Edge TPU-Research Paper-CSI-3640

Ruben Parra Montenegro

May 2024

1 Introduction

Google has released an interesting product, the USB Accelerator by Coral which is brought to us by Google research. The interesting thing about this product is that it contains an EDGE TPU for "high-speed machine learning inferencing". I chose this topic because we can still delve into some of the architecture that makes these TPUs interesting while also exploring another topic that I Believe is interesting, neural networks. I will be discussing a few topics other than architecture just out of curiosity. My presentation will hopefully have some use cases of the USB Accelerator in it(I purchased one and want to see if I can mess around with machine vision). It is supported on major systems, including anything running Debian Linux, macOS, and Windows 10. I hope you enjoy the paper, I enjoyed learning about all of this and will continue to do so after this paper.

2 Neural Networks

So why is the topic of neural networks relevant to my research on the Edge TPU developed by Google (specifically the USB accelerator)? Well the Edge TPU is typically used to do a lot of math that is found in neural networks, they can do this because of their interesting architecture designed specifically for those kinds of calculations. TPUs have access to huge matrix multipliers, memory with high bandwidth, and have networks of processors that rhythmically compute data while passing it through the system which are called systolic arrays [9]. They also don't do any general purpose computing, this allows them to focus purely on math for neural networks to determine what something is, of infer.

I want to dive into this subject because it makes it easier to understand what a TPU is for and neural networks are fascinating. I will be covering a simple neural network that was made and explained by a youtuber named Samson Zhang[10]. I will have the link for this amazing video in the references. He goes through the math and programming side of a neural network and shows you step-by-step how to train a simple model with the ability to tell what number is written down, using a dataset of handwritten numbers from the MNIST dataset.

2.1 The Basics

The way neural networks work is by taking in inputs and doing a bunch of math and spitting out a confidence value while guessing what something is, in our case its numbers. In the MNIST dataset, the resolution of the images of the numbers is 28x28 pixels, in total that is 784 pixels. With each pixel having a value between 0 and 255. Since there are 784 pixels in total every pixel has to be used as an input for the neural network. So in the math sense, they are put into matrices such as this one where each row represents a data set or 1 image. But this is later transposed so each column can represent an image while each row represents its own pixel in that column. As seen below you can see a figure of this.

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,784} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,784} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,784} \end{bmatrix}$$

Figure 1: This figure shows each row as an image and each column representing individual pixels. [10]

$$\mathbf{X}^\top = \begin{bmatrix} x_{1,1} & x_{2,1} & \cdots & x_{n,1} \\ x_{1,2} & x_{2,2} & \cdots & x_{n,2} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1,784} & x_{2,784} & \cdots & x_{n,784} \end{bmatrix}$$

Figure 2: This figure shows each column as an image and each row representing individual pixels. [10]

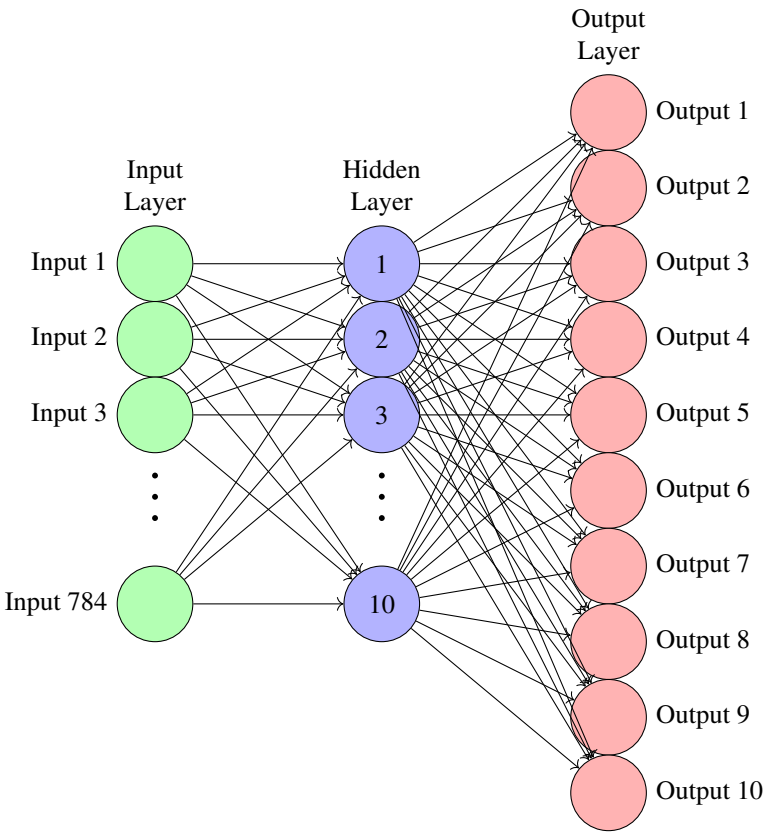

Figure 3: This is a simple representation of the neural network created using the video by Samson Zhang [10]

So with the images being 28x28 pixels, this results in 784 pixels, if each transposed column represents an image and each row represents a pixel with a value of 0 to 255, you can infer that the amount of inputs would be 784 as seen in figure 3. Each pixel RGB value is being input into those 784 inputs respectively. We call that first layer the Input layer, the last layer we call the output layer, and anything in between we call the hidden layer. In this model, we only have 1 hidden layer with 10 ”nodes”.

Now the goal with a neural network is that we can give it an image of a number and it will guess it correctly for example:

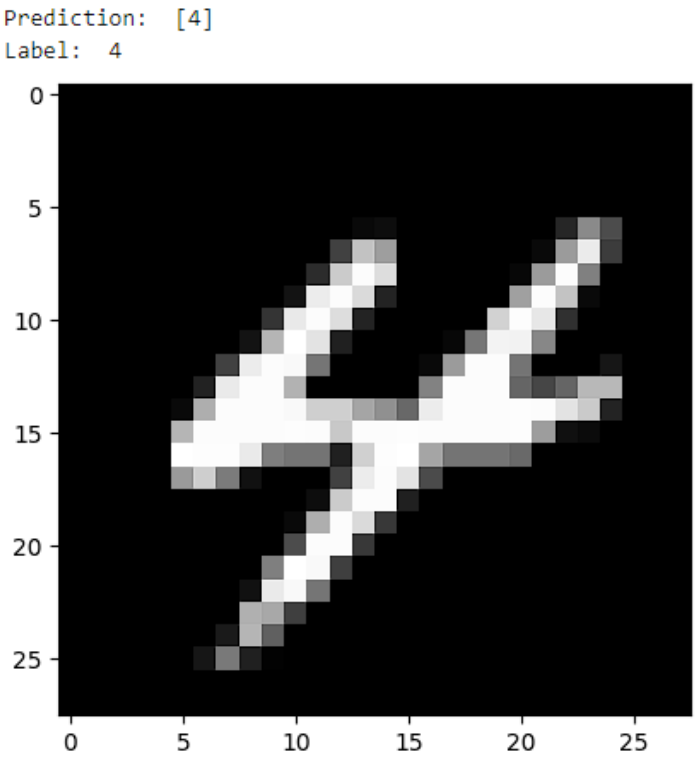

Figure 4: this is a result from a model I was able to train in Kaggle with the help of Samson Zhang, using his youtube video "Building a neural network FROM SCRATCH (no Tensorflow/Pytorch, just numpy & math)" [10]

