FastScale - Video Super Resolution

James S. Garcia, Joshua Lau, Kunal Barde
Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—Recent trends in display hardware are trending towards larger screens and higher display resolutions. Legacy videos are not made for these formats and thus must be upscaled. While this can be done in many ways, most tend to perform poorly on speed, image quality, or form-factor. We set out to create an embedded real-time, super resolution device using a Xilinx Ultra96v2 MPSoC and variations on existing algorithms, with our final design being based off of the paper FSRCNN. Our design aimed to be convenient and intuitive for a general, non-technical audience, as well as offering advantages to video quality compared to traditional digital signal processing approaches, without sacrificing on speed.**

***Index Terms*—Super Resolution, Upscaling, Standard Definition, High Definition, Throughput, Latency, FPGA, Hardware Acceleration, SSIM, DSP, Machine Learning**

1 INTRODUCTION

Modern screens are made to display high definition videos, presenting a problem for legacy videos recorded or stored in standard definition. Enlarging these videos has often been challenging, as it is a classic signal reconstruction problem. Contemporary super resolution algorithms tend to rely upon either DSP methods or ML. While DSP methods can be fast and may qualitatively[1] appear to improve, it remains that to achieve a quantitatively higher degree of image reconstruction, machine learning methods must be deployed. As a rule, these are slower, but can be run on larger, more power-consumptive processors to run at speed. The disadvantage with this is the cost and reduced-usability of these more powerful devices for everyday consumers, with additional detriment if the device is a 'unitasker' product i.e., one which only serves one purpose.

We set out to create an embedded device which caters to everyday consumers, who may not be especially tech savvy. Our product would leverage hardware acceleration in order to balance speed, quantitative image reconstruction, and form-factor. More specifically, we set out to develop a super resolution algorithm which we would then implement on an MPSoC so that it could be hardware accelerated by an FPGA. Our product was designed to outperform standard fast DSP upscaling methods for transforming standard definition video frames to high definition. Additionally, our product was designed to play video with negligible latent delay and native video frame-rate. We also designed our product to have a small footprint.

2 DESIGN REQUIREMENTS

We strived to meet requirements in two main areas: quality and timing.

In terms of quality, to quantitatively determine what constitutes better reconstruction, we decided to use the *structural similarity index measure* or SSIM. This metric measures similarity in images, or in our case video frames, but unlike metrics like mean-squared error or peak signal to noise ratio, accounts for human perception in that it accounts for: the fact that humans perceive more error if the error is concentrated in interdependent pixels, that nearby pixels are usually interdependent, and that bright or highly textured parts of an image decrease a human's ability to detect errors, i.e. contrast, structure, and luminance. This is the main reason why we chose SSIM to measure the image quality of our super resolution algorithm. Using SSIM, we wanted to show that we could do better than the most common *digital signal processing*, or DSP, method - bicubic interpolation.

We aimed to convert from standard definition to high definition. High definition is usually defined as a 1080 by 1920 display, and since we target digital displays, we chose 1080p as our target resolution. Standard definition for digital screens is typically defined as any of the following: 144p, 240p, 360p, or 480p. We chose 240p as it was a very common display resolution around the turn of the century, and has still persisted until today as a resolution for low bandwidth streaming. We further specified that we would use 240p-widescreen as the non-widescreen version of 240p has a 3:4 aspect ratio, while 1080p, which is widescreen by default, has an aspect ratio of 9:16. Using 240p-widescreen as our input resolution fixes the need for any stretching or black-bar padding, as it has the same[2] aspect ratio as our high definition output. From these choices, we constrained our necessary scaling factor by the relationship described in (1), thus finding a necessary scaling factor of 4.5.

$$\text{scaling factor} = \frac{\text{output resolution}}{\text{input resolution}} \tag{1}$$

In terms of timing, originally we intended our device to sit between a video stream and an output and hence constrained our latent delay to be the maximum amount of time permissible without causing audio-video desynchronisation: 60ms. Since we changed our intended application to draw from stored videos, we wanted our latent delay to

[1]These qualitative measurements tend to be anecdotal or from large scale user feedback surveys.
[2]Since display is pixel-quantised, this is technically a 9:15.975 aspect ratio.

still be negligible, allowing a near instantaneous experience for the user. Hence, we ended up aimed for a 250ms latent delay, around the minimum human reaction time.

In terms of throughput, we knew that the illusion of video is only valid down to 12FPS. Hence, our minimum throughput would have to be the inverse, or 83ms of delay. At the same time however, we knew that below 24-30FPS video footage becomes perceptibly choppy and less smooth. As video quality is at the core of our project, we chose a target throughput of 30FPS, as this would be the minimum throughput at which the video would be perceived as smooth.

3 ARCHITECTURE OVERVIEW

3.1 Software Overview

Our final software model is based off the FSRCNN-s algorithm which is a 5-staged convolutional neural network. As depicted in figure 1, we take 1080p videos from our dataset, downscale them through bicubic interpolation, and feed the frames into the CNN. We then take the upscaled output, compare that with the frames from our dataset using SSIM, and backpropagate in order to train our CNN. As depicted in Figure 3 we can see the first layer is a convolution with a 5 by 5 kernel responsible for extracting key features from the input image. Then, there is a shrinking step which reduces the number of feature maps to a much smaller number than output by the first feature extraction step. After this, we apply a mapping step where the maps are convolved with 3 by 3 kernels multiple times in order to extract relevant information through non-linear mapping. Then the maps are expanded to increase the amount of information in the system. Finally a deconvolutional layer[3] outputs the higher resolution frame whose scale is based on the stride of the deconvolution.

3.2 Hardware Overview

Our system leverages the Ultra96v2 development board to act as a media center device. At a high level, user interaction with our system is not dissimilar to interacting with some media centers on other single board computers, such as a raspberry pi. The difference in our system, and reason we used the Ultra96v2, is that the Ultra96v2 has both an ARM based CPU and a programmable logic fabric, allowing computationally-intensive parallelisable algorithms to be hardware accelerated. As we targeted a very computationally intense problem, super resolution, having this ability was key.

Our system provides an interface for the user to interact with our hardware accelerated algorithm without having to be too 'in-the-weeds' of the algorithm itself. While the interface is via the terminal, as opposed to a proper GUI, it still provides some abstraction away from writing and compiling software one ones own.

While at the beginning, we intended for our system to be compatible with any screen, we found that we could only ensure compatibility with DisplayPort enabled screens of certain resolutions.

Figure 2 details all the user has to interact with. Trapezoidal boxes are user IO, with the right-most column being items which users provide themselves. Everything else is an in-built part of the system, as we would need to ship the system with a pre-synthesized hardware kernel for super resolution, as well as the glue scripts which interact with sub-programs written for the system.

Further in figure 2, the rounded boxes refer to programs which we wrote for the Ultra96v2, the square boxes refer to hardware components and subcomponents of the Ultra96v2 board, and the hexagonal boxes are converters or interconnect.

