FastScale: Hardware-Accelerated Upscaling

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

***Abstract*—Upscaling is a process that increases the quality of a video or image to a higher resolution format, whilst still displaying correctly. This problem has been explored by various researchers and companies, and there are many existing software-based solutions. We are implementing a hardware-based, real-time video upscaling tool, that correctly upscales low-resolution videos to higher resolutions, up to 1080p. We aim to provide a plug-and-play solution that is both convenient and intuitive to everyone, which will cut down any time and effort wasted on traditional approaches, such as having to upload onto a device and running super-resolution on the videos. Our upscaling algorithm is based off a paper exploring single-image super-resolution using a CNN. [1]**

***Index Terms*—CNN (Convolutional Neural Network), FPGA, HD (High Definition), SD (Standard Definition), (key words or phrases in alphabetical order, use your own words, delete the ones listed as an example)**

1 INTRODUCTION

In the modern age, consumers have come to expect high resolution, low-latency, on-demand video options. While higher definition video formats and displays have become more widespread, legacy content still uses lower resolution formats more typical of the time in which they were created.

Solving this issue has presented consumers with a three-way choice where no choice is optimal. They could watch the video at lower resolution, but this would be suboptimal because of the modern high-resolution expectation. They could upscale their video through a third-party web-application, but this raises privacy concerns; how would one know what the site will do with one's data. Lastly, they could upscale their video by spinning up their own super-resolution model on their own computer, but for the vast majority of people, this is an extremely high technical barrier to entry.

To solve this issue, we are building a standalone, plug-and-play device that can take a low resolution video and upscale it to a higher resolution in real time. More specifically, our device takes Standard Definition (SD) video input with dimensions of 426x240 (the same dimensions as NTSC widescreen or 240p widescreen) and upscales it to High Definition (HD) video with dimensions of 1920×1080 (Full HD). Our video streams will operate at 30 frames per second (FPS), as that is an industry standard for movies, television, and other streamed content which make up our scope. Additionally, the native video output of the Ultra96 board is 60FPS, and so the conversion from 30FPS is trivial. Hence, our device maintains a throughput of $\tilde{3}3.3$ms per image so there is no need for video buffering. Our device adheres to the EBU Technical Recommendation R37 – 2007 for the synchronisation of audio and video on audio-video streams for consumer entertainment, so, the latency of the device is no more than 60ms, ensuring that the audio and video outputs remain synchronised.

There currently exist software which do real-time video upscaling, however, the degree or factor of upscaling is less than what our device is capable of. Other real-time video upscaling algorithms are sufficient for scaling factors of two, and adequate for factors of 3, however begin to produce artifacts and unfaithful reconstructions of the original image at higher scaling factors. Our device operates with a scaling factor of 4.5, eclipsing the reliable working domain of the most popular current real-time video upscaling algorithm(s).

Our approach is also hardware accelerated. This gives us the advantage of a lower latency and higher throughput. Furthermore, with our hardware acceleration being implemented on an FPGA, this allows for our device to be reconfigurable; if a new model is found to upscale videos more accurately, the user would not need to purchase a new device, but the upgrade would be downloadable. Further still, an FPGA gives us more flexibility to experiment with improvements not possible on certain ASICs which may, on the surface, present themselves as equally applicable. A GPU, for instance, may perform well for matrix computation, but would be less applicable if implementing a systolic array based method of computation. GPU's also have the drawback of their relatively high power consumption, as compared with that of FPGAs.

2 DESIGN REQUIREMENTS

Our product must be able to upscale a SD 240p widescreen video by a factor of 4.5. This is the scaling factor between SD and HD, and so our device should be able to perform this transformation; otherwise, we would not restore the video into the correct dimension. While upscaling alone is a fairly trivial task, we also must attain an average SSIM of greater than 0.6, as this provides a decent quality when we checked by eye, and also is on-par with what current literature on image upscaling for videos

is able to attain. We will determine the average SSIM of an upscaled video by taking the mean of all SSIM's for each video frame.

