEast = 0;
                    prevNorthEast = 0;
                }
                else{
                    prevNorthNorthEast = prevDecodedImage.at<Vec3b>(j - 2,i + 1);
                    prevNorthEast = prevDecodedImage.at<Vec3b>(j - 1,i + 1);
                }
                if(abs(prevWest.val[z] - West.val[z]) <= threshold && abs(prevNorthWest.val[z] - NorthWest.val[z]) <= threshold
                    && abs(prevNorth.val[z] - North.val[z]) <= threshold
                    && abs(prevNorthEast.val[z] - NorthEast.val[z]) <= threshold){
                    decodedPrediction = prevPixel.val[z];
                    flag = 1;
                    goto finish4;
                }
            }
```

```
                    }
                }
                threshold ++;
            }
        }
        finish4:

        e = (signed int16_t)undoRemapping(decodedImagePixel.val[z], decodedPrediction);
        pixelVal = decodedPrediction + e;
        decodedImage.at<Vec3b>(y,x)[z] = uint8_t(pixelVal);

        flag = 0;
        threshold = 0;
      }
    }
  }
  if(imageCount == 1){
    decodedImage1 = decodedImage;
  }
  else if(imageCount == 2){
    decodedImage2 = decodedImage;
  }
  else if(imageCount == 3){
    decodedImage3 = decodedImage;
  }
  else if(imageCount == 4){
    decodedImage4 = decodedImage;
  }
  else if(imageCount == 5){
    decodedImage5 = decodedImage;
  }
}

Vec3b orig;
Vec3b decoded;
Vec3b test;
```

```
for(int imageCount = 0; imageCount < 6; imageCount ++){
  for(y = 0; y < h; y++){
    for(x = 0; x < w; x++){
      if(imageCount == 0){
        orig = image0.at<Vec3b>(y,x);
        decoded = decodedImage0.at<Vec3b>(y,x);
      }
      else if(imageCount == 1){
        orig = image1.at<Vec3b>(y,x);
        decoded = decodedImage1.at<Vec3b>(y,x);
      }
      else if(imageCount == 2){
        orig = image2.at<Vec3b>(y,x);
        decoded = decodedImage2.at<Vec3b>(y,x);
      }
      else if(imageCount == 3){
        orig = image3.at<Vec3b>(y,x);
        decoded = decodedImage3.at<Vec3b>(y,x);
      }
      else if(imageCount == 4){
        orig = image4.at<Vec3b>(y,x);
        decoded = decodedImage4.at<Vec3b>(y,x);
      }
      else if(imageCount == 5){
        orig = image5.at<Vec3b>(y,x);
        decoded = decodedImage5.at<Vec3b>(y,x);
      }
      for(z = 0; z < c; z++){
        if(orig.val[z] != decoded.val[z]){
          printf("image \%d\n", imageCount);
          printf("\%d != \%d. You messed up at (\%d, \%d, \%d) (\%d, \%d, \%
              d) -> (\%d, \%d, \%d) \n", decoded.val[z], orig.val[z], x, y,
              z, decoded.val[0], decoded.val[1], decoded.val[2], orig.val
              [0], orig.val[1], orig.val[2]);
          return -1;
        }
      }
```

```
            }
        }
    }

    printf("CONGRADULATIONS! YOU DID IT! \n");
    return 0;
}
```

Listing B.1: Match - main.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <fstream>
#include <string>
#include <math.h>
#include <iterator>
#include <vector>

#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>

#include <math.h>

using namespace cv;
using namespace std;

uint32_t symbol_count_e[257] = {0};//[9] = {0};//[257] = {0};
uint32_t cum_count_e[257] = {0};//[9] = {0};//[257] = {0};
extern uint32_t maxCount;

uint16_t pix_to_index_e[256] = {0};//[8] = {0};//[256] = {0};
uint16_t index_to_pix_e[257] = {0};//[9] = {0};//[257] = {0};

uint16_t l_e = 0, u_e = 65535;
uint32_t scale3_e = 0;