4 DESIGN TRADE STUDIES

4.1 Upscaling Quality Metric

4.1.1 SSIM vs VMAF

Our training and evaluation metric was done based on SSIM. During early conception of the project, we evaluated several other metrics, such as PSNR, MSE, and an open source metric developed by Netflix - VMAF. Evaluating the use cases which each metric performed best in, we arrived at a decision between SSIM and VMAF. PSNR and MSE, both pixel-by-pixel comparisons, fail to adequately describe the quality of an image as we define it. To explain it better, multiple forms of degradation performed on an image can yield the same MSE and PSNR, whereas SSIM takes into account a combination of three separate factors - luminance, contrast, and structure. Consider the case where every pixel is 'off' by some small delta. If each pixel is 'off' in the same direction, PSNR and MSE could represent this as a large difference, whereas VMAF and SSIM would take into account more factors and return a lower error, more closely representing how the human eye would regard a washed out image to be less 'bad' than an image with 'static' noise.

Between SSIM and VMAF, the Netflix metric initially appeared more applicable to our project as it was a custom metric that was developed specifically to give a predictive measure of how people would react to differences in quality. However, upon testing an implementation of it, we realised that it ran very slow. The total time to evaluate VMAF on clips from our dataset of around 10 seconds was close to a minute on one of our CPUs locally. Moreover, running on a GPU would help with the transcoding process, but the calculation of VMAF wouldn't receive much additional benefit from using a GPU instead of a CPU. Thus, we disqualified VMAF as a usable metric for training.

[3]This is a poor term, but unfortunately is the industry standard. A more correct and descriptive term would be to call it a backwards or fractionally strided convolutional layer

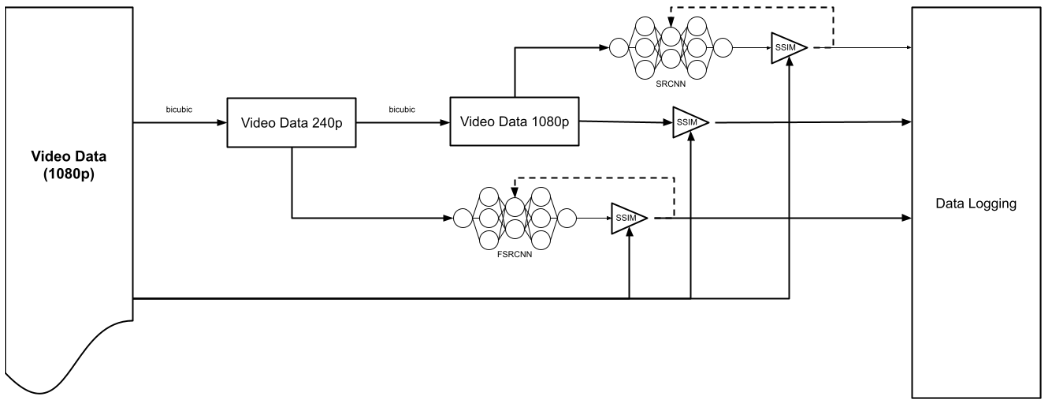


Figure 1: High Level Overview of the Software Model

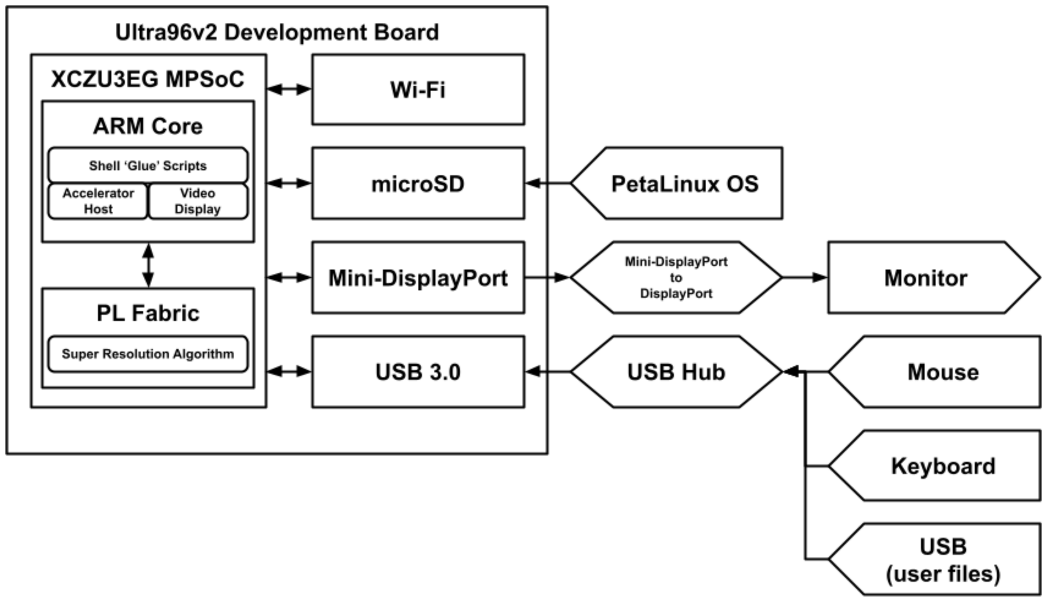


Figure 2: High Level Overview of the System on the Ultra96v2

We finally settled on SSIM, as, similar to VMAF, it can be tuned to account better for human vision but, from our tests, has a much lower computational overhead compared to VMAF. In our choice of SSIM, we know that unlike VMAF, it is not a metric which operates directly on videos or sequences of images but rather single images alone. However, the paper "Video Quality Assessment Based on Structural Distortion Measurement" by Wang, Lu, and Bovik[5] details how to modify SSIM to analyze video data. We will be using their same formulation for training and validating our system, which we will go into detail in Section 6.

4.2 Timing and Latency

4.2.1 SRCNN vs FSRCNN

The main difference, in terms of our requirements, between SRCNN and FSRCNN was the sacrifice in speed and the subsequent increase in throughput. FSRCNN's structure was much deeper compared to SRCNN, but operated on the low-resolution images at 240p, instead of preprocessing the images using bicubic interpolation before feeding it into the network like SRCNN operates. As a result, there was a noticeable decrease in quality, in terms of the SSIM taking a hit, but the subsequent increase in throughput outweighed the downsides. Since we were initially meeting our quality metrics with SRCNN, and failing our timing metrics with FSRCNN, it made sense to pivot to the latter, because the quality metrics were still matched with the new implementation, but came with a significant increase in throughput.

4.3 Hardware Development Environment

While there are many different environments and languages for developing for hardware, we mainly considered three options: SystemVerilog, Vitis, and PYNQ. While VHDL is a widely used language in industry, none of our members had any experience with it at all, and so we did not treat it as an option at all, especially since it was a low-level RTL language like SystemVerilog.

Between Vitis and PYNQ, both high level synthesis formats, we went with Vitis over PYNQ, because we had more immediate resources and guidance for Vitis. Furthermore, as none of our members had experience with PYNQ it would have had to present very strong benefits over Vitis for us to consider it over that framework. It did not, and hence we did not select it.