Our product must also convert in real time. This means that we must have low latency and high throughput. For throughput, we will be able to process at least thirty frames every second, in order to be able to keep up with standard 30FPS video. For latency, we should be able to complete processing with latent delay of less than 60ms. This latent delay was chosen due to the EBU Recommendation R37-200, a recommendation on "The relative timing of the sound and vision components of a television signal".

Our product must also provide a certain degree of ease of use. We want for users to be able to use the product with minimal preexisting technical knowledge or frustration.

3 ARCHITECTURE OVERVIEW

Our system architecture is split into two components: a software architecture, describing the derivation and creation of our model and its parameters; and a hardware architecture, which implements our software model using the derived parameters and displays the upscaled video.

3.1 Software Architecture

Our final software model is fundamentally based off a cascading of DSP & deep learning methods. The frames of the video will feed in one by one into a bicubic interpolation layer which will perform the necessary operations to upscale the image to an acceptable resolution. From here, the upscaled image will be forwarded to layers of a CNN which will be responsible for performing super-resolution on the output of the bicubic interpolation layer. The bicubic interpolation is done by approximating a smooth curve in between a cluster of points on the rbg valued function grid. This is deemed a non-learning method & is a formal DSP method to take a set of pixels and duplicate averaged pixels across the cluster. This will be implemented via the numpy & scipy libraries in Python. We will then benchmark the bicubic interpolation implementation and from here determine the optimal depth for the CNN to perform super-resolution on the resultant image. The CNN will work in the following manner: (1) a patch extraction block (2) a non-linear mapping block (3) and then output reconstruction from these higher resolution patches. The first layer convolves the input image with a 9x9 kernel with padding and expands the the three-channel image into 64 feature maps. The second layer applies a 1x1 kernel to condense to 32 feature maps. The third layer will apply a 5x5 kernel to generate the output image. Both the first and second convolution layers are forwarded through ReLU. Ultimately, we will run this through a series of tests involving images downscaled to 240p which will be run through our algorithm to get up to 1080p.

3.2 Hardware Architecture

Our final hardware architecture consists of data flowing from an attached USB through the FPGA via the ARM core host. The data makes one final hop back to the ARM core before being sent out the DisplayPort terminal. The DisplayPort terminal is then connected to a monitor or screen for viewing.

From the point of view of a user, they would plug in their USB containing low-resolution videos. Then they would select which video to play by using the button on the Ultra96 board. Once their selection is made, the host program, living on the ARM core, will begin reading in the video file and sending it to the FPGA. The FPGA will process the video frames by upscaling and correcting them. The output is then sent back to the host program, which is then finally sent out to the DisplayPort, attached to an external screen, for viewing. For testing and benchmarking our metrics, we will save the video output back to the USB as opposed to displaying it.

The main controller for this will be the host program, which we will write ourselves in Vitis HLS. The FPGA will handle accelerating the ML model which we specify in our software architecture. The parameters will be fixed so no training or variation will be required during the use of the hardware.

The content in 3.2 in gray is off the shelf. The content in orange is content which we create, but is heavily based off of preexisting content. The content in teal is content which we generate. The content in yellow is off the shelf but requiring our own installation, as opposed to a ready-to-go component. The content in blue is that which we make ourselves. Square boxes represent hardware, rounded square boxes represent software, and ovals represent data.

4 DESIGN TRADE STUDIES

4.1 Video Similarity Metric

As noted earlier, our training and evaluation metric will be SSIM. We were deciding between SSIM, 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 were too broad of error metrics which could give a higher error than what a person would actually perceive. Consider the case where every pixel is 'off' by some small delta. If each pixel is 'off' in the same direction, PSNR and MSE would find that this is a large error while VMAF and SSIM would give back 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. While we did first plan to use VMAF, upon testing an