```

```
unsigned long outputBitLocation_e = 0;

extern uint32_t fileByteRW;
uint64_t sizeOfArray_e = fileByteRW*8;
uint64_t index_e = 0;
uint8_t * output_array_e;
FILE * encodedFile = NULL;

void setBitArray(uint64_t bitLocation){
    int byteLocation = bitLocation >> 3;
    *(output_array_e + byteLocation) |= (0x01 << (7 - (bitLocation & 0x07)));
    index_e ++;
    outputBitLocation_e ++;

    if(index_e == sizeOfArray_e){
        //Write the first array to the file
        fwrite(output_array_e, sizeof(uint8_t), fileByteRW, encodedFile);
        index_e = 0;
    }
}

void clearBitArray(uint64_t bitLocation){
    int byteLocation = bitLocation >> 3;
    *(output_array_e + byteLocation) &= ~ (0x01 << (7 - (bitLocation & 0x07)))
        ;
    index_e ++;
    outputBitLocation_e ++;
    if(index_e == sizeOfArray_e){
      //Write the first array to the file
      fwrite(output_array_e, sizeof(uint8_t), fileByteRW, encodedFile);
      index_e = 0;
    }
}

void updateLimits_e(uint16_t symbol){
  uint16_t low = l_e;
  l_e = l_e + (((u_e - l_e + 1) * cum_count_e[symbol]) / cum_count_e[0]);
```

```
  u_e = low + (((u_e - low + 1) * cum_count_e[symbol - 1]) / cum_count_e[0]) - 1;
}

void checkLimits(){
  uint8_t lBit_e, uBit_e;
  lBit_e = ((uint16_t) l_e >> 15) & 0x01;
  uBit_e = ((uint16_t) u_e >> 15) & 0x01;
  while(lBit_e == uBit_e){
    l_e = l_e << 1;
    u_e = (u_e << 1) | 0x01;

    if(lBit_e == 0){
      clearBitArray(index_e);
      if(scale3_e > 0){
        for(uint32_t k = 0; k < scale3_e; k++){
          setBitArray(index_e);
        }
        scale3_e = 0;
      }
    }

    else if (lBit_e == 1){
      setBitArray(index_e);
      if(scale3_e > 0){
        for(uint32_t k = 0; k < scale3_e; k++){
          clearBitArray(index_e);
        }
        scale3_e = 0;
      }
    }

    lBit_e = ((uint16_t) l_e >> 15) & 0x01;
    uBit_e = ((uint16_t) u_e >> 15) & 0x01;
  }
}
```

```
void E3Check(){
  uint16_t l_sMSB_e, u_sMSB_e;
  l_sMSB_e = l_e & 0x4000;
  u_sMSB_e = u_e & 0x4000;
  while((l_sMSB_e == 0x4000) && (u_sMSB_e == 0)){
    scale3_e ++;
    l_e = l_e << 1;
    u_e  = (u_e << 1) | 0x01;
    u_e |= 0x8000;
    l_e &= 0x7FFF;
    l_sMSB_e = l_e & 0x4000;
    u_sMSB_e = u_e & 0x4000;
  }
}

void updateCounts_e(uint16_t symbol){
  int i; //new index for symbol

  //See if frequency counts are at the maximum
  if(cum_count_e[0] == maxCount){
    //halve all of the counts (keeping them non-zero)
    int cum;
    cum = 0;
    for(i = 256; i >= 0; i --){
      symbol_count_e[i] = (symbol_count_e[i] + 1) / 2;
      cum_count_e[i] = cum;
      cum += symbol_count_e[i];
    }
    symbol_count_e[0] = 0;
  }

  //find the symbols new index
  i = symbol;
  while(i > 0){
    if(symbol_count_e[i] != symbol_count_e[i-1]){
      break;
    }
```

```
    i--;
  }

  //update the translation tables if the symbol has moved
  if(i < symbol){
    int pix_i, pix_symbol;
    pix_i = index_to_pix_e[i];
    pix_symbol = index_to_pix_e[symbol];
    index_to_pix_e[i] = pix_symbol;
    index_to_pix_e[symbol] = pix_i;
    pix_to_index_e[pix_i] = symbol;
    pix_to_index_e[pix_symbol] = i;
  }
  else{
    i = symbol;
  }

  //increment the count for the symbol and update the cumulative counts
  symbol_count_e[i] += 1;
  //i = symbol;
  while(i > 0){
    i --;
    cum_count_e[i] += 1;
  }
}

void encode(uint16_t byte){
  updateLimits_e(byte);
  checkLimits();
  E3Check();
  updateCounts_e(byte);

  //Double check the limits to ensure it's working
  if(u_e <= 0){
    cout << "Error. The upper limit is malfunctioning" << endl;
  }
  if(l_e >= 65535){
```

```
      cout << "Error. The lower limit is malfunctioning" << endl;
    }
}

void sendLowerLimit(void){
  uint16_t l_binary;
  for(int j = 0; j < 16; j++){
    l_binary =((uint16_t) l_e >> (15-j)) & 0x01;

    if(l_binary == 1){
      setBitArray(index_e);
      if(scale3_e > 0){
          for(uint32_t k = 0; k < scale3_e; k++){
            clearBitArray(index_e);
          }
          scale3_e = 0;
      }
    }
    else{
      clearBitArray(index_e);
      if(scale3_e > 0){
          for(uint32_t k = 0; k < scale3_e; k++){
            setBitArray(index_e);
          }
          scale3_e = 0;
      }
    }
  }
  if(index_e >0){
    fwrite(output_array_e, 1, ((index_e + 7) >> 3), encodedFile);
  }
}
```