Finally, between SystemVerilog and Vitis, we ultimately chose Vitis. This is because we knew we would be implementing a large-scale hardware system and iterating on it quickly. While RTL languages are good for precise control over what a system is doing, they do not lend themselves to fast, over-arching changes. This is where HLS frameworks like Vitis shine – the notion of hardware at the speed of software. Our choice of Vitis also allowed much easier integration between off-board memory and the programmable logic fabric. As Vitis manages AXI bus integration of user-defined hardware kernels automatically, we were able to treat these data transfers as if they were a simple function call. In Verilog, however, this would have entailed writing this transfer protocol ourselves, or at the very least managing its use and connection ourselves. Furthermore, for testing, Vitis allows our tests to be written in C, as well as providing an HLS environment that estimates the latency of the kernel based on the inferred hardware implementation. This made testing much faster and much easier than if it had been in Verilog.

The only drawback presented by Vitis, was the very same as its best selling point: that it does many things that a designer does not. Ultimately we decided that the benefits provided outweighed our concerns, as we wanted to ensure our ability to quickly iterate on optimising our CNN, as opposed to getting bogged down with communication and data transferring.

4.4 Hardware Acceleration Platform

We considered a few different platforms for implementing our hardware acceleration; broadly we considered FPGAs, ASICs, and GPGPUs.

Developing an ASIC in the amount of time provided for this project would be immensely difficult, as we would have to tape out some time in the middle of the course, vastly reducing the amount of time we had to develop our system. Additionally, ASICs, once created, are no longer able to be modified. As a result of all of these restrictions, we ruled out ASIC development as an option.

Between FPGAs and GPGPUs we had an interesting choice to make. On one hand, GPGPUs are usually better at generic graphical operations than FPGAs. On the other hand however, we were never intending to implement a *generic* graphical operation, rather we intended to implement a very *specific* graphical operation, down to the weights of our convolutions. Unlike a GPGPU, which would need to be general purpose enough to perform any convolution asked of it, we only wanted to perform convolutions with our specific kernels.

In this case, an ASIC in fact *would* have been the best option given far more time to implement our system, however, due to the restriction to a single semester, we ended up using an FPGA as a configurable ASIC.

As for our choice in FPGA, we compared the Ultra96v2, the Zynq UltraScale+ MPSoC ZCU104, and the Terasic DE0-CV FPGA. Video output on the DE0-CV would be difficult as well as file and memory transfer as it is just an FPGA and not an MPSoC. The only advantage it provided was familiarity, as we had all used it before in previous classes. Between the Ultra96v2 and the ZCU104, we would have prefered using the ZCU104, as it had more compute, specifically having more DSP blocks (1,728 vs 360) and a dedicated video codec unit. Unfortunately, we had to rule this out as it alone would have exceeded our provided budget by over three times, costing $1,554. Thus, we went with the Ultra96v2, which as an added bonus we had support and familiarity with through a course that one of our

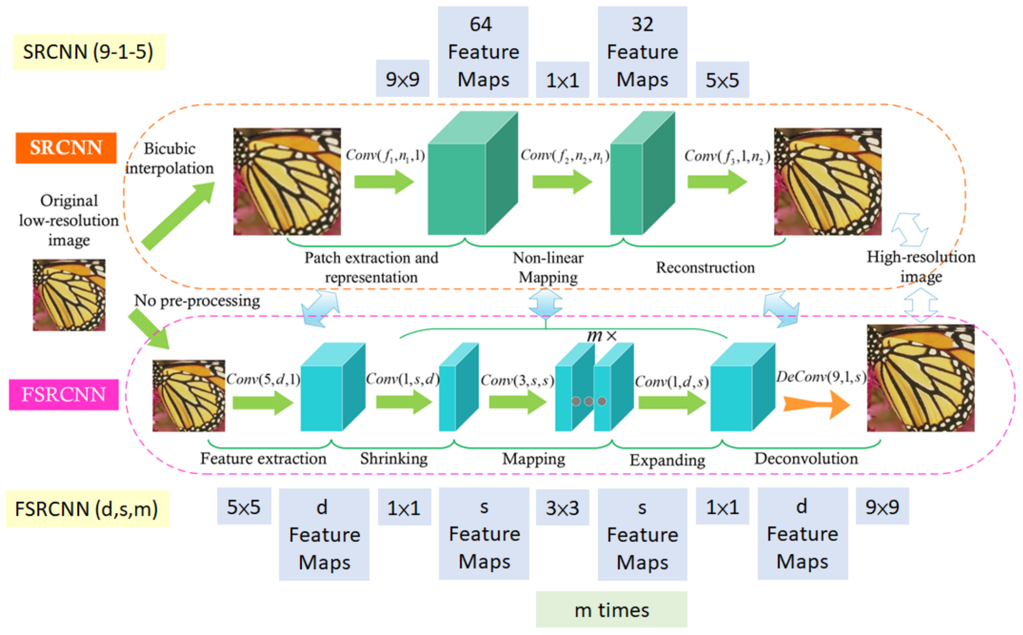


Figure 3: High Level Overview of the Software Model

members was taking concurrently with capstone.

5 SYSTEM DESCRIPTION

5.1 Software Model

The CNN for our upscaling portion was trained on a GPU on Google Colab, and also locally, on an Nvidia RTX 3070, and utilized the high computation speed during training in order to hit our super resolution target in terms of quality. The software model used the module PyTorch on Python, and utilized the speed of the GPU through cuDNN and CUDA. Originally, as detailed in the design report, the model was based off a paper[2] called SRCNN. Referring to the diagram below, the model first pre-processed the image by upscaling it to its final resolution through bicubic interpolation, and then passed the image through a 3-layered CNN. The three layers each corresponded to these functions: (1) a patch extraction block (2) a non-linear mapping block (3) and reconstruction from these higher resolution patches. In terms of the reasoning, the second layer adds non-linearity to the model, which is desired since it allows a more information-rich mapping between the input and output. In comparison to FSRCNN, our original model was much more shallow, but it processed a lot more data, since it was operating on the 1080p frames.

On the other hand, FSRCNN operates on the low-resolution, 240p frames. It has a deeper network, but it is offset by the massive decrease in the amount of data being processed. Referring to the diagram, you can see that there is no pre-processing being done on the image before the upscaling process is conducted. Instead, a deconvolution layer is added at the end, in order for the CNN to output an image at the correct, upscaled resolution.

To go in-depth about the differences and the hyperparameters chosen originally for our SRCNN implementation, we originally tested our model with a (9-1-5) structure, meaning we had a filter size of 9x9, 1x1 and 5x5 respectively for the three layers, with the correct padding added. We also had 128 filters for the first layer, as well as 32 for the second layer. This first attempt gave us a reasonable baseline comparison to work off of, since those were similar to the hyperparameters used in the paper. After swapping to a (9-5-5) structure and training for the same number of iterations, there was a clear increase in the video quality. In contrast to the numbers on the paper, the new structure was only around 3x slower, whereas the paper claimed a much higher number[2]. Overall, the increase in quality was high enough to justify the decrease in throughput.

However, after finding out that our timing was unattainable on the Ultra96, we swapped to using FSRCNN[1]. Since we didn't have much time left, we opted to immediately use the hyperparameters listed on the FSRCNN paper, and to start optimizing on them on hardware before training the model fully. Referring to the diagram, we started with FSRCNN(56, 12, 4), and used the filter sizes and strides listed on the paper. We then swapped to FSRCNN(35, 5, 1), which was a smaller version of FSRCNN with less parameters, but with slightly decreased quality overall. Overall, the trade-off in quality led to a several order of magnitude increase in throughput, so it was justified.