To better understand how they work we have to delve into the math of neural networks and thankfully this is made easy by all the recourses out there, I used the Samson Zhangs video[10], "Building a neural network FROM SCRATCH (no Tensorflow/Pytorch, just numpy & math)". He goes through the math and the programming aspects while covering it all in a very intuitive manner. This paper is not on neural networks so I feel comfortable using this as a reference to show in general why TPU technology is applicable today.

2.2 Forward Propagation

Forward propagation is literally just running an image or whatever data you have through the network to get the result. So in this network, we have 784 inputs in the input layer. We have 10 nodes in the hidden layer, and we have 10 outputs. Samson Zhang mathematically represents these values as:

Input Layer:

$$A^{(0)} = X \quad (784 \times m)$$

Hidden Layer:

$$Z^{(1)} = W^{(1)} A^{(0)} + b^{(1)} \quad (10 \times 784) \quad (784 \times m) \quad (10 \times 1 \rightarrow 10 \times m)$$

Activation:

$$A^{(1)} = g(Z^{(1)}) = \text{ReLU}(Z^{(1)})$$

The representation of the input layer is pretty much X and the 784 is the amount of pixels per column(m). The hidden layer is represented by the function Z1 which is the unactivated hidden layer, which has a weight matrix and an input layer matrix that the dot product of them is taken and added to a bias term. After this, you need to apply an activation function to Z1 to add the ability for the model to learn more complex patterns by introducing non-linearity. For us we are going to use a ReLU(Rectified Linear Unit) function which can be represented as:

$$\text{ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

So that represents the activation function for Z1 which we go from A0 to Z1 and then we activate Z1. A0 is the input layer and Z1 is the hidden layer. Now we only have 3 layers in total, the last one is the output layer or Z2 do we also activate this layer in the same way? No, it is represented in the same way as Z1:

Hidden Layer:

$$Z^{(2)} = W^{(2)}A^{(1)} + b^{(2)} \quad (10 \times 10) \quad (10 \times m) \quad (10 \times 1 \to 10 \times m)$$

$$A^{(2)} = \text{softmax}(Z^{(3)})$$

Now how do we activate the output layer or Z2, we apply softmax because we want each of the output neurons to output a probability of what that number could be. The probability ranges from 0 to 1. The activation function looks like this:

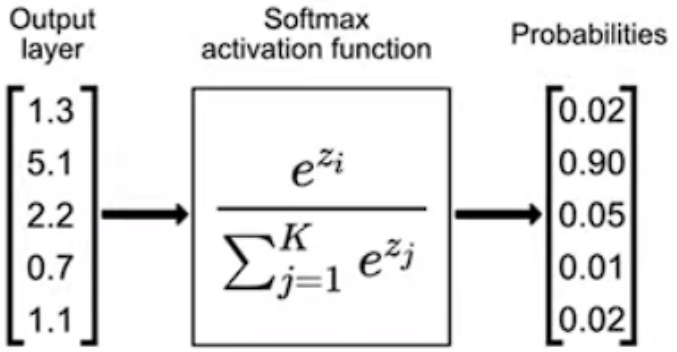

Figure 5: the output layer produces numbers that seemingly mean nothing but then are transformed into probabilities so we can better understand them.(Image from Samson Zhangs video) [10]

2.3 Back Propagation

So you might have noticed the term weight and bias passed around in the last section. These are values that can be adjusted to ensure the ability of the model to make good predictions. So what can we do to adjust these weights and biases automatically? Well through back propagation, which is essentially doing the same thing as forward propagation backwards. Now when training a neural network, you have to have data labeled so the network can use that during back propagation. So during this process, the labeled data will be used to compare the output probability matrix with a matrix that one image represents. For example in Figure 5 we see the probabilities as many different values in a matrix, the labeled data is compared to the output data and sort of thrown back through the network but just backward, in simple terms. So similar to Forward propagation we have more equations.

$$dZ^{[2]} = A^{[2]} - Y \qquad 10 \times m$$

$$dW^{[2]} = \frac{1}{m} dZ^{[2]} (A^{[1]})^T \qquad 10 \times 10 = \frac{1}{m}(10 \times m)(m \times 10)$$

$$db^{[2]} = \frac{1}{m} \sum dZ^{[2]} \qquad 10 \times 1 = \frac{1}{m} \sum (10 \times m)$$

$$dZ^{[1]} = {W^{[2]}}^T dZ^{[2]} \odot g'(Z^{[1]}) \qquad 10 \times m = (10 \times 10)(10 \times m) \odot (10 \times m)$$

$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T \qquad 10 \times 784 = \frac{1}{m}(10 \times m)(m \times 784)$$

$$db^{[1]} = \frac{1}{m} \sum dZ^{[1]} \qquad 10 \times 1 = \frac{1}{m} \sum (10 \times m)$$

So these equations are used for gradient descent which is for optimizing the weights and biases of the network. I won't elaborate further because neural networks aren't necessarily in the scope of this project but I leave the equations here to show in general the computations being done when models are running to give a better understanding of what is going on and for me to also learn something about neural networks while writing a paper on TPUs.

2.4 Updating Parameters

So when the backpropagation is complete we now update the weights and biases of the network for further learning using the equations below. The alpha symbol is for the learning rate which we can define.

$$
\begin{aligned}
W^{[1]} &:= W^{[1]} - \alpha dW^{[1]} \\
b^{[1]} &:= b^{[1]} - \alpha db^{[1]} \\
W^{[2]} &:= W^{[2]} - \alpha dW^{[2]} \\
b^{[2]} &:= b^{[2]} - \alpha db^{[2]}
\end{aligned}
$$

So once the weights and biases are updated we then iterate through the cycle of forward propagation, backward propagation and updating the parameters however many times you specify which can be 500 times, or millions of times or even more than that.