Listing B.2: Adaptive Arithmetic Encoder

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
```

```
#include <fstream>
#include <string>
#include <math.h>

#include <iterator>
#include <vector>

#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>

using namespace cv;
using namespace std;

//#include "encoder.cpp"

uint32_t symbol_count_d[257] = {0};
uint32_t cum_count_d[257] = {0};
extern uint32_t maxCount;

uint16_t pix_to_index_d[256] = {0};
uint16_t index_to_pix_d[257] = {0};

uint16_t l_d = 0, u_d = 65535;
uint16_t tag;
uint16_t tBit;
uint64_t t;
uint16_t byte_d;

extern uint32_t fileByteRW;
uint8_t toDecode[32768] = {0};//[3981068] = {0};
uint64_t index_d = 16;
uint32_t sizeOfArray_d = fileByteRW*8;
uint8_t flag_d = 0;
uint8_t flag_rows = 0, flag_cols = 0;
uint32_t x_d = 0,y_d = 0;

extern uint64_t fileSize;
```

```
extern uint64_t fileLocation;
extern uint64_t maxFileLocation;
extern uint32_t h, w, x, y;
extern uint8_t z, c;
extern Mat decodedImage, decodedImage0, decodedImage1, decodedImage2,
    decodedImage3, decodedImage4, decodedImage5;

void updateLimits_d(uint16_t symbol){
  uint16_t low = l_d;
  l_d = low + (((u_d - low + 1) * cum_count_d[symbol]) / cum_count_d[0]);
  u_d = low + (((u_d - low + 1) * cum_count_d[symbol - 1]) / cum_count_d[0]) -
        1;
}

void limitCheck(){
  uint16_t lBit_d, uBit_d, newBit;
  lBit_d = ((uint16_t) l_d >> 15) & 0x01;
  uBit_d = ((uint16_t) u_d >> 15) & 0x01;
  while(lBit_d == uBit_d){
      l_d = l_d << 1;
      u_d = (u_d << 1) | 0x01;
      byte_d = index_d >> 3;
      newBit = (toDecode[byte_d] >> (7 - (index_d & 0x07))) & 0x01;
      tag = (tag << 1) | newBit;
      index_d++;
      if(index_d == sizeOfArray_d){
        index_d = 0;
        if(flag_d == 0){
          //Read in next bit of file
          fread(toDecode,1,fileByteRW,encodedFile);
          fseek(encodedFile,fileByteRW*fileLocation,SEEK_SET);
          fileLocation ++;
          if(fileLocation > maxFileLocation){
            flag_d = 1;
          }
        }
        else{
```

```
            //Read in the last bit of the file
            fread(toDecode,1,(fileSize - (fileByteRW*maxFileLocation)),
                encodedFile);
          }
        }
        lBit_d = ((uint16_t) l_d >> 15) & 0x01;
        uBit_d = ((uint16_t) u_d >> 15) & 0x01;
  }
}