5.2 Hardware Super Resolution

We implemented hardware super resolution for the Ultra96v2's programmable logic using Xilinx's Vitis Unified Software Platform, specifically Vitis and Vitis HLS. We used Vitis HLS to have faster iteration cycles for hardware optimisation, and Vitis for compiling to the board and packaging the hardware kernels for the Ultra96.

We followed hardware acceleration advice regarding tiling operations and rearranging for better memory access patterns from papers[6, 4] documenting FPGA acceleration of CNNs. While these optimisations gave some increases in performance, they did not allow us to achieve the full performance we required, settling around 200 seconds per frame. Our next steps were to examine breaking some of the best practices given in the papers.

5.2.1 Non-Uniform Tile Sizes

In [6], the authors recommend against using non-uniform tile sizes for optimising convolution loops in hardware. They say this because by having uniform tile sizes, hardware designers will converge on an optimised model faster; the code will be more readable, comprehensible, and maintainable; and the improvement is only on around a 7% to 10% difference.

Implementing this, we found it to be even less fruitful than what they described. This was because in order to speed up one layer by increasing its tile sizes, we had to degrade others, as we were limited by our hardware's limited number of logic elements, specifically BRAMs and LUTs.

5.2.2 Fixed Weights

Another optimisation that we attempted was using fixed weights. In our reference papers, the designs included AXI bus interfaces for the input data, output data, and weight data. We knew what weights we would be using beforehand, and thus could forego allowing generalities like allowing arbitrary weights to the system.

This actually provided us great speedup, however, for both the SRCNN and FSRCNN models we were not able to use it. This is because this optimisation pushed BRAM, DSP block, and FF utilisation. As our board has a relatively low number of DSP blocks, only 360, this quickly became a limiting factor for this optimisation.

5.2.3 Model Size

As we were unable to reach our target throughput and latency for our initial model, SRCNN, we looked at model size in terms of number of GOPS as a metric to reduce. Looking at this we decided to implement FSRCNN which had 1.38GOPS as opposed to SRCNN's 118.58GOPS. While we saw a large amount of improvement, we still were not able to meet our target throughput and latency with this optimisation. Furthermore the aforementioned optimisation, using fixed weights, still posed a problem in that it spiked utilisation, restricting us to smaller, less optimal designs.

5.2.4 Kernel Count

Our last observation was that our fixed-weight optimisation was increasing utilisation due to the number of kernels. While SRCNN and FSRCNN had vastly different numbers of operations, as detailed above, they have similar numbers of kernels: 2144 for SRCNN and 2032 for FSRCNN. We thus implemented yet another model FSRCNNs, a variation on FSRCNN. This model had fewer still operations than FSRCNN, only 0.48GOPS, but more importantly had much fewer kernels: only 445. With this change in implementation, we were able to leverage the fixed-weights optimisation to achieve latency of 250ms, just barely meeting our adjusted latency requirement. As for our throughput however, as it was latency-bound as described in section 6.3, we were still unable to satisfy this specification.

5.3 Video Display

Video display was handled by the ARM core, as opposed to the PL fabric. As PetaLinux provides out-of-the-box support for OpenCV and the Ultra96v2 supports native mini-DisplayPort output at 1080p this portion was fairly straight forward to implement on the proper hardware.

Finding and using the proper hardware for this was not trivial however. While the Ultra96v2 supports mini-DisplayPort out, and claims to support active conversion to HDMI, this is not necessarily always the case. Due to undefined behaviour in the 2020.1 version of Petalinux regarding output display, certain displays, display resolutions, and display conversions were not supported. We found that the only display resolutions supported were 1080p and 720p, meaning the connected monitor had to support one of these, if not, display would fail to work.

To diagnose this further, we investigated running with PYNQ to get video display up on a HDMI based projector. While no video output was generated even in this configuration, we were able to get more useful information from `dmesg` in this OS. It reported the step of generating output was not causing an error but the connection that we had was causing the issue. Thus we were able to determine that the discrepancy must be happening either in conversion from mini-DisplayPort to HDMI or on the display itself.

To further solve this we switched to using the lab monitors and mini-DisplayPort to DisplayPort converters. This worked, but reduced the efficacy of our system. After all, the most visible case for super resolution is when the image is blown up on a large screen, as is done with a projector. A monitor reduced our ability to visually see differences as provided by our super resolution algorithm.

Once working, we were able to visually see slowdowns in outputting video to display from the Ultra96v2. This is almost certainly because the computation was done on the ARM core, as opposed to any specialised video processing unit. While we did not have the time to benchmark how many frames per second videos were degraded to, by a visual inspection video playback was around 10-20FPS, as

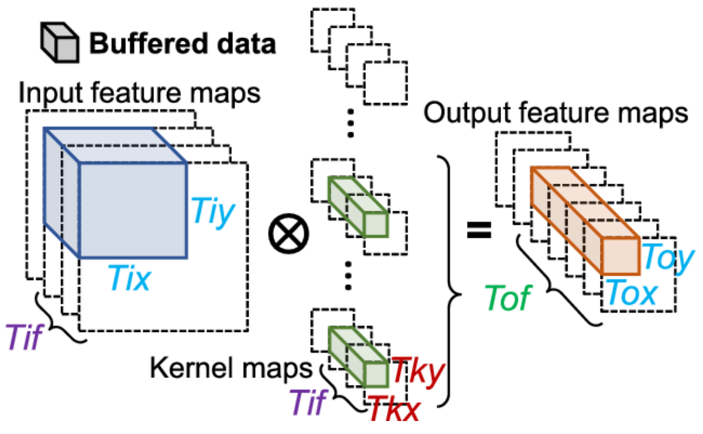


Figure 4: Graphical depiction of loop tiling operations for better hardware computation

frame stuttering was very noticeable, but it still maintained the illusion of motion, presenting itself as a video.

5.4 Glue Logic

As the completion our system was delayed for various reasons as detailed below in sections 7 and 10, it was not in scope for our timeline to implement a good UI/UX. Instead we implemented a terminal-based UI that glued our different programs together via shell scripts.

This, as a solution, is not elegant. It directly contradicts our initial proposition to provide the general public a more usable system. This solution however, provides one redeeming quality: while produced on a razor thin timeline, it is both reliable and functional.

Thus this section, while having no intention of being part of our design from its conception was added in as a shim to have something that, at the very least, worked and could be shipped at the end of our project.

5.5 Peripherals

As the design intent of our system was aimed at the general public, we made sure that our system was modular. Thankfully the Ultra96v2 supports most input methods, so testing it with a standard keyboard (a Dell KB216) and two standard mice (a Dell MS116 and Logitech M-R0012) we found no problems. Due to the limited number of ports on the Ultra96v2, these had to be run through a USB hub for the final system, but this presented no issue in functionality either.