3 TPU

Now why did I go through the hassle of learning all that stuff to throw on this paper? I was interested in doing so and also I feel it makes understanding what a TPU does actually much easier. TPUs can be used to both train and run these models, the Google Coral Edge USB Accelerator is not used to train models but it will run them. Now what allows TPUs to be so good at these computations?

So to begin, TPUs or Tensor processing units are custom ASICs designed by Google, with the main purpose of them being to accelerate machine learning workloads. They are designed to compute large matrix multiplications, this is why I chose to explain the neural networks before this. As neural networks get bigger they will require exponentially more compute power and most of those computations are matrix multiplications, which is where the TPU comes into play. They can also be clustered, google calls it a pod and they say it can be done with ”little to no code changes”.

Now considering Google invented the TPU, most of the reliable information comes from them. I’ve gotten most of my understanding and information from Google on the topic of TPUs. When talking about TPUs, you want something to compare it to so you can really see what is so special about it. Now CPUs and GPUs are similar in regards that they both use ALUs, CPUs have a few of them, and GPUs have thousands of them. When they do any calculations, they access registers or shared memory to read the operands and they also have to store intermediate results which is the norm.

One interesting thing about the TPU is that it is really good at doing matrix operations for neural networks but it can’t really do much of anything else, it is application-specific. So their main task is matrix processing which is a combination of multiply and accumulate operations. Google states that TPUs contain thousands of multiply-accumulators that are directly connected to each other to form a ”large physical matrix”. Which is coined as systolic array architecture. Their cloud TPUV3 contains 2 systolic arrays of 128x128 ALUs on a single processor. [9]

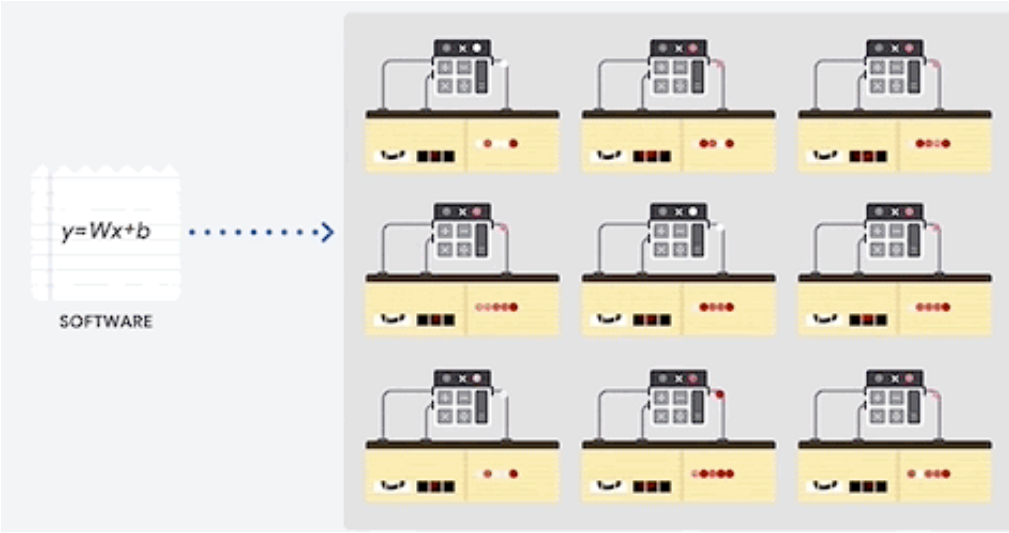

Figure 6: Conceptual representation on how GPUs/CPUs computations work.(Image from Google Cloud) [9]

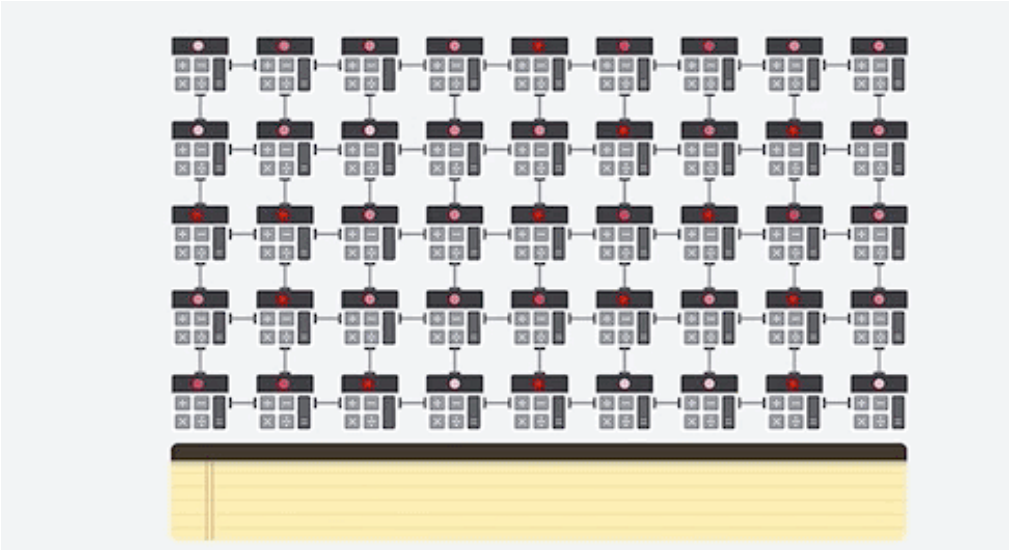

Figure 7: Conceptual representation on how the MXU of a TPU works, so how they do computations.(Image from Google Cloud) [9]

Going back to CPUs and GPUS they have intermediate steps in their arithmetic, TPUs do not. What I mean by that is they don't need to access memory during their computations, due to their interesting architecture. The way the matrix operations are performed is interesting. First, the TPU loads the parameters from its High Bandwidth Memory into its matrix multiplication Unit(MXU). As each multiplication is executed, this result is passed down to the next multiply-accumulator. The output is a summation of the multiplication results between the data and the parameters according to Google. Memory is not accessed during the matrix multiplication process. This results in high computational throughput on neural network calculations. [9]

Now the code that runs on a TPU has to be compiled on the accelerator linear algebra (XLA) compiler. According to Google, the XLA compiler is a just-in-time compiler, it takes the graph emitted by the machine learning framework application and compiles this into machine code for the TPU. The TPU machine code includes the linear algebra, loss, and gradient components of the graph. [9]

So essentially a TPU is made for running machine vision/ machine learning models, which I will attempt and hopefully include in the presentation and here to demonstrate what we can do with a USB Accelerator from Google's Coral Edge TPU. With the efficiency of computing matrices, we can even run models to do real-time inferring with a Raspberry Pi, a camera, and the USB Accelerator. Essentially making a Raspberry Pi into a supercomputer at this point.