void E3Check_d(){
  uint16_t l_sMSB_d, u_sMSB_d;
  uint8_t newBit;
  l_sMSB_d = l_d & 0x4000;
  u_sMSB_d = u_d & 0x4000;
  while((l_sMSB_d == 0x4000) && (u_sMSB_d == 0)){
    l_d = l_d << 1;
    u_d = (u_d << 1) | 0x01;
    u_d |= 0x8000;
    l_d &= 0x7FFF;
    l_sMSB_d = l_d & 0x4000;
    u_sMSB_d = u_d & 0x4000;
    //Dealing with the tag
    tBit = tag & 0x4000;
    if(tBit == 0){
      byte_d = index_d >> 3;
      newBit = (toDecode[byte_d] >> (7 - (index_d & 0x07))) & 0x01;
      tag = (tag << 1) | newBit;
      tag |= 0x8000;
    }
    else{ //tbit = 1
      byte_d = index_d >> 3;
      newBit = (toDecode[byte_d] >> (7 - (index_d & 0x07))) & 0x01;
      tag = (tag << 1) | newBit;
      tag &= 0x7FFF;
    }

```

```
    index_d++;
    if(index_d == sizeOfArray_d){
      index_d = 0;
      if(flag_d == 0){
        //Read in next bit of file
        fread(toDecode,1,fileByteRW,encodedFile);
        fseek(encodedFile,fileByteRW*fileLocation,SEEK_SET);
        fileLocation ++;
        if(fileLocation > maxFileLocation){
          flag_d = 1;
        }
      }
      else{
        //Read in last bit of the file
        fread(toDecode,1,(fileSize - (fileByteRW*maxFileLocation)),encodedFile
            );
      }
    }
  }
}

void updateCounts_d(uint16_t symbol){
  int i; //new index for symbol

  //See if frequency counts are at the maximum
  if(cum_count_d[0] == maxCount){
    //halve all of the counts (keeping them non-zero)
    int cum;
    cum = 0;
    for(i = 256; i >= 0; i--){
      symbol_count_d[i] = (symbol_count_d[i] + 1) / 2;
      cum_count_d[i] = cum;
      cum += symbol_count_d[i];
    }
    symbol_count_d[0] = 0;
  }

```

```
    //find the symbols new index
    i = symbol;
    while(i > 0){
        if(symbol_count_d[i] != symbol_count_d[i-1]){
            break;
        }
        i--;
    }

    //update the translation tables if the symbol has moved
    if(i < symbol){
        int pix_i, pix_symbol;
        pix_i = index_to_pix_d[i];
        pix_symbol = index_to_pix_d[symbol];
        index_to_pix_d[i] = pix_symbol;
        index_to_pix_d[symbol] = pix_i;
        pix_to_index_d[pix_i] = symbol;
        pix_to_index_d[pix_symbol] = i;
    }
    else{
        i = symbol;
    }

    //increment the count for the symbol and update the cumulative counts
    symbol_count_d[i] += 1;
    while(i > 0){
        i --;
        cum_count_d[i] += 1;
    }
}


void decode(int height, int width, int channels){
    Vec3b NorthNorth, NorthNorthEast, NorthWest, North, NorthEast, WestWest,
        West, decodedImagePixel;
    int16_t e;
    uint16_t decodedPrediction;
```

```
h = height;
w = width;
c = channels;
z = 0;
y_d = 0;
x_d = 0;

uint8_t pix;
uint16_t symbol;
uint8_t imageCount = 0;

while(imageCount < 6){
  t = (((tag - l_d + 1) * cum_count_d[0] - 1)/(u_d - l_d + 1));
  //find symbol:
  for(symbol = 1; cum_count_d[symbol] > t; symbol++);
  //translate to a character
  pix = index_to_pix_d[symbol];

  //plug the decoded symbol into the matrix:
  if(flag_rows == 0){
    if(imageCount == 0){
      decodedImage0.at<Vec3b>(y_d,x_d)[z] = pix;
    }
    else if(imageCount == 1){
      decodedImage1.at<Vec3b>(y_d,x_d)[z] = pix;
    }
    else if(imageCount == 2){
      decodedImage2.at<Vec3b>(y_d,x_d)[z] = pix;
    }
    else if(imageCount == 3){
      decodedImage3.at<Vec3b>(y_d,x_d)[z] = pix;
    }
    else if(imageCount == 4){
      decodedImage4.at<Vec3b>(y_d,x_d)[z] = pix;
    }
    else if(imageCount == 5){
      decodedImage5.at<Vec3b>(y_d,x_d)[z] = pix;
```