While the Ultra96v2 was robust in accepting various inputs, it was less robust in accepting various display methods. We tested a Sony MP-CD1 mobile projector, as well as other various HDMI monitors. As the Ultra96v2 outputs mini-DisplayPort, we used an mini-DisplayPort to HDMI converter. First attempts using a passive converter were not fruitful, and so we referenced documentation and support pages by Xilinx. We found that the Ultra96v2 supported mini-DisplayPort to HDMI conversion only via active converters because of the way in which the device drives the display on a hardware level. So we then switched to the CableCreation CD0095, one of Xilinx's recommended mini-DisplayPort to HDMI converters. Testing with this and the HDMI monitors still led to no success. After this, we did more investigation, and found some user reports of the Ultra96v2 not supporting video output in various configurations depending on conversion, OS, OS version, the exact monitor to which it was connected, and so on. We finally settled on using the displays which were in the lab with a mini-DisplayPort to DisplayPort converter. While this display configuration finally worked, it definitely slowed down our testing abilities by restricting where we could test display, as well as contributing to a delay in getting this part of the system up and running. Furthermore, this presents itself as a detriment to the general user. If a user does not have a DisplayPort enabled display, they would not be able to use our device.

6 TEST & VALIDATION

6.1 SSIM

First, we define the method in which we calculate SSIM as a video metric, as from the paper. We define $SSIM_{ij}^{x}$ to be the SSIM index value of the x-th color component of the j-th sampling window in the i-th video frame. Then let W_x be the weight of the x-th color component, the values of which are taken from the paper by Wang, Lu, and Bovik. Now, evaluating the video adapted version of the SSIM index, we have the following:

$$SSIM_{ij} = \sum_{x \in \{Y, Cb, Cr\}} W_x SSIM_{ij}^{x} \tag{2}$$

where we evaluate over the Y, Cb, and Cr color channels,

$$Q_i = \frac{\sum_{j=1}^{R_s} w_{ij} SSIM_{ij}}{\sum_{j=1}^{R_s} w_{ij}} \tag{3}$$

where R_s is the number of sampling windows per video frame,

$$Q = \frac{\sum_{i=1}^{F} \mathcal{W}_i Q_i}{\sum_{i=1}^{F} \mathcal{W}_i} \quad (4)$$

where F is the number of video frames, and $\mathcal{W}_i$ is the weight of the i-th video frame. Then Q is the quality of the video and Q_i was the quality of a single frame of video, specifically the i-th frame.

In our case, for all $j \in R_s$ and $i \in F$, we maintained that $\mathcal{W}_i = 1$, as well as $w_{ij} = 1$. In other words, we went with the more straightforward approach, which involved calculating the SSIM between two of the same frames on the upscaled video and the reference video, whilst using equal weights for all individual pixels, and then taking the average SSIM out of the all the frames, whilst using equal weights for all individual frames. The reason for this was two-fold: Having non-trivial weights would be more computationally complex, whilst having more general weights would generalize it to more different types of videos, which is exactly what our dataset and use case was.

The results of SSIM on our dataset are included in table 1.

An important thing to note is although there was some inconsistency between the algorithms, e.g. SRCNN-EX sometimes performed slightly worse compared to FSRCNN, overall, all implementations outperformed bicubic *almost* on every single video in our dataset.

6.2 Latency

To evaluate latency, we used the accelerator host program to benchmark the duration of the execution of the hardware kernel on a single frame. To measure the time elapsed, we used the `std::chrono` library, giving us resolution in miliseconds. In 5 we have the latency of bicubic interpolation run on an AMD Ryzen 5 3600, a typical consumer grade CPU. The super resolution algorithms are measured as implemented on the Ultra96v2. Anything under 250ms is what we considered ‘real-time,’ with respect to latency.

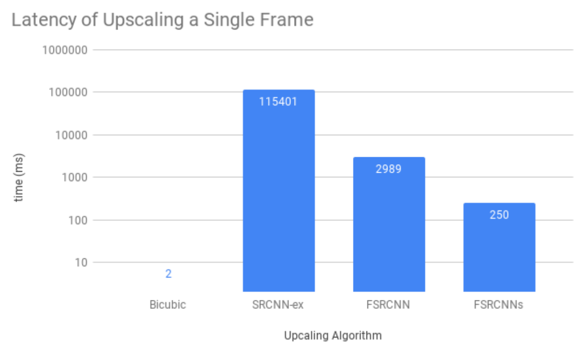


Figure 5: Single frame latency of different upscaling algorithms, plotted on a logarithmic scale.

As seen in 5, the amount of time to upscale an image with bicubic interpolation is almost negligible – the bar barely registers on the plot. As for the other algorithms, we can see steady improvements with the reduction in size of the model, though not linearly compared to number of operations per second, which points to overhead cost involved in the system. As in table 2, we were able to achieve operation throughput in the MOPS/s range, with less OPS/s for FSRCNN than SRCNN-ex. This is because of the amount of time to implement the model; while FSRCNN is smaller and has far less operations, it is still fairly large and our timeline provided us with much less time to optimise it than SRCNN-ex. The FSRCNNs model has a higher operation throughput than the other two, which seems counter-intuitive, as it was the model which we had the least number of weeks to optimise for hardware. While this was the case, we spent the same amount of time optimising FSRCNNs as FSRCNN, as well as leveraging the optimisations in FSRCNN, as the models are similar, much moreso than FSRCNN compared to SRCNN-ex.

One point to note, however, is that estimating target latency with the average number of Giga-operations per second that Xilinx engineers are able to achieve for optimisation examples provided in Vitis documentation, some of our models should ave been able to meet our specifications for latency and throughput, even with latency-bound throughput; namely the FSRCNN and FSRCNNs models.

6.3 Throughput

Our accelerator host program, running on the ARM core of the Ultra96v2, ultimately contributed to a limitation in our throughput. While throughput and latency are inherently different metrics, our implementation of frame processing was iterative and sequential, thus resulting in our throughput being a direct function of our latency. More specifically our host program limited our throughput to be bound by our latency, arriving at a constant throughput of the reciprocal of our latency. As our system’s throughput was bound by its latency, throughput we can talk about *throughput-bound latency*, as this is what ended up being the relevant throughput metric for out system.

This meant that in order to meet our specification for throughput, we would need to achieve a latency of no greater than 33ms. Further for the bare minimum of video-grade throughput we would need no greater than 83ms! As the fastest we were able to reliably achieve was 250ms, we did not meet this metric, and were an order of magnitude off, at best.

Referring back to table 2, we can see again that the ideal time metrics would have allowed us to reach even latency-bounded throughput, however due to constraints on design time, as well as familiarity with the design environment led us to ultimately not meet this ideal bound.