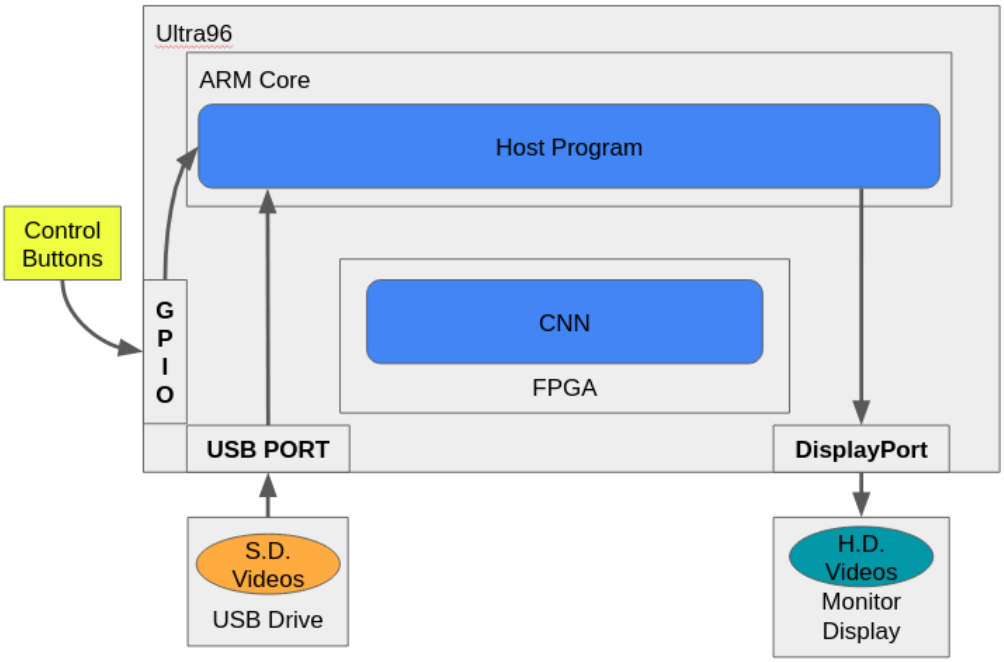

Figure 1: hardware block diagram

implementation of it, we realised that it ran very slow. The total time to evaluate VMAF on a 20s clip was on the order of a minute, thus disqualifying VMAF as a usable metric for training.

We finally settled on SSIM, as, similar to VMAF, it can be tuned to account 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 details how to modify SSIM to analyze video data. We will be using their same formulation for training and validating our system.

First 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:

$$SSIM_{ij} = \sum_{x \in \{Y,Cb,Cr\}} W_x SSIM_{ij}^{x} \tag{1}$$

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

$$Q_i = \frac{\sum_{j=1}^{R_s} w_{ij} SSIM_{ij}}{\sum_{j=1}^{R_s} w_{ij}} \tag{2}$$

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

$$Q = \frac{\sum_{i=1}^{F} \mathcal{W}_i Q_i}{\sum_{i=1}^{F}} \tag{3}$$

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.

The values of R_s, F and the $\mathcal{W}_i$'s may be tuned for specific applications. We, however, anticipate following similar parameters as described in the referenced paper in order to tune Q towards human viewing experience.

We recognise that this is a trade-off in that VMAF has been shown to be a much better metric for identifying the quality of video, but in return takes much longer to evaluate. Thus, we are limited by the capabilities of our compute to preform such an intensive checking function, and instead must use an arguably less accurate metric. Since SSIM is still good, albiet less human-accurate than VMAF, we now aim to have a score of over .6 in VMAF as this would place our device on a bit under the score for typical 2x video upscaling, on-par with 3x upscaling, and slightly ahead of 4x video upscaling. Given the context that SSIM usually decreases as the scaling factor increases, we are intentionally undershooting some of the higher SSIM indices achieved with low scaling factors, as we are scaling to a multiple of 4.5.