```
    }
    z ++;
    if(z == 3){
      z = 0;
      x_d ++;
      if(x_d == 2){
        x_d = 0;
        y_d ++;
        if(y_d == h){
          y_d = 0;
          x_d = 2;
          flag_rows = 1;
        }
      }
    }
  }
  else if(flag_rows == 1 && flag_cols == 0){
    if(imageCount == 0){
      decodedImage0.at<Vec3b>(y_d,x_d)[z] = pix;
    }
    else if(imageCount == 1){
      decodedImage1.at<Vec3b>(y_d,x_d)[z] = pix;
    }
    else if(imageCount == 2){
      decodedImage2.at<Vec3b>(y_d,x_d)[z] = pix;
    }
    else if(imageCount == 3){
      decodedImage3.at<Vec3b>(y_d,x_d)[z] = pix;
    }
    else if(imageCount == 4){
      decodedImage4.at<Vec3b>(y_d,x_d)[z] = pix;
    }
    else if(imageCount == 5){
      decodedImage5.at<Vec3b>(y_d,x_d)[z] = pix;
    }
    z++;
    if(z == 3){
```

```
          z = 0;
          y_d ++;
          if(y_d == 2){
            y_d = 0;
            x_d ++;
            if(x_d == w){
              y_d = 2;
              x_d = 2;
              flag_cols = 1;
            }
          }
        }
      }
      else{
        if(imageCount == 0){
          decodedImage0.at<Vec3b>(y_d,x_d)[z] = pix;
        }
        else if(imageCount == 1){
          decodedImage1.at<Vec3b>(y_d,x_d)[z] = pix;
        }
        else if(imageCount == 2){
          decodedImage2.at<Vec3b>(y_d,x_d)[z] = pix;
        }
        else if(imageCount == 3){
          decodedImage3.at<Vec3b>(y_d,x_d)[z] = pix;
        }
        else if(imageCount == 4){
          decodedImage4.at<Vec3b>(y_d,x_d)[z] = pix;
        }
        else if(imageCount == 5){
          decodedImage5.at<Vec3b>(y_d,x_d)[z] = pix;
        }
        z ++;
        if(z == 3){
          z = 0;
          x_d ++;
          if(x_d == w){
```

```
            x_d = 2;
            y_d ++;
            if(y_d == h){
              flag_cols = 0;
              flag_rows = 0;
              x_d = 0;
              y_d = 0;
              z = 0;
              imageCount ++;
            }
          }
        }
      }

      //Update the limits
      updateLimits_d(symbol);
      //Check the bits
      limitCheck();
      //Check the E3 condition
      E3Check_d();
      //Update the Count
      updateCounts_d(symbol);
    }
}
```

Listing B.3: Adaptive Arithmetic Decoder

```
#include <stdio.h>
#include <stdlib.h>

#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>

#include <math.h>

using namespace cv;
using namespace std;
```

```
//Find the initial prediction
uint16_t initially_predict(uint8_t NN, uint8_t NNE, uint8_t NW, uint8_t N,
    uint8_t NE, uint8_t WW, uint8_t W){
  int hdifference, vdifference;
  uint8_t x_hat;
  int predictedPixel;

  //Finding the horizontal difference = |W - WW| + |N - NW| + |NE - N|
  hdifference = abs(W - WW) + abs(N - NW) + abs(NE - N);

  //Find the vertical difference = |W - NW| + |N - NN| + |NE - NNE|
  vdifference = abs(W - NW) + abs(N - NN) + abs(NE - NNE);

  //Now to determine the initial prediction of the Pixel
  if(hdifference - vdifference > 80){
    predictedPixel = N;
  }
  else if(vdifference - hdifference > 80){
    predictedPixel = W;
  }
  else{
    predictedPixel = ((N + W) / 2) + ((NE - NW) / 4);
    if(hdifference - vdifference > 32){
      predictedPixel = (predictedPixel + N) / 2;
    }
    else if(vdifference - hdifference > 32){
      predictedPixel = (predictedPixel + W) / 2;
    }
    else if(hdifference - vdifference > 8 && hdifference - vdifference < 32){
      predictedPixel = ((3 * predictedPixel) + N) / 4;
    }
    else if(vdifference - hdifference > 8 && vdifference - hdifference < 32){
      predictedPixel = ((3 * predictedPixel) + W) / 4;
    }
  }
  return predictedPixel;
```