Table 1: Super Resolution Quality (SSIM)

Video Number	SRCNN-Ex	FSRCNN	FSRCNN-s	Bicubic
1	0.725	0.711	0.705	0.693
2	0.698	0.701	0.677	0.675
3	0.639	0.672	0.672	0.623
4	0.741	0.705	0.699	0.694
5	0.953	0.935	0.918	0.908
6	0.733	0.657	0.655	0.652
7	0.783	0.795	0.771	0.772
8	0.851	0.769	0.758	0.749
9	0.751	0.689	0.685	0.671
11	0.685	0.652	0.667	0.650
12	0.727	0.717	0.715	0.706
13	0.721	0.665	0.651	0.643
14	0.759	0.759	0.727	0.725
Mean	0.751	0.724	0.715	0.705
Variance	6.18×10^{-3}	5.92×10^{-3}	5.07×10^{-3}	5.56×10^{-3}

Table 2: Super Resolution Timing on Ultra96v2

Algorithm	Time (ms)	GOPS	GOPS/s	Ideal Time (ms)
SRCNN-ex	115401	118.58	0.103	593
FSRCNN	2989	1.38	0.046	7
FSRCNNs	250	0.48	0.190	3

7 PROJECT MANAGEMENT

7.1 Schedule

Our schedule for the project had to undergo a significant change, most notably due to the change in our implementation during the project. We essentially had to go through two cycles of model development, both on the software and hardware (Ultra96) portions, from training an entirely new CNN with a different structure, as well as having to redo a lot of previous optimizations on our hardware in order to reach timings on our Ultra96 implementation. Since a lot of the previous infrastructure for training and benchmarking was already in place, as well as having a lot more familiarity when it came to working on the model as a whole, we managed to create a functioning implementation on the Ultra96 board for our second model in about a third of the time it took for our first model. For the finer details of what we set out to do versus what we accomplished, the Gantt chart below illustrates the project as a whole.

7.2 Team Member Responsibilities

As stated in the previous design report, Kunal & James planned on handling the hardware implementation of the image upscaling algorithm. Joshua worked on the software model, and we planned on having input from other members, in terms of the design. We all had planned on contributing to the algorithm development aspect of the project. During the development of the project, we ended up having very distinct parts, with Joshua fully taking over the software model, including setting up cloud computing, developing model and training, benchmarking all implementations etc. James essentially took over the entire Ultra96 implementation, covering all parts including hardware optimization and acceleration, benchmarking, verification, I/O etc. At the beginning, Kunal was assigned the task of developing, testing and integrating I/O. He was also supposed to be in charge of benchmarking our final implementation, as well as developing UI for a user-friendly experience. However, little to no progress was made on all fronts, and the project never developed to the point where UI could even be considered. James also ended up rewriting a lot of the I/O code provided by Kunal.

7.3 Budget

Our project was fairly low-budget. While the Ultra96v2 development board cost $249, one of our members, James, was taking the class Reconfigurable Logic which provides the board on loan to its students. Because of this, we were able to use the board at no cost to our group. The board peripherals such as the monitor, USB hub, mouse, keyboard, etc., were designed to be modular. As such, they were able to be provided by our team's members or borrowed from labs. Note then, that this is why in the bill of materials these parts are listed generically, without a specific model number nor manufacturer; this is not an omission, rather a demonstration of our system's modular nature. For Google Colab, we paid out of pocket due to an immediate need for our code to be run, and because of the low cost of it overall.

Table 3: Bill of materials

Description	Model #	Manufacturer	Quantity	Cost @	Total
Ultra96v2 Board	AES-ULTRA96-V2-G	Avnet	1	$249	$0
Google Colab	*	Google	1	$10	$0
AWS	*	Amazon	1	$50	$50
					$50

Amazon Web Services usage is further detailed in the next subsection.

7.4 AWS Credit Usage

We initially planned on using AWS for the entirety of our project, to host our training code and to utilize the high-performance, cost-effective GPU instances that they provide. We chose to use the Amazon EC2 P3 instance type, with a single GPU instance (p3.2xlarge), due to it being high-performance and cost-effective. With the NVIDIA V100 GPU, we chose this specific instance type due its effectiveness for deep learning training, which helped with training our software model very well. On paper, it suited the development of our software model extremely well. However, we ended up only running our code on a smaller instance with the first $50 provided, and later transitioned to running our code on two systems: Google Colab, as well as locally on an Nvidia RTX 3070 GPU. This was mostly due to an internal issue on our part, and in the short time we utilized AWS, we found it very convenient and reliable. We give our thanks to Amazon for providing us with AWS credits to aid in our project at Carnegie Mellon University.

7.5 Risk Management

Despite us having a risk diagram for our design review report, as well as evaluating each risk and coming up with ways to mitigate them, the largest problem that we encountered was our throughput on the Ultra96 implementation not meeting our ideal case. Since our initial schedule had placed full confidence in our initial choice of algorithm to work, we had not taken enough precaution to mitigate the chance that it wouldn't work. Our initial risk mitigation involved leaving 2 weeks of slack time for ourselves at the end of the semester, but as we had a pivotal transition in the middle of the project, those 2 weeks quickly disappeared, and our workload close to doubled or tripled during the last couple of weeks. Essentially, the risk of our design had been underestimated greatly, leading to an inconsistent amount of workload during the semester. To increase throughput, we had to sacrifice on quality, but we were careful to ensure that our quality never lost against bicubic, since that was our baseline, and would defeat the purpose of our project.

Another risk that we took into account and mitigated was the fact that our software model could have been delayed. Since the hardware implementation couldn't begin without significant information about the hyperparameters and structure of the software model, we did research on fallback, off-the-shelf models which had the model structure and weights available, and could be substituted in case our software model completely failed to work. Thankfully, our software model worked in the end, and although a delay in starting our software model shifted back our schedule, we managed to recover from that risk.

Finally, a major risk we had taken too lightly at the beginning was the effect of workload from other classes throughout the semester, as well as our members having an unequal amount of work planned out - some members had a lot of work near the beginning, and some members had a lot of work near the end. Whilst we had major milestones of the Capstone class panned out, we had less consideration of other classes incorporated into our schedule. This not only ended up affecting our schedule, it also meant that communication was reduced, as our sleep schedules and availabilities didn't always align. We were completely aware of this flaw by the design review, as well as the risk of burnout from each member, so we set out to equalize the amount of work being distributed between members on a week-by-week basis, using our weekly reports as an indicator to allocate tasks to other members who initially were not scheduled to do those tasks.

8 ETHICAL ISSUES

Since this product lowers the bar for entry for using super resolution, it runs the risk of making super resolution more readily available to bad actors to use it for malicious purposes. One possibility is that bad actors who want to spy on others could purchase and use lower quality cameras and upscale the footage using this device. Having lowered the bar for entry, this means that this could be come a more proliferated problem.

The accuracy and correctness of the super resolution could also play a key role with regards to the end user. For example, someone might use our upscaling tool to increase the quality of some low-quality security camera footage, with the intention of identifying a criminal from a crime scene, or a license plate from a car etc. Although this is not within our use case and we could actively discourage people from using it in such scenarios, there is still a chance that upscaled videos could be used in scenarios such as inaccurate evidence, leading to the identification of the wrong person or the wrong license plate, for example.

As the function of super resolution is to extract information from blurry frames, videos which have been blurred to anonymize them run the risk of this transformation being undone to some extent and the anonymization being undone. This is especially relevant for legal proceedings in which footage of defendants is often blurred to preserve their safety and privacy. Related to this is the possibility that low-quality security camera footage is upscaled and used as evidence to implicate an individual in a crime. Since the fine details of the frame are created by the algorithm, the defining features of a person may cause a false match to occur. If used as evidence, this could lead to a wrongful conviction.

Another way in which our product could be used by a bad actor to deanonymize video footage is with pornographic or lewd content. These types of videos are often blurred to protect the identity or decency of the individuals depicted, but if it is able to be undone, this presents a risk to the individuals involved.

