479 Lab 3 Report

Neil Rayu (nrayu2)
Neil Deo (neildeo2)
Ved Vyas (vedsv2)

Track 1 and 2

We are doing a combination of both tracks, so our project (as a group of three) has a wider breadth and focuses on both aspects of IOT and Cognitive computing.

Problem

Sign language prediction problem:

Sign language operates at a high rate of information transmission, with fluent signers reaching speeds of up to 150 words per minute. This presents a lot of difficulties for real-time recognition systems, which have to process visual input with minimal latency to remain responsive at a quick pace. Considering the cumulative delays introduced by frame capture, data preprocessing (which is an especially heavy hitter if high accuracy is needed), and transmission, the inference time of the model itself must be extremely low, ideally within the range

of tens of milliseconds. Achieving such performance is essential to ensure accurate and timely interpretation in practical, real-time settings.

Challenges we faced:

The biggest challenge we faced was inference the model on the FPGA. We wrote custom HLS to run a Convolutional Neural Network (CNN), and we had to create all of the different layers in hardware. A challenge we faced specifically with running the model on the FPGA was that the PyTorch conventions were not fully apparent, so when we implemented the model, some of the weights were not compatible with the way we did calculations. Further, we had issues with the way the convolution was being calculated. We implemented a convolution without padding, and that caused stalling in our stream input collection.

We also had challenges with finding a good tradeoff between model accuracy and parameter size. When we implemented a 3-layer CNN with 3 fully-connected layers[1], we had to employ quantization on the FPGA, which was challenging to do within each kernel. We decided to use float32 on the FPGA, but in order to budget the tradeoffs between “real-time” inference, and resource utilization, we decided to use a smaller CNN architecture with only 2 convolutional layers and 2 fully connected layers. This caused our model accuracy to decrease slightly.

The last component of the design was the dashboard with both front-end and back-end on the Raspberry Pi. Although writing the UI elements of it was a pretty standardized template the challenges mainly lied in trying to run cv2 optimizations to try and process the native camera frames in the backend (which was running at 1280x720) to the correct data format for sign MNIST which is 28x28. The difficulties of applying real world data to a machine learning model is known by most and we faced it especially with our model.

Design

Novelty in this field:

The novelty in our design is that we combined an Edge Device and an FPGA, in a configuration that used the FPGA as a neural network accelerator, and allowed the Edge Device to display an intuitive, pleasing dashboard with Text-to-Speech. We were able to run a pretty complex dashboard with a text-to-speech interface, and offload the majority of the computation to the FPGA. We also were able to unlock the power of ethernet for high-speed communication between the FPGA and the Raspberry PI.

Design specifications and how we faced challenges:

[FPGA design]

The HLS model implementation had issues with optimizations. The model itself worked fine with minimal usage, but the issues occurred with pipeline pragmas. Essentially, the pipeline pragmas caused major DSP/LUT usage spikes, which caused the model to be unsynthesizable. The issue with pipelining and unrolling is that the memory partitions required were too large for the total BRAM space on the FPGA, so we were not able to fully leverage the inherent parallelism of the FPGA. Since we were not able to rely on the Vivado built-in optimizations with certain pragmas, we had to manually optimize the design. This included repeated synthesis with different unrolling configurations to find the optimal resource usage. After testing, we found a stable and efficient design that optimized for latency and resource usage, while also accurately computing inferences.

[Model design]

The Convolutional Neural Network was trained on 27,455 28x28 images of ASL hand-signs[3][4]. It achieved a testing accuracy of 85% on a separate 7172 images. The first convolutional layer applied 8 3x3 convolutional filters, such that it took in inputs of shape (1,28,28), and outputted 8 feature maps, each of size (28,28), since we used valid padding. The first pooling layer downsampled the feature maps by a factor of 2 in both height and width, such that the output shape was (8,14,14). The second

convolutional layer applied 16 convolutional filters to the 8 pooled feature maps, such that the output was (16,14,14). The second pooling layer downsampled the feature maps by the same factor as the first pooling layer, so the output shape was (16,7,7). The first linear layer linearly transformed the outputs of the convolution to a 64-dimensional space. The second linear layer was the final classification layer for the 25 classes we used, each corresponding to a sign language alphabet.