4.2 Ultra96 Development Board

We chose to base our project off of the Ultra96 development board. Since we anticipated the need for hardware acceleration, we knew that we could not implement our

project on a CPU, so the three main remaining options we considered were a (GP)GPU, an ASIC, or an FPGA. We immediately disqualified an ASIC as the field in which we are creating a product is rapidly evolving, and an ASIC would have a far too long product development cycle, as well as an inability to be modified or improved in a meaningful way after tape out. Additionally, within the scope of this class, the development of an ASIC, especially one of this scale, would not be an achievable deliverable. Between a GPU and an FPGA, the choice was closer; both options would be good and, to an extent, similarly applicable. We settled on an FPGA due to the reasoning that, if this project were to go into production, an FPGA would be superior to a GPU, in the wild. This is because for a rapidly evolving field, reconfigurability is a great asset, and even a necessity when the system architecture may demand a change beyond what GPUs are able to support. A fact that we also slightly take into account was that our group was more familiar with FPGAs than GPUs. It should also be noted that, while GPUs on average take more power than FPGAs, we did not consider this as a design factor, as we assume that the product will be plugged into the wall, meaning that so long as power consumption is not exceptionally high, it would not become a limiting factor.

Once we decided on using an FPGA, we considered a few options, mainly between the boards which we had used previously in classes, as well as the Ultra96 board. Since the Ultra96 has a vast amount of community support, native mini-DisplayPort output, USB inputs, as well as an ARM core that is capable of running petalinux, we chose this board as a natural fit to work with our project. The physical footprint of the board is also more compact than the boards used for other hardware classes during undergrad, making the form factor of the Ultra96 more conducive to our device's ease-of-use. The acquisition of the board was simplified by the fact that one of our team's members is currently in a course which uses the board, and so has access to one, thus meaning the board does not adversely affect our budget.

5 SYSTEM DESCRIPTION

5.1 Hardware System

As our hardware design is based around a single board with a great deal of flexibility, our project does not exhibit subsystems in the way that others might. As stated before, our hardware system is centred around the Ultra96 development board. This board has a ZU3EG-1 Zync device as its processor, as well as 154K programmable logic cells and 2GB LPDDR4 memory. The footprint is small as mentioned before – 3.35" x 2.1" x 1.3", on par with the footprint of other more recognisable embedded devices, such as the arduino or raspberry pi.

5.2 Software System

The first subsystem that will be fully implemented is the software model. This subsystem encompasses the bicubic interpolator & the layered CNN as described earlier. This subsystem will be ported onto the fpga and is the brain of the video upscaling algorithm.

6 TEST & VALIDATION

6.1 Video Similarity Metric

In selecting the metric by which we will measure the faithfulness of our image reconstruction, we ran a test on VMAF to benchmark how long it takes to run. In our tests, it took around 3 seconds per second of video to run. That is to say, the amount of time it took to evaluate a pair of videos was triple the amount of time that the video itself was. This revelation was not expected to us. Since we determined from this that this amount of computational load was too great for our use-case as we had intended to use VMAF as a test and loss function for training our CNN, we had to change our stated video similarity metric function from the initial proposal. Thus, we made a trade off and used a SSIM index for testing and loss instead. This was a trade off because SSIM, while nevertheless a good metric, is not as focused on nor detailed with the use-case of the human experience of video as VMAF is.

In verifying that our system is correct, we will check two stages. First, we will run the software model against our SSIM metric and will require that it has greater than 0.66 SSIM. We chose this baseline because, according to a team based in Facebook AI Research [2], running bicubic interpolation on a frame-by-frame basis will result in an average SSIM value of that amount for the resulting, upscaled video. After the software model is validated, we will do the same with the hardware model. Since nothing should change between the hardware and software models, the check should pass easier in hardware than in software, since all of the work to get sufficient quality will have already been done.

More specifically, our SSIM metric will take in the HD 'ground-truth' image and compare it against the super resolution reconstruction.