4 ASIC

So the Edge TPU which is found in the Google Coral USB Accelerator is an ASIC, which means it is an application-specific integrated circuit. These kinds of circuits as the name suggests are made to do certain tasks, and do them really well and efficiently. Modern ASIC's can have somewhere around 100 million logic gates in them currently, which is a little wild to think of when we are learning about individual gates and to think we can put that many of them on a chip and just make it work is beyond me. Now if you have ever heard of the term SOC or system on chip this is referring to an ASIC that has these: ROM, RAM, EEPROM, Flash memory. Which they also can include microprocessors in them, so that is where you can get the term SOC(System on Chip). [15]

5 Systolic Arrays

"A systolic array is a pipeline network or "wavefront" of data processing elements. Each of these elements does not need a program counter since execution is triggered by the arrival of data." [16] Systolic arrays are rather interesting even regarding how they operate, being triggered by the arrival of data introduces interesting challenges when trying to use them properly. From what I was able to understand from a lecture I watched by Professor Onur Mutlu at ETH Zürich[17], the software has to orchestrate this data flow to the systolic arrays. Providing this proper timing is crucial due to the way systolic arrays function, which in a TPU case would be a physical matrix. But a good example was presented by Professor Onur Mutlu in his lecture.

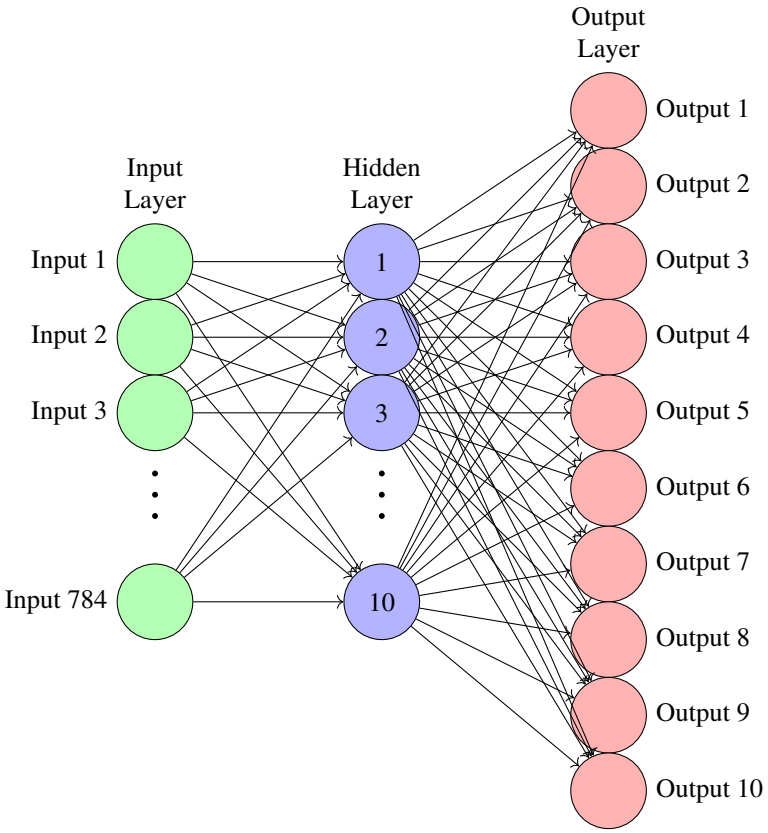

Figure 8: This is a repeated figure but it is helpful at this point in the reading[10]

But before we dive into "orchestration", I want to explain why the math is being done and what is being done. During forward propagation, data is input into the input layer. We will call the inputs X, there are 784 X values per each set of data or images. If you have 2 images you now have a 2x784 matrix of data being put into this input layer one after the other. We transpose this before it is input but that doesn't really matter. But during forward propagation these matrices of data are being put through an operation inside the neural network which in the case of the one talked about in this paper, our hidden layer is only 10 nodes. So if you look at the connections and recognize a pattern in how they are connected, you would have to do 10 operations per input leading to 7840 operations.

Hidden Layer:

$$Z^{(1)} = W^{(1)} A^{(0)} + b^{(1)} \quad (10 \times 784) \quad (784 \times m) \quad (10 \times 1 \rightarrow 10 \times m)$$

Which would be multiplying our weight value by our input and adding a bias 7840 times, this is a small model. After this, we apply our softmax function to arrive at a confidence level and then we backpropagate. So it gets computationally intense quickly.

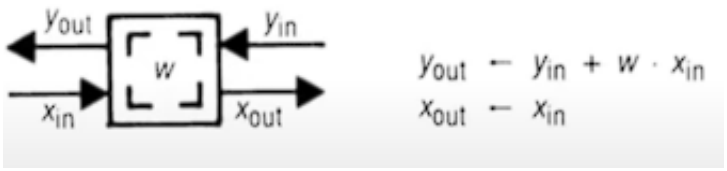

Figure 9: Example of a computation in a systolic array[17]

This figure from Professor Onur Multu demonstrates the math for forward propagation in a general example. Going back to how the data has to be orchestrated you can also see that here. The Xin being our input value, Yin being the Bias, and W being our weight. Looking back at our formula for the hidden layer we just spoke about, we can see when these values go this little "unit" the Yout is equal to the Z formula for the hidden layer. Now this is only one "unit". What do more look like?

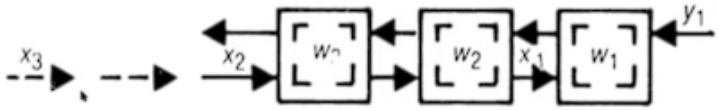

Figure 10: Example of a computation in a systolic array[17]

You can see this slightly more elaborate version of the one before and notice how there is a space between x3, x2, and x1. If you try to go through the diagram you will see the value for Y1(Bias) would hit each x value because of the space put between them. This is an example of the orchestration mentioned by Professor Onur Multu.

Systolic arrays are crucial to employ when in need of high throughput computations specifically for neural networks in the case of the TPU.