The risk of deanonymization is easily avoided by using black-bar anonymization as opposed to blurring anonyization. On the other hand, if determined to anonymize via blurring, by being sure to blur strongly enough, the details of the footage can be kept anonymous. However, as for the other risks, there is nothing outside of new or existing laws, or user licence agreements which could be done to prevent them.

9 RELATED WORK

Since video upscaling is a hot topic, there are many related papers, ideas and solutions out there that demonstrate upscaling using various machine learning techniques, on various forms of hardware. One paper that is most closely related to our project is [3], which also uses an FPGA to perform super resolution. There are other super resolution algorithms as well, but we did not implement these in our project. For instance there are those using GANs, or relying more heavily on advanced DSP methods.

10 SUMMARY

Our system was not able to meet our design specifications. By the end of the semester we realised through our system testing that, given the Ultra96v2 as our hardware acceleration device, we had to choose either a decrease in performance or a decrease in algorithmic complexity. Our ability to get performance out of the Ultra96v2 was limited by our familiarity with both the board itself and our familiarity with Vitis, the environment which we used to program it. Beyond this, our system was also limited by the size of the board itself.

The easiest way to allow our system to meet our specification would be to get more powerful hardware.

10.1 Future Work

Since our system had not been completed to our initial requirements, there is a lot of potential for further improvements to our project. Since the requirement we couldn't fulfill was timing, that meant any future work to improve the system would be to address the throughput, specifically, increasing it to the point where it could run in real-time. Despite our rapid and massive increase in throughput over the last few weeks of the project, replicating such a difference would have been considerably hard and possibly unrealistic, given the fact that any further optimizations would have given diminishing returns.

To address this, one thing we could do is instead of using Vitis, we could rework the architecture on the Ultra96, and rewrite our implementation on a much lower level, such as through Verilog. This would allow to get the fine-level of optimization needed to improve the throughput of our design further, with major milestones being 12FPS, 24FPS and 30FPS respectively. A throughput of 12FPS would have been the minimum for people to perceive a collection of images as a video, 24FPS would have been the industry standard and the frame rate movies are shot at, and 30FPS would have been our initial requirement, and the frame rate of the videos in our dataset.

Another thing that is relevant to our possible future work would be to explore using more powerful hardware. Although we didn't explore using a more powerful FPGA due to budget constraints and also our use case being targeted towards a more general audience, a more costly board would allow us to explore other scaling factors on higher resolutions, such as from 1080p to 4k or even 8k resolutions. This could possibly have greater relevance in the future, as higher resolution displays become more widely adapted. This would also require rethinking the choice of algorithm, which would involve carefully considering how increase of several orders of magnitude of data would be addressed, and how the quality could still be retained whilst keeping real-time expectations.

10.2 Lessons Learned

One of the biggest lessons we learned was our scheduling process. Specifically, our initial schedule did not leave enough time for us to iterate, and we were too optimistic when allocating time for the optimization on the U96 board. A much better approach would have been to utilize some form of hardware-software co-interaction, and have much tighter iteration cycles in order to discover and change or shift our project early on. This would have allowed us to discover the unrealistic timings for our first implementation much earlier, and allowed us time to pivot and swap to a different implementation without tripling our workload in the last month. In short, a more consistent use of time would have allowed for a much smoother integration process between our hardware and software sections, which could have been achieved with a tighter and well-documented schedule.

Glossary of Acronyms

- ASIC - Application Specific Integrated Circuit

- BRAM - Block RAM (Random Access Memory)
- CNN - Convolutional Neural Network
- CPU - Central Processing Unit
- DSP - Digital Signal Processing
- FF - Flip Flops
- FPGA - Field Programmable Gate Array
- FPS - Frames Per Second
- FSRCNN - Fast Super Resolution Convolutional Neural Network
- FSRCNNs - Fast Super Resolution Convolutional Neural Network (small)
- GAN - Generative Adverserial Network
- GPGPU - General Purpose Graphical Processing Unit
- GPU - Graphical Processing Unit
- HD - High Definition
- HLS - High Level Synthesis
- IO - Input / Output
- HLS - High Level Synthesis
- LUT - Look Up Table
- ML - Machine Learning
- MPSoC - Multi-Processor System on Chip
- MSE - Mean Squared Error
- OPS - Operations
- PL - Programmable Logic
- PSNR - Peak Signal to Noise Ratio
- RTL - Register Transfer Language
- SD - Standard Definition
- SRCNN - Super Resolution Convolutional Neural Network
- SRCNN-ex - Super Resolution Convolutional Neural Network (extended)
- SSIM - Structural Similarity Index Measure
- UI - User Interface
- UX - User Experience
- VMAF - Video Multimethod Assessment Fusion

References

[1] Chao Dong, Chen Change Loy, and Xiaoou Tang. *Accelerating the Super-Resolution Convolutional Neural Network*. 2016. arXiv: `1608.00367 [cs.CV]`.

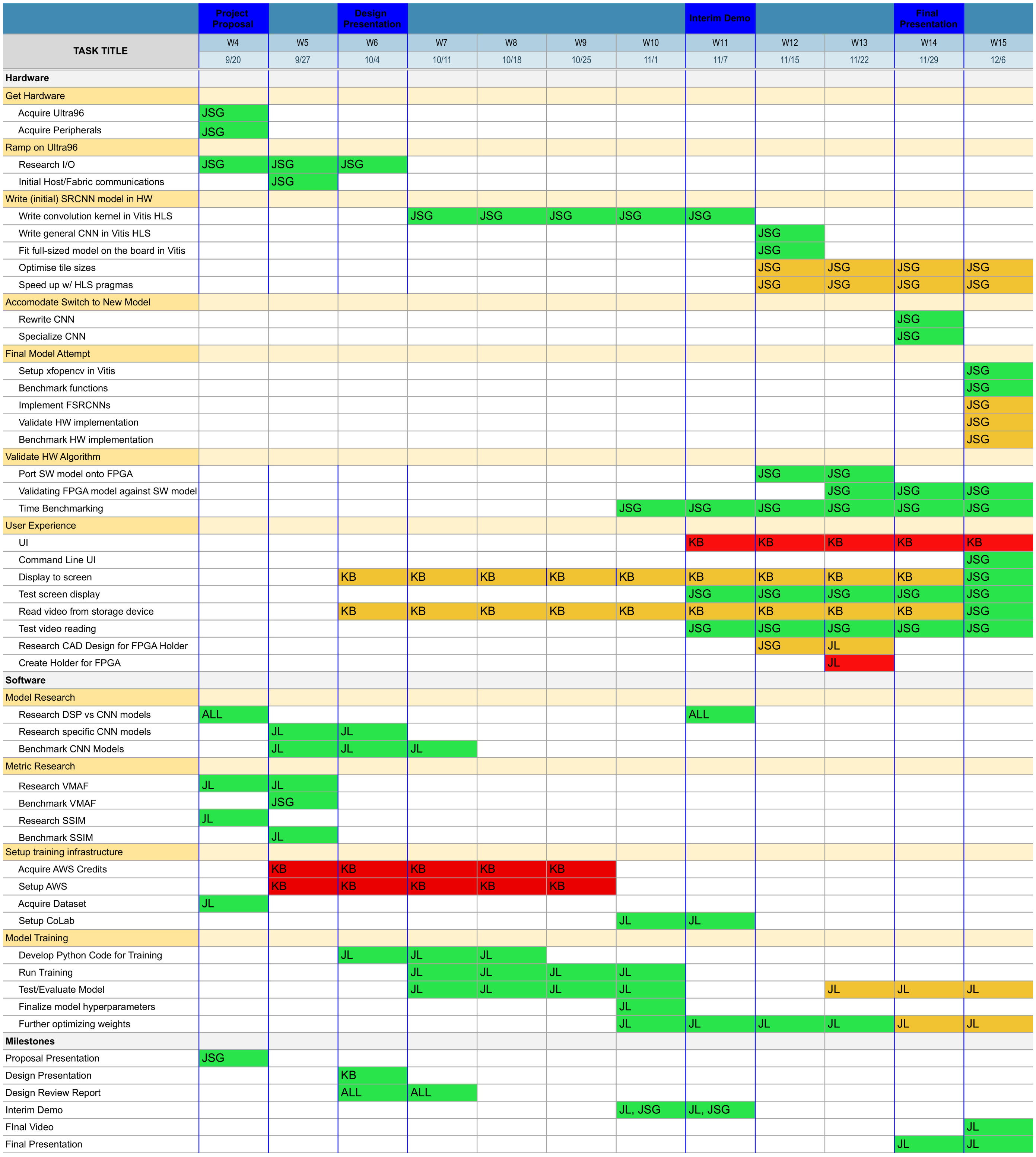
[2] Chao Dong et al. *Image Super-Resolution Using Deep Convolutional Networks*. 2015. arXiv: `1501.00092 [cs.CV]`.