```
}

//Remapping so that all the error values are positive and within the range
    0-255
uint8_t Remap(int16_t e, int16_t prediction){
  uint8_t r;

  if(prediction <= 127){
    if(abs(e) <= prediction){
      if(e <= 0){
        r = 2 * (abs(e));
      }
      else{
        r = (2 * e) - 1;
      }
    }
    else{
      r = e + prediction;
    }
  }
  else{
    if(abs(e) <= (256 - 1 - prediction)){
      if(e <= 0){
        r = 2 * (abs(e));
      }
      else{
        r = (2 * e) - 1;
      }
    }
    else{
      r = abs(e) + (256 - 1 - prediction);
    }
  }
  return r;
}

int16_t undoRemapping(uint8_t remapped, int16_t prediction){
```

```
  int16_t originalError;

  if(prediction <= 127){
    if(remapped <= 2 * prediction){
      if(remapped % 2 == 0){
        originalError = - (remapped / 2);
      }
      else{
        originalError = (remapped + 1) / 2;
      }
    }
    else{
      originalError = remapped - prediction;
    }
  }
  else{
    if(remapped <= 2*(256 - 1 - prediction)){
      if(remapped % 2 == 0){
        originalError = - remapped / 2;
      }
      else{
        originalError = (remapped + 1) / 2;
      }
    }
    else{
      originalError = - remapped + (256 - 1 - prediction);
    }
  }
  return (signed int16_t)originalError;
}
```

Listing B.4: CALIC

```
def compare_images(imageA, imageB):
        # compute the mean squared error and structural similarity
        # index for the images
    m = mse(imageA, imageB)
    s = ssim(imageA, imageB)
```

```
    return m, s

image0 = cv2.imread("/path/to/file", 0))
image1 = cv2.imread("/path/to/file", 0))
image2 = cv2.imread("/path/to/file", 0))
image3 = cv2.imread("/path/to/file", 0))
image4 = cv2.imread("/path/to/file", 0))
image5 = cv2.imread("/path/to/file", 0))

MeanSquaredError1, StructuralSimilarity1 = compare_images(image0, image1)
MeanSquaredError2, StructuralSimilarity2 = compare_images(image1, image2)
MeanSquaredError3, StructuralSimilarity3 = compare_images(image2, image3)
MeanSquaredError4, StructuralSimilarity4 = compare_images(image3, image4)
MeanSquaredError5, StructuralSimilarity5 = compare_images(image4, image5)

print(StructuralSimilarity1, StructuralSimilarity2, StructuralSimilarity3,
    StructuralSimilarity4, StructuralSimilarity5)
```

Listing B.5: SSIM (Python)

```
def edgeDetection(img, sigma, h, w):
    #We need to find the threshold, in which the threshold = 0.1(Cmax - Cmin)
        + Cmin
    #start by finding Cmax and Cmin which denote the maximum and minimum
        values of the norm of the gradient output
    #use sobel filters to calculate gradient
    dx = [[-1/8, 0, 1/8], [-2/8, 0, 2/8], [-1/8, 0, 1/8]]
    dy = [[1/8, 2/8, 1/8], [0, 0, 0], [-1/8, -2/8, -1/8]]

    gx = signal.convolve2d(img,dx)
    gy = signal.convolve2d(img,dy)

    gx_norm = ((gx - gx.min()) / (gx.max() - gx.min()))
    gy_norm = ((gy - gy.min()) / (gy.max() - gy.min()))

    C = np.hypot(gx,gy)
    C = (C / C.max()) #needs to be between 0 and 1
    Cmax = C.max()
```

```
    Cmin = C.min()

    #now that we have Cmax and Cmin, we can calculate the threshold
    T = (0.1 * (Cmax - Cmin)) + Cmin

    #blur the image with Gaussian
    blur = cv2.GaussianBlur(C, (5,5), sigma)

    E = np.zeros((h,w),dtype=np.float)

    for i in range(0, h):
        for j in range(0, w):
            if(C[i, j] > T):
                E[i,j] = 1
            else:
                E[i,j] = 0
    return E

def edgeStabilityMap(E1, E2, E3, E4, E5, h, w):
    Q = np.zeros((h,w),dtype=np.float)
    count = 0

    for i in range(0,h):
        for j in range(0,w):
            if E1[i,j] == 1 and E2[i,j] == 1 and E3[i,j] == 1 and E4[i,j] == 1
                and E5[i,j] == 1:
                Q[i,j] = 1
                count = count + 1
            else:
                Q[i,j] = 0
    if count == 0:
        count = 1
    return Q, count

image0 = cv2.imread("/path/to/file", 0))
image1 = cv2.imread("/path/to/file", 0))
image2 = cv2.imread("/path/to/file", 0))
```