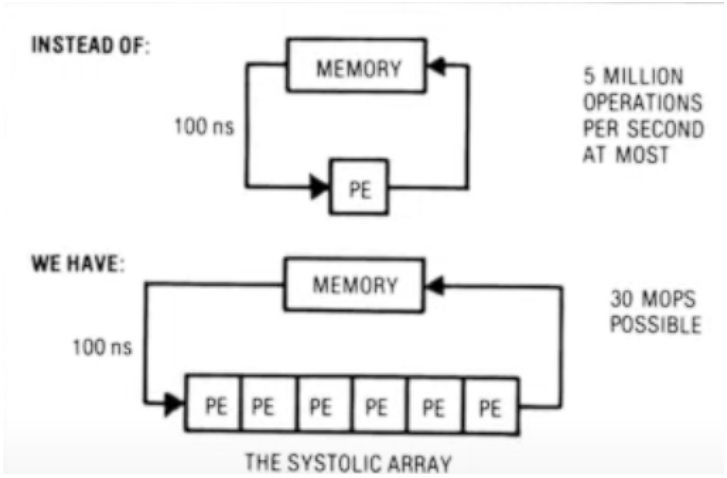

Figure 11: Example of a systolic array versus the typical Von Neumann Architecture of a CPU.[17]

Here is just another image of a bigger representation of one for reference in how they operate. You can see why you would need to orchestrate all of this data, it needs to go through all of the processing elements while also giving us what we want.

6 USB Accelerator

Figure 12: The Coral USB Accelerator [6]

After all this talk about these weird topics, we have finally arrived at what inspired me to learn a bit more then was necessary. I had to make the paper interesting for myself of course. So what is the Coral USB Accelerator? According to Google, it performs high-speed inferencing for Machine Learning models. It can perform 4 trillion operations per second. This can be attributed to those MXUs mentioned. We are able to use this little device on major platforms such as Debian Linux, macOS, and Windows 10. I chose to use a Raspberry Pi 5 which I ended up buying along with the USB Accelerator because I thought it was cool and it just worked out that I could write my paper on this. The outcomes are interesting, I did some testing with these devices with a few models to see how they held up and what they could pick up, it was exciting. Something I haven't gotten to mess with before.

6.1 Setup

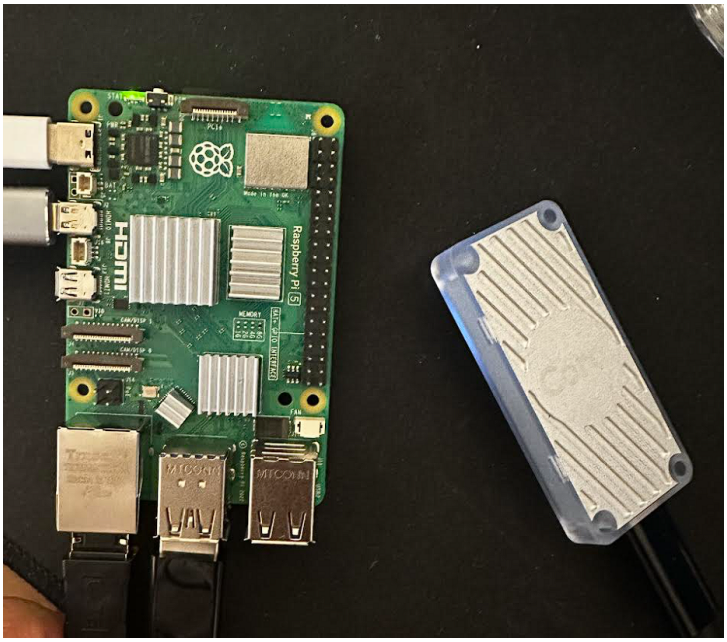

Figure 13: Raspberry Pi5 and Coral USB Accelerator

So I purchased the Raspberry Pi5 and a USB Accelerator because I thought what you can do with these items is interesting. When using these 2 devices, you want to employ active cooling because they will get hot when running a model to detect things. To get the setup to make inferences from video I ended up using a great video I watched called "Realtime Speed (FPS) for YOLOv8 and YOLOv9 on Raspberry Pi 5/4: Google Coral Edge TPU — Ultralytics", he also had a GitHub repository with everything I needed to start using TensorFlow Lite models to detect things from videos that I chose. I will leave a link to the specific repository and a YouTube link for it in the references. It was a simple setup without a hassle at all, just following what he said in the video got me to where I needed to be to test the USB Accelerator with the Raspberry Pi5. Unfortunately, I wasn't able to get the Raspberry Pi5 to do this by itself so I could compare them but it was still interesting to see regardless, exploring this interesting technology.

6.2 Use Cases

Now who are these devices marketed for? According to Google, they are used for Object tracking, image classification, object detection, pose estimation, predictive maintenance, machine vision, and much more. In reality that all sounds like it's the same thing to me, besides predictive maintenance. One of the more interesting uses for this is predictive maintenance, I went to Chicago to attend the Automate Show for my company where I work as a Process Manufacturing Intern for Nylok. We are in the business of coating bolts in sticky stuff. So we have a lot of equipment that can fail, when I spoke to someone at Siemens, they spoke to me about their predictive maintenance product. To explain it simply, we hook up all of our equipment to something to measure the current usage. Using this their AI software can learn how our machinery operates based on the current being used and how much is used at certain times. So it can see when for some reason you might have a high current draw and tell us this piece of equipment might fail, and other interesting things I don't really remember, the representative was quiet. This is possible with this Coral USB device. The part that I like about the Coral USB Accelerator which can be applied to industry and our specific process is the Machine Vision that it can enable us to use. We already use cameras with similar capabilities, but it is interesting that I have access to such similar technology for a fraction of the cost, I was considering training a simple model to detect good and bad parts for my own amusement and learning because after working on equipment such as that with all the sensors and such. It is exciting to see what is possible. So prototyping for industrial use Machine Vision is certainly possible with this and something I might end up doing as a side project.

6.3 Comparisons

Model architecture	Desktop CPU [1]	Desktop CPU [1] + USB Accelerator (USB 3.0) *with Edge TPU*	Embedded CPU [2]	Dev Board [3] *with Edge TPU*
Unet Mv2 (128x128)	27.7	3.3	190.7	5.7
DeepLab V3 (513x513)	394	52	1139	241
DenseNet (224x224)	380	20	1032	25
Inception v1 (224x224)	90	3.4	392	4.1
Inception v4 (299x299)	700	85	3157	102
Inception-ResNet V2 (299x299)	753	57	2852	69
MobileNet v1 (224x224)	53	2.4	164	2.4
MobileNet v2 (224x224)	51	2.6	122	2.6
MobileNet v1 SSD (224x224)	109	6.5	353	11
MobileNet v2 SSD (224x224)	106	7.2	282	14

Figure 14: This is a table from Google Corals website with comparisons made with their Edge TPU product. [7]

So I pulled this table from Google Corals website and they show how a device on its own and with the Edge TPU performs, as you can see in the table the time per inference(in milliseconds) drops drastically after using an Edge TPU.