[Edge/RPI design]

The main architecture of how the Raspberry Pi dashboard works is by running a backbone flask server that will handle every action that occurs on the HTML page through routes, so we have different POST and GET requests running when the user presses an option like capture image that will capture an image using the pycamera and cv2 libraries and send this image later to the FPGA. I also used a standalone function to prepare the data for MNIST prediction which incorporated these steps:

- **Centered ROI image cropping**. This entails drawing a box on the camera screen so the user can center there hand in the box, I also researched more intensive ROI measures for example MediaPipe Hands[2] which does keypoint detection on the hands but this doesn't fit our use case and general YOLO ROI automatic ROI Detection but this increased the latency too much for the benefit of FPGA commuting to apply.
- **Gray-scale conversion**. This was a pretty straightforward measure applied since the MNIST Model expects grayscale images.
- **Histogram Equalization**. This measure increases the visibility of features in our case digits of fingers on the hand by spreading out intensity values so there are not as many intense spot effects, which was a problem we were getting with the data when we tried techniques to purely enhance contrast.
- **Image Brightness Adjustment.** This was added due to after researching more on the training dataset it seemed MNIST data was modified or captured with high lighting with white hands on a black background.
- **Bilateral Filtering**. This filter was added to smooth and preserve edges of the image to be inference since we noticed that heavy downscaling would lead to very pixelated values with the edges slightly merged together.
- **Aspect Ratio Adjustment**. I added a quick aspect ratio adjustment to make cv2 capture the image from a square aspect ratio in order to make it easier to crop to sare 28x28 image.
- **Resize**. Scaled the image down to proper 28x28 for MNIST.

- **Normalization**. Last but not least I just normalized the gray scale values by dividing by 255 for inference.

Design Diagrams:

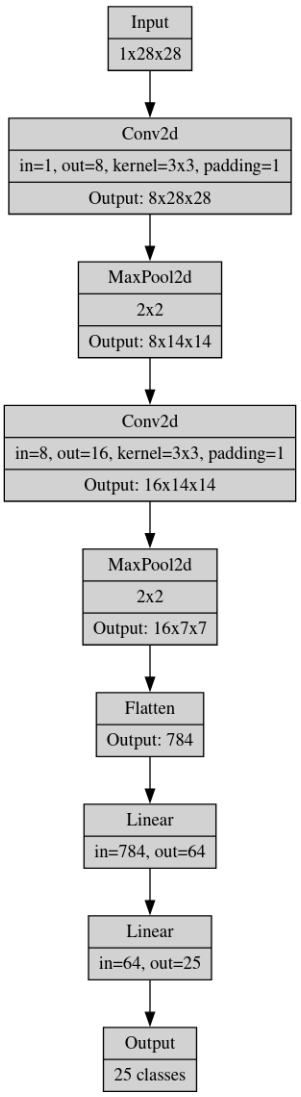

Figure 1: Model Architecture

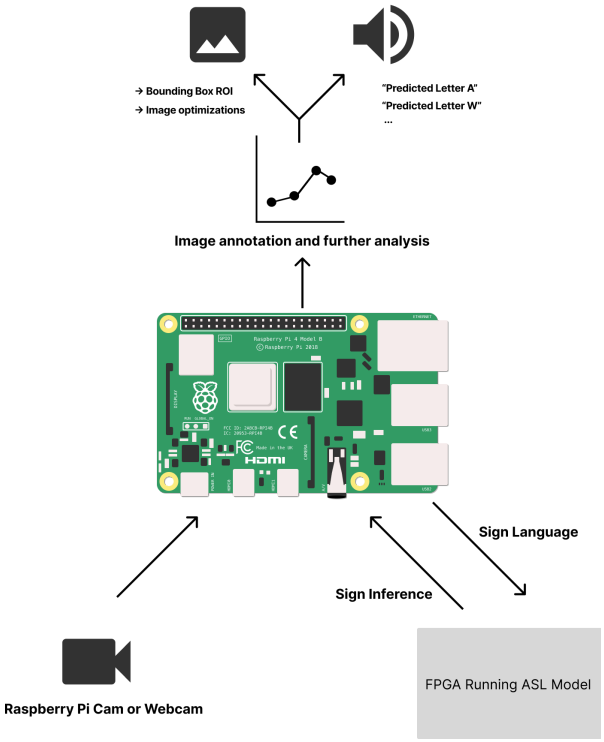

Figure 2: Full Design Layout

Results

Demo/ Image predictions:

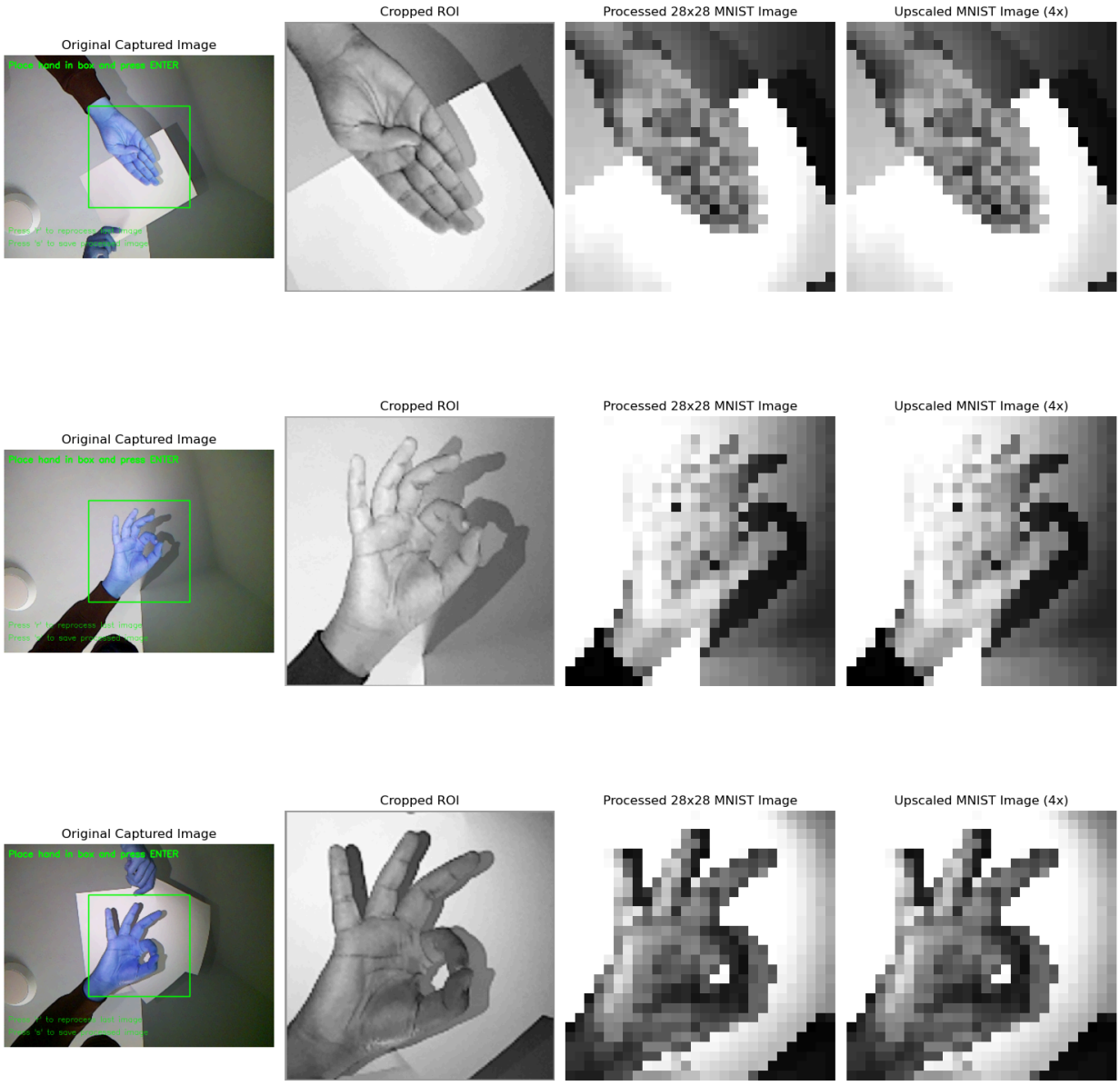

Figure 3: Preprocessing done on captured images before sending to the FPGA

```
finished configuration
connected to ved pc
packet recieved:b'\x02\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
2
time: 0.0014321290000225417
```

Figure 4: Raspi-to-FPGA communication Latency (Ethernet)

```
0.8505298382598996
281.5847547620001
```

Figure 5: Testing Accuracy on the FPGA (top) and total time to inference over 7172 images in seconds (bottom)

Design specific optimization results:

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	9.379 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
1232326	1232326	12.323 ms	12.323 ms	1220326	1220326	dataflow

Detail

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	62	-
FIFO	51	-	1258	2847	-
Instance	142	35	8123	14681	0
Memory	36	-	0	0	0
Multiplexer	-	-	-	108	-
Register	-	-	12	-	-
Total	229	35	9393	17698	0
Available	280	220	106400	53200	0
Utilization (%)	81	15	8	33	0