```
image3 = cv2.imread("/path/to/file", 0))
image4 = cv2.imread("/path/to/file", 0))
image5 = cv2.imread("/path/to/file", 0))

#Find size of the image
dimensions = image1.shape
height = image0.shape[0]
width = image0.shape[1]

E1 = edgeDetection(image0, 1.19, height, width)
E2 = edgeDetection(image0, 1.44, height, width)
E3 = edgeDetection(image0, 1.68, height, width)
E4 = edgeDetection(image0, 2.0, height, width)
E5 = edgeDetection(image0, 2.38, height, width)
Q0, count0 = edgeStabilityMap(E1, E2, E3, E4, E5, height, width)

E1 = edgeDetection(image1, 1.19, height, width)
E2 = edgeDetection(image1, 1.44, height, width)
E3 = edgeDetection(image1, 1.68, height, width)
E4 = edgeDetection(image1, 2.0, height, width)
E5 = edgeDetection(image1, 2.38, height, width)
Q1, count1 = edgeStabilityMap(E1, E2, E3, E4, E5, height, width)

E1 = edgeDetection(image2, 1.19, height, width)
E2 = edgeDetection(image2, 1.44, height, width)
E3 = edgeDetection(image2, 1.68, height, width)
E4 = edgeDetection(image2, 2.0, height, width)
E5 = edgeDetection(image2, 2.38, height, width)
Q2, count2 = edgeStabilityMap(E1, E2, E3, E4, E5, height, width)

E1 = edgeDetection(image3, 1.19, height, width)
E2 = edgeDetection(image3, 1.44, height, width)
E3 = edgeDetection(image3, 1.68, height, width)
E4 = edgeDetection(image3, 2.0, height, width)
E5 = edgeDetection(image3, 2.38, height, width)
Q3, count3 = edgeStabilityMap(E1, E2, E3, E4, E5, height, width)

```

```
E1 = edgeDetection(image4, 1.19, height, width)
E2 = edgeDetection(image4, 1.44, height, width)
E3 = edgeDetection(image4, 1.68, height, width)
E4 = edgeDetection(image4, 2.0, height, width)
E5 = edgeDetection(image4, 2.38, height, width)
Q4, count4 = edgeStabilityMap(E1, E2, E3, E4, E5, height, width)

E1 = edgeDetection(image5, 1.19, height, width)
E2 = edgeDetection(image5, 1.44, height, width)
E3 = edgeDetection(image5, 1.68, height, width)
E4 = edgeDetection(image5, 2.0, height, width)
E5 = edgeDetection(image5, 2.38, height, width)
Q5, count5 = edgeStabilityMap(E1, E2, E3, E4, E5, height, width)

#Calculate the edge stability mean squared error between Frames
sum0 = 0
for i in range(0, height):
    for j in range(0, width):
        if(Q0[i,j] == 1):
            sum0 = sum0 + (Q0[i,j] - Q1[i,j])**2
ESMSE1 = (sum0 / count0)

sum1 = 0
for i in range(0, height):
    for j in range(0, width):
        if(Q1[i,j] == 1):
            sum1 = sum1 + (Q1[i,j] - Q2[i,j])**2
ESMSE2 = (sum1 / count1)

sum2 = 0
for i in range(0, height):
    for j in range(0, width):
        if(Q2[i,j] == 1):
            sum2 = sum2 + (Q2[i,j] - Q3[i,j])**2
ESMSE3 = (sum2 / count2)

sum3 = 0
```

```
for i in range(0, height):
    for j in range(0, width):
        if(Q3[i,j] == 1):
            sum3 = sum3 + (Q3[i,j] - Q4[i,j])**2
ESMSE4 = (sum3 / count3)

sum4 = 0
for i in range(0, height):
    for j in range(0, width):
        if(Q4[i,j] == 1):
            sum4 = sum4 + (Q4[i,j] - Q5[i,j])**2
ESMSE5 = (sum4 / count4)

print(ESMSE1, ESMSE2, ESMSE3, ESMSE4, ESMSE5)
```

Listing B.6: Edge Quality (Python)