The table above is fairly consistent with the behavior I have seen from looking around, as seen below.

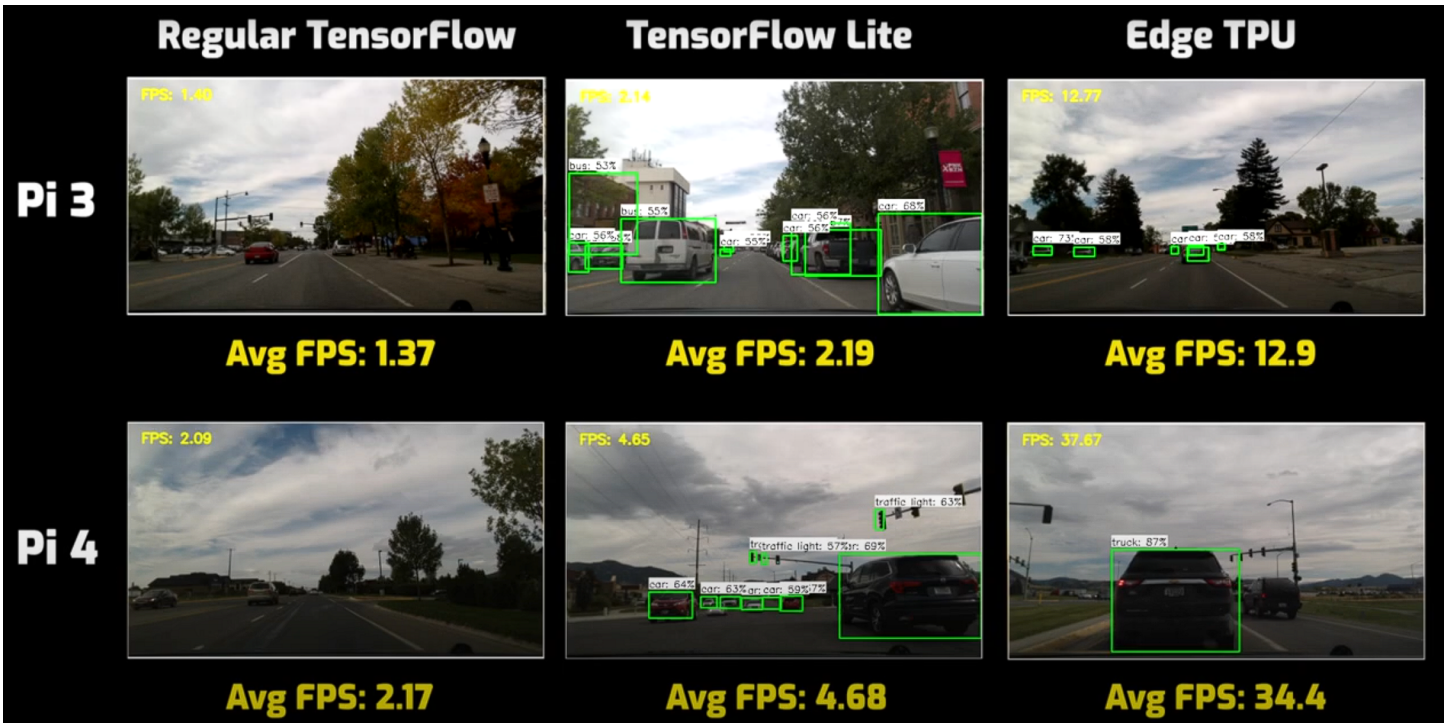

Figure 15: This is from a video comparing various ways of running a model and its performance from "Raspberry Pi 3 vs Raspberry Pi 4 Performance with TensorFlow, TF Lite, & Coral USB Accelerator" [13]

From the screenshot from "Raspberry Pi 3 vs Raspberry Pi 4 Performance with TensorFlow, TF Lite, & Coral USB Accelerator" [13] we can notice similar trends to that of the table presented by Coral. The performance of the models gets gradually better as we change from TensorFlow to TensorFlow Lite and then add the USB Accelerator to it(Which contains an Edge TPU). It goes to show that just adding a small TPU to a little board such as Pi4 in that video can make a complete difference in the capability of the inference. Truly shows how powerful Google's Edge TPUs are.

7 My Results/Conclusion

So what do I have to show, So using a Raspberry Pi5 and the USB Accelerator that contains and Edge TPU, I was able to run a model called `240_yolov8n_full_integer_quant_edgetpu.tflite`, using TensorFlow Lite. Now the public doesn't have access to all the information about the Edge TPU. But it is similar to what I talked about in this paper. An ASIC containing physical matrices of ALUs to do matrix operations that are typically found in neural networks can be used to accelerate devices such as a Raspberry Pi5. Now continuing the discussion of the physical matrix, the TPU does not have to access memory while completing a sequence of operations given to it thanks to its unique architecture which can also be described as a systolic array. It allows the TPU to not have to access memory while in an operation, unlike a CPU or GPU which are based on the Von Neuman architecture.

So after following "Realtime Speed (FPS) for YOLOv8 and YOLOv9 on Raspberry Pi 5/4: Google Coral Edge TPU — Ultralytics", I was able to run a model on my Pi 5 as said earlier, here is what I saw that was interesting. I ran the model on a few videos I had in my camera roll.