6.2 Testing for Software Model

For the software portion of our project, we will be running our tests on a GPU, specifically, on the NVIDIA V100 GPUs that are available on AWS for our use. We plan to use half of our video dataset for training, and half of our video dataset for testing and validation. To test the model, we will firstly take the high-resolution videos, downscale them with a free, off-the-shelf, downscaling tool online [4]

6.3 Results for Design Specification B

7 PROJECT MANAGEMENT

7.1 Schedule

We are planning on distributing the work across the weeks remaining in the course. We are first planning on implementing a rudimentary neural network on the Ultra96 board first which will confirm our I/O and FPGA implementations work correctly. On the hardware side, we implemented our I/O and will make sure the FPGA implementations work correctly is an essential part in streamlining the progress of this project. The software model work and the hardware related work can be done in parallel. Once we have a software model working & the hardware mechanisms working as well we can port the two and benchmark our implementation. The placing of the neural network on the fpga will be a crucial aspect in the deliverables surrounding this project. Once we've verified a rudimentary neural network can be successfully placed on the board we can move forward with writing the bicubic portion and then verify that workflow is functional and then complete our project through adequate benchmarks. Refer to the Gantt Chart attached at the end of the document.

7.2 Team Member Responsibilities

As in the Design Report, include primary and secondary responsibilities of each team member. Responsibilities may be overlapping. They may have changed since the Design report, in which case you should document who actually did the work and, perhaps, discuss why the change occurred.

Kunal & James are taking charge of the hardware implementation of the image upscaling algorithm. Joshua will be working on the software model along with the help from the other team members. We all will be part of the algoru

This subsection should be no more than half a column.

7.3 Budget

You can use an entire page for the table of parts and budget at the end of the report, or possibly inline as in Table 1. Include the description, model number, manufacturer, quantity and cost of each component purchased. ("Amazon" and "Digikey" are probably not the manufacturers of the parts you purchased.)

It should be clear exactly what parts are required to recreate your project, so include parts that were purchased with your budget, parts that were borrowed from the Capstone course, parts that were scrounged, etc. Mark what you bought but did not use as your project may have changed direction since the start of the semester. Mark what you did not plan for in your design report, but you realized you need to get to complete your project. This section may end up being one or two sentences referring to the page where the budget spreadsheet is located in your final report.

7.4 AWS Credit Usage

We chose to use AWS for the convenience and reliability of the service, and because it suited the development of our software model extremely well. 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. So far, we've only used around $50 credit to run our code, but we anticipate that we will use more in the future when we further develop our project. We give our thanks to Amazon for giving us AWS credits to aid in our project at Carnegie Mellon University.

7.5 Risk Management

This full column should describe how you handled your project risk from the standpoints of design, schedule, and resources (budget and personnel) and identify how you mitigated against the risk that cropped up through the semester (e.g., fallback designs, risk reduction measures).

Focus on the primary risk elements, and use about a full column so that there is no more than one page for the entire Project Management section 7.

8 ETHICAL ISSUES

Although we've changed our use case from security footage and video streams to old videos and recordings, it still remains a fact that our video super-resolution tool could be used for those aforementioned cases. In those particular scenarios, the accuracy and correctness of the super-resolution could play a key, ethical 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. In the case where our algorithm fails to even produce a recognizable nor useful image, this might not actually pose an ethical problem, but there is a chance that the upscaled video could be used as inaccurate evidence, leading to the identification of the wrong person or the wrong license plate, for example.

Situations such as these can either have very minor consequences or very dire ones, depending on what context the end user decides to use the upscaling tool in. It could just lead to someone's features looking more distorted than usual, or it could falsely implicate someone as a criminal. Although this might seem like an edge case with a low chance of happening, as our use case is different from this,

Table 1: Bill of materials

Description	Model #	Manufacturer	Quantity	Cost @	Total
Ultra96 Development Board	V2	Xilinx	1	$0[1]	$3.98
					$14.00

we must consider the accuracy and the recommended use of our technology, and add disclaimers if necessary to discourage people from making, potentially, false assumptions with our technology.

9 RELATED WORK

The paper [3] is relate to our use case in that it is from where we chose our video quality metric. We draw heavy inspiration from the model described in [1]. This paper is also similar in that it is performing super resolution with a CNN, but is different in that it is doing the computation on a CPU and is only doing single images at a time.

What other projects or products are similar to what you were finally able to put together?

You can use up to a full column in the two column format for this. Use no more than one page for both Related Work and Summary.