[3] Zhuolun He et al. "FPGA-Based Real-Time Super-Resolution System for Ultra High Definition Videos". In: *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 2018, pp. 181–188. DOI: `10.1109/FCCM.2018.00036`.

[4] Yufei Ma et al. "Optimizing the Convolution Operation to Accelerate Deep Neural Networks on FPGA". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26 (2018), pp. 1354–1367.

[5] Zhou Wang, Ligang Lu, and Alan C. Bovik. "Video quality assessment based on structural distortion measurement". In: *Signal Processing: Image Communication* 19.2 (2004), pp. 121–132. ISSN: 0923-5965. DOI: `https://doi.org/10.1016/S0923-5965(03)00076-6`. URL: `https://www.sciencedirect.com/science/article/pii/S0923596503000766`.

[6] Chen Zhang et al. "Optimizing FPGA-Based Accelerator Design for Deep Convolutional Neural Networks". In: *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. FPGA '15. Monterey, California, USA: Association for Computing Machinery, 2015, 161–170. ISBN: 9781450333153. DOI: `10.1145/2684746.2689060`. URL: `https://doi.org/10.1145/2684746.2689060`.



	Project Proposal		Design Presentation					Interim Demo			Final Presentation	
TASK TITLE	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14	W15
	9/20	9/27	10/4	10/11	10/18	10/25	11/1	11/7	11/15	11/22	11/29	12/6
Hardware												
Get Hardware												
Acquire Ultra96	🟩 JSG											
Acquire Peripherals	🟩 JSG											
Ramp on Ultra96												
Research I/O	🟩 JSG	🟩 JSG	🟩 JSG									
Initial Host/Fabric communications		🟩 JSG										
Write (initial) SRCNN model in HW												
Write convolution kernel in Vitis HLS				🟩 JSG	🟩 JSG	🟩 JSG	🟩 JSG	🟩 JSG				
Write general CNN in Vitis HLS									🟩 JSG			
Fit full-sized model on the board in Vitis									🟩 JSG			
Optimise tile sizes									🟨 JSG	🟨 JSG	🟨 JSG	🟨 JSG
Speed up w/ HLS pragmas									🟨 JSG	🟨 JSG	🟨 JSG	🟨 JSG
Accomodate Switch to New Model												
Rewrite CNN											🟩 JSG	
Specialize CNN											🟩 JSG	
Final Model Attempt												
Setup xfopencv in Vitis												🟩 JSG
Benchmark functions												🟩 JSG
Implement FSRCNNs												🟨 JSG
Validate HW implementation												🟨 JSG
Benchmark HW implementation												🟨 JSG
Validate HW Algorithm												
Port SW model onto FPGA									🟩 JSG	🟩 JSG		
Validating FPGA model against SW model										🟩 JSG	🟩 JSG	🟩 JSG
Time Benchmarking							🟩 JSG	🟩 JSG	🟩 JSG	🟩 JSG	🟩 JSG	🟩 JSG
User Experience												
UI								🟥 KB	🟥 KB	🟥 KB	🟥 KB	🟥 KB
Command Line UI												🟩 JSG
Display to screen			🟨 KB	🟨 KB	🟨 KB	🟨 KB	🟨 KB	🟨 KB	🟨 KB	🟨 KB	🟨 KB	🟩 JSG
Test screen display								🟩 JSG	🟩 JSG	🟩 JSG	🟩 JSG	🟩 JSG
Read video from storage device			🟨 KB	🟨 KB	🟨 KB	🟨 KB	🟨 KB	🟨 KB	🟨 KB	🟨 KB	🟨 KB	🟩 JSG
Test video reading								🟩 JSG	🟩 JSG	🟩 JSG	🟩 JSG	🟩 JSG
Research CAD Design for FPGA Holder									🟨 JSG	🟨 JL		
Create Holder for FPGA										🟥 JL		
Software												
Model Research												
Research DSP vs CNN models	🟩 ALL							🟩 ALL				
Research specific CNN models		🟩 JL	🟩 JL									
Benchmark CNN Models		🟩 JL	🟩 JL	🟩 JL								
Metric Research												
Research VMAF	🟩 JL	🟩 JL										
Benchmark VMAF		🟩 JSG										
Research SSIM	🟩 JL											
Benchmark SSIM		🟩 JL										
Setup training infrastructure												
Acquire AWS Credits		🟥 KB	🟥 KB	🟥 KB	🟥 KB	🟥 KB						
Setup AWS		🟥 KB	🟥 KB	🟥 KB	🟥 KB	🟥 KB						
Acquire Dataset	🟩 JL											
Setup CoLab							🟩 JL	🟩 JL				
Model Training												
Develop Python Code for Training			🟩 JL	🟩 JL	🟩 JL							
Run Training				🟩 JL	🟩 JL	🟩 JL	🟩 JL					
Test/Evaluate Model				🟩 JL	🟩 JL	🟩 JL	🟩 JL			🟨 JL	🟨 JL	🟨 JL
Finalize model hyperparameters							🟩 JL					
Further optimizing weights							🟩 JL	🟩 JL	🟩 JL	🟩 JL	🟨 JL	🟨 JL
Milestones												
Proposal Presentation	🟩 JSG											
Design Presentation			🟩 KB									
Design Review Report			🟩 ALL	🟩 ALL								
Interim Demo							🟩 JL, JSG	🟩 JL, JSG				
FInal Video												🟩 JL
Final Presentation											🟩 JL	🟩 JL

Figure 6: Gantt Chart – Initials signify task owner; colour signifies progress on the task, with green, yellow and red being completed, partially completed, and no progress, respectively