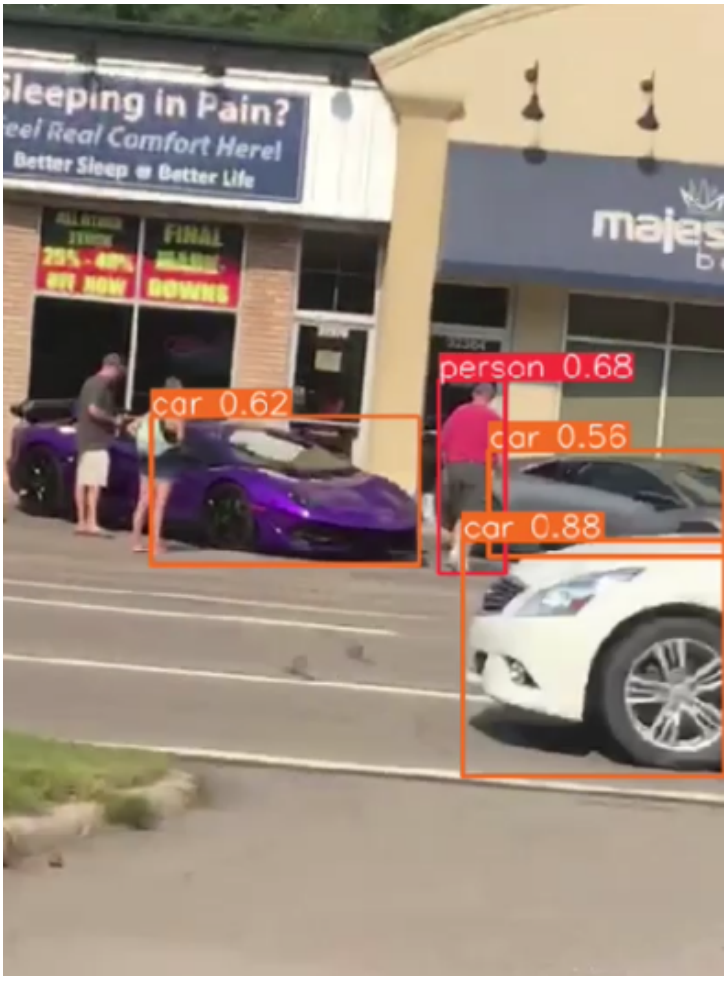

Figure 16: Model detecting a person and 3 cars in a video I took during the Dream Cruise on Woodward [14]

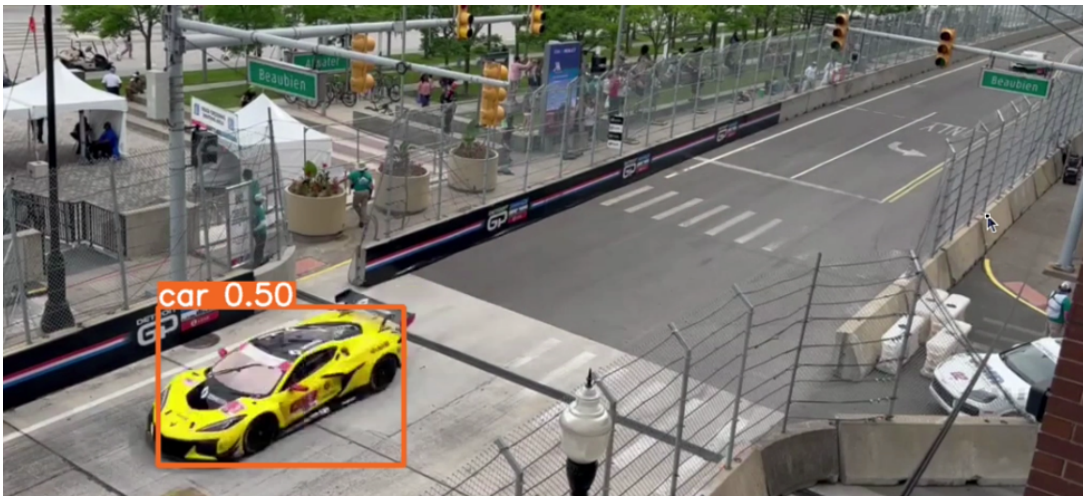

Figure 17: Model detecting a car during the Detroit Grand Prix last week [14]

Figure 18: Model detecting chair in my backyard [14]

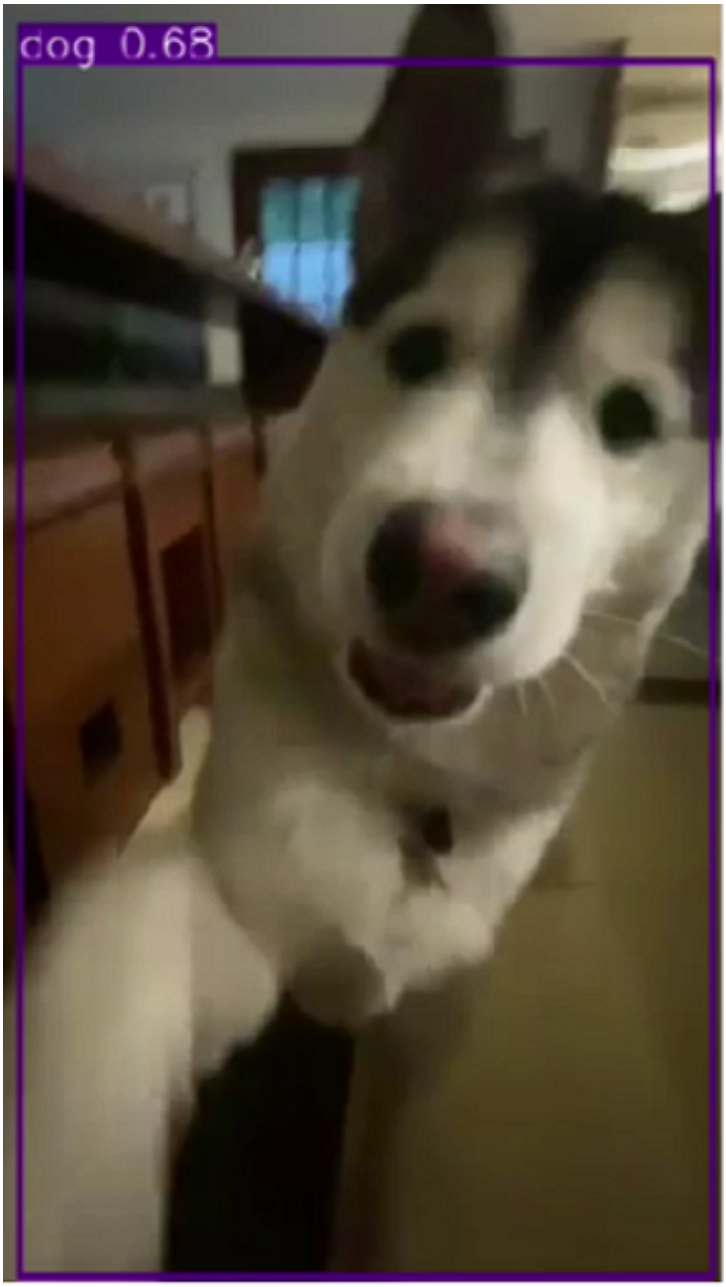

Figure 19: Model detecting my cousin's dog [14]