Figure 6: Full Model Utilization Report

These are the HLS design synthesis reports. As listed above, the latency for the model execution was significantly quicker than an inference on the RPi CPU. This

model was also accurate to the outputs of the actual model, which can be compared below with a sample input:

Actual:	Expected:
Logit 0: 4.06091	Logit 0: 4.0605
Logit 1: 7.05525	Logit 1: 7.0560
Logit 2: -15.3075	Logit 2: -15.3079
Logit 3: -7.98411	Logit 3: -7.9830
Logit 4: 17.9696	Logit 4: 17.9701
Logit 5: -6.84764	Logit 5: -6.8479
Logit 6: -13.211	Logit 6: -13.2120
Logit 7: -3.78429	Logit 7: -3.7849
Logit 8: 7.55354	Logit 8: 7.5541
Logit 9: -50.5838	Logit 9: -50.5836
Logit 10: -18.0171	Logit 10: -18.0168
Logit 11: -28.9231	Logit 11: -28.9237
Logit 12: 15.3201	Logit 12: 15.3208
Logit 13: 10.455	Logit 13: 10.4557
Logit 14: -3.22912	Logit 14: -3.2288
Logit 15: -9.08677	Logit 15: -9.0874
Logit 16: -7.20367	Logit 16: -7.2038
Logit 17: -18.4232	Logit 17: -18.4229
Logit 18: 13.2162	Logit 18: 13.2164
Logit 19: -7.29613	Logit 19: -7.2966
Logit 20: -6.62628	Logit 20: -6.6253
Logit 21: -5.99861	Logit 21: -5.9982
Logit 22: -4.47841	Logit 22: -4.4787
Logit 23: -5.85376	Logit 23: -5.8534
Logit 24: -11.9274	Logit 24: -11.9280
Actual Max: 4	Expected Max: 4

For our track 2 element, we were specifically trying to compare optimization techniques for the convolution computation, and compare the resource usage and latency between the two designs. Here are the synthesis reports, side by side:

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
57626	57626	0.692 ms	0.692 ms	57626	57626	none

Detail

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	13	-	-	-
Expression	-	0	0	685	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	1	-	0	0	0
Multiplexer	-	-	-	427	-
Register	-	-	577	-	-
Total	1	13	577	1112	0
Available	280	220	106400	53200	0
Utilization (%)	~0	5	~0	2	0

Figure 7: Depthwise-Sep. Conv.

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
93753	93753	0.938 ms	0.938 ms	93753	93753	none

Detail

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	3	-	-	-
Expression	-	0	0	841	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	2731	-
Register	-	-	241	-	-
Total	0	3	241	3572	0
Available	624	1728	460800	230400	96
Utilization (%)	0	~0	~0	1	0

Figure 8: Traditional Conv.

We can distinguish the differences between the approaches below:

1. Our model accuracy was higher with the traditional convolution compared to the depthwise-separable convolution.
2. Our resource utilization was lower with the traditional convolution compared to the depthwise-separable convolution, especially BRAM usage.
3. Our latency was lower on the depthwise-separable convolution compared to the traditional convolution.

For our implementation, we chose the traditional convolution. The issue with the depthwise-separable convolution is that even with a lower number of computations required, the hardware is harder to optimize, since there are less perfect loops. With the traditional convolution, our total model latency decreased by a factor of two, which makes the model significantly more responsive. Overall, with the FPGA acceleration of our model, we were able to achieve “real-time” speeds even with all of the communication latencies between FPGA and RPi.

References

[1] Rakhmatov, D. R., & Breccia, A. (2024, February). *Real-time sign language recognition with convolutional neural networks*. ResearchGate. https://doi.org/10.13140/RG.2.2.31797.24805

[2] Zhang, F., Bazarevsky, V., Vakunov, A., Tkachenka, A., Sung, G., Chang, C.-L., & Grundmann, M. (2020). *MediaPipe Hands: On-device real-time hand tracking*. arXiv. https://doi.org/10.48550/arXiv.2006.10214

[3] "Sign Language MNIST." *Kaggle*, uploaded by datamunge, https://www.kaggle.com/datasets/datamunge/sign-language-mnist.

[4] Alteaimi, Amal, and Mohamed Tahar Ben Othman. "Sign Language MNIST Dataset Samples." *ResearchGate*, https://www.researchgate.net/figure/Sign-language-MNIST-dataset-samples-26_fig7_358819753.