9.1 Future Work

It would be interesting to train a model based on VMAF. As stated earlier, the runtime of VMAF is too long to use as a loss function, however, hardware accelerating the VMAF metric could allow for it to be used as a loss function, and for a video upscaling algorithm to be trained more directly on how the human eye judges video. Due to the scope of this course and the project we chose this wouldn't be feasible, but it is definitely something of note that would be worth investigating further after the end of the course.

Glossary of Acronyms

- ASIC - Application-Specific Integrated Circuit
- AWS - Amazon Web Service
- CNN – Convolutional Neural Net
- CPU - Central Processing Unit
- DSP - Digital Signal Processing
- FPGA - Field Programmable Gate Array
- FPS - Frames Per Second
- (GP)GPU - (General Purpouse) Graphical Processing Unit
- HD - High Definition
- MSE - Mean Squared Error
- PSNR - Peak Signal to Noise Ratio
- SD – Standard Definition
- SR – Super Resolution
- SSIM - Structural Similarity
- VMAF - Video Multimethod Assessment Fusion

References

[1] Chao Dong et al. "Image Super-Resolution Using Deep Convolutional Networks". In: *arXiv* (July 2015).

[2] *VID4 - 4X upscaling benchmark (video super-resolution).* URL: `https://paperswithcode.com/sota/video-super-resolution-on-vid4-4x-upscaling?fbclid=IwAR0dpagVn7NalfIXJuQSc4Bx-iearHC7BkRW47qA6VkrYcAVoJLI1zi5NDw`.

[3] Zhou Wang, Ligang Lu, and Alan C. Bovik. "Video Quality Assessment Based on Structural Distortion Measurement". In: *SIGNAL PROCESSING: IMAGE COMMUNICATION* 19.2 (Feb. 2004), pp. 121–132.

Your references should be a very carefully crafted list, cited in the appropriate ways. Don't merely list a Wikipedia page or a bunch of GitHub URLs. Note that any code you used in your project does need to be cited.

You can insert blank pages after the references to add full page figures or tables for

- Architecture and system description figures – if so, make sure you refer to them in section 3 or 5 as appropriate.
- Milestone and Schedule chart – if so, make sure you refer to it from section 7.1.
- Budget and Parts list – if so make sure you refer to it from section 7.3.

You are allowed no more than 3 optional pages to ensure any large system diagrams, and your milestones and budget are readable at the end of your document.

	Project Proposal		Design Presentation				Interim Demo				Final Presentation
TASK TITLE	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14
	9/20	9/27	10/4	10/11	10/18	10/25	11/1	11/7	11/15	11/22	11/29
Hardware											
Acquire Ultra96	~~JSG~~										
Acquire Peripherals	~~JSG~~										
Research I/O	~~JSG~~	~~JSG~~	JSG + KB								
Implement I/O			KB + JSG								
Test I/O			JSG + KB								
Get Comms between ARM Core and FPGA		~~JSG~~									
Write Math Functions for CNN in Vivado HLS				JSG	JSG	JSG					
Validate HW				KB	KB	JL+KB	JL+KB				
Port SW model onto FPGA							JSG + KB				
Validating FPGA model against SV model							KB	ALL			
Software											
Research DSP vs CNN models	~~ALL~~										
Acquire AWS Credits		~~KB~~									
Setup AWS		~~KB~~	KB + JL								
Acquire Dataset	~~JL~~										
Familiarize VMAF Documentation	~~JL~~	~~JL~~									
Research specific CNN models		~~KB + JL~~	KB + JL								
Benchmark VMAF		~~JSG + JL~~									
Research SSIM	~~KB~~										
Benchmark SSIM		~~KB + JL~~									
Benchmark CNN Models		~~JL + JSG~~	JL								
Develop Python Code for Training			JL + KB	JL	JL						
Train Model				JL	JL						
Test/Evaluate Model				JL	JL						
Misc											
Slack/Slop									ALL	ALL	
Milestones											
Proposal Presentation	~~JSG~~										
Design Presentation			KB								
Design Review Report			ALL	ALL							
Interim Demo							ALL	ALL			
Final Presentation											JL

JSG
JL
KB

James
Joshua
Kunal

Figure 2: Gantt Chart