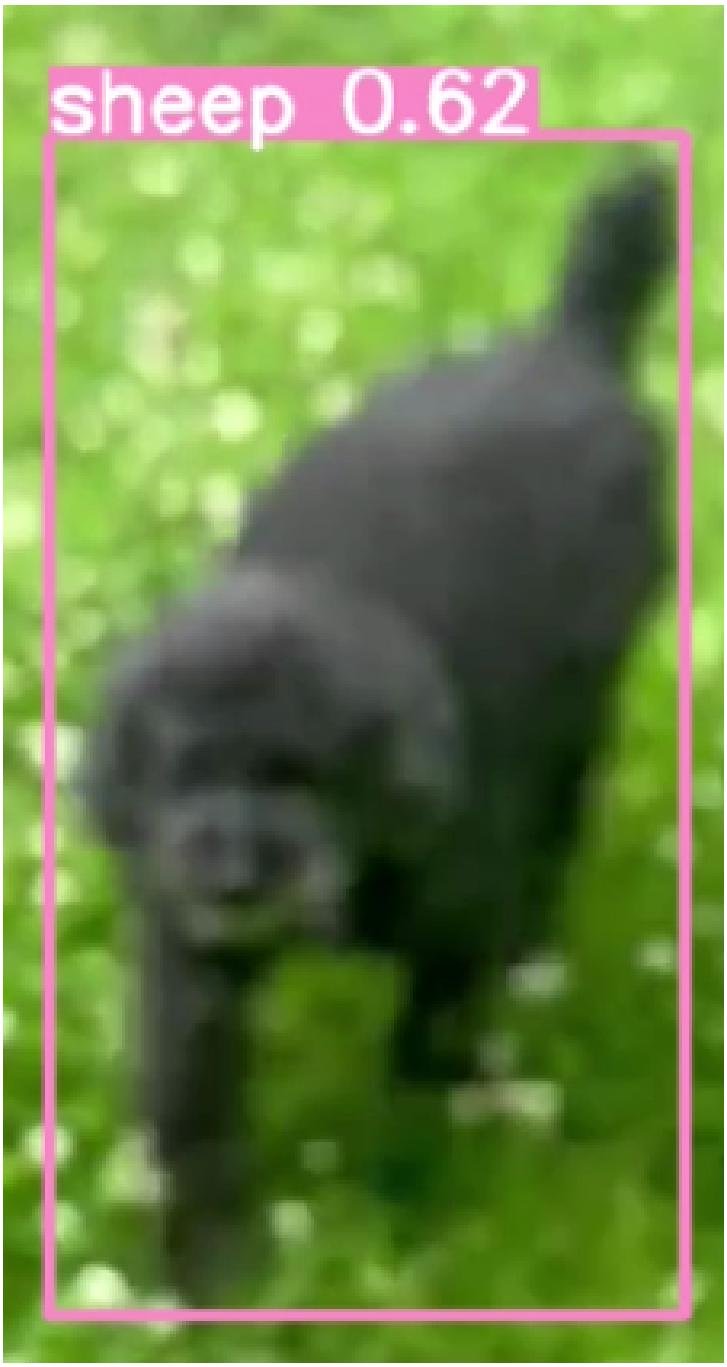

Figure 20: Model thinking my dog is a sheep [14]

Figure 21: Model thinking my dog is a bird [14]

So here we can see above some of the `240_yolov8n_full_integer_quant_edgetpu.tflite` model made. Some wrong but most were correct. To be able to run these models on something like a Raspberry Pi5 with a little thing attached to it is mindblowing to me how far this technology has come and how the architecture of a processor could make all the difference in providing something that is energy efficient enough to run on a little device while also being able to do the necessary computations to produce something worth using. Devices specialized for matrix computations are certainly an interesting topic to research.

So A few topics besides computer organizations were discussed in this paper, I felt that discussing the topic of neural networks and how they work to a certain degree could make it easier to understand why something like a TPU or USB Accelerator with an Edge TPU would be worth buying or learning about.

I also recorded a video of me looking at the videos with the model, I will leave the video link when I turn it in and in the references.

I enjoyed writing this paper. Thank you.

8 References

[1]“Introduction to Cloud TPU,” Google Cloud. https://cloud.google.com/tpu/docs/intro-to-tpu#: :text=intermediate%20calculation%20results.- (accessed Jun. 11, 2024).

[2]“System Architecture — Cloud TPU,” Google Cloud. https://cloud.google.com/tpu/docs/system-architecture-tpu-vm

[3]“Systolic array,” Wikipedia, Jun. 22, 2021. https://en.wikipedia.org/wiki/Systolic_array

[4]“Parallel processing - systolic arrays,” GeeksforGeeks, Aug. 03, 2018. https://www.geeksforgeeks.org/parallel-processing-systolic-arrays/

[5]“Examples,” Coral. https://coral.ai/examples/

[6]“USB Accelerator — Coral,” Coral, 2020. https://coral.ai/products/accelerator

[7]“Edge TPU performance benchmarks,” Coral. https://coral.ai/docs/edgetpu/benchmarks/

[8]“Get started with the USB Accelerator,” Coral. https://coral.ai/docs/accelerator/get-started/#requirements

[9]“Introduction to Cloud TPU,” Google Cloud. https://cloud.google.com/tpu/docs/intro-to-tpu

[10]Samson Zhang, “Building a neural network FROM SCRATCH (no Tensorflow/Pytorch, just numpy & math),” YouTube, Nov. 24, 2020. https://youtu.be/w8yWXqWQYmU?si=CJxmDkDag5p62fyz (accessed Jun. 11, 2024).

[11]D. NYARKO, “DAVIDNYARKO123/edge-tpu-silva,” GitHub, Jun. 07, 2024. https://github.com/DAVIDNYARKO123/edge-tpu-silva (accessed Jun. 11, 2024).

[12]Koby_n_Code, “Realtime Speed (FPS) for YOLOv8 and YOLOv9 on Raspberry Pi 5/4: Google Coral Edge TPU — Ultralytics,” YouTube, Apr. 10, 2024. https://www.youtube.com/watch?v=sOxQTRRh9tw&t=827s&ab_channel=Koby_n_Code (accessed Jun. 11, 2024).

[13]Edje Electronics, “Raspberry Pi 3 vs Raspberry Pi 4 Performance with TensorFlow, TF Lite, & Coral USB Accelerator,” YouTube, Oct. 16, 2019. https://www.youtube.com/watch?v=TiOKvOrYNII&ab_channel=EdjeElectronics (accessed Jun. 11, 2024).

[14]Ruben Parra, “Experimentation with Raspberry Pi 5 with Coral USB Accelerator,” YouTube, Jun. 09, 2024.

https://www.youtube.com/watch?v=hdtZ9aLY7w0&t=2s&ab_channel=RubenParra (accessed Jun. 11, 2024).

[15]“Application-specific integrated circuit,” Wikipedia, Jul. 05, 2021. https://en.wikipedia.org/wiki/Application-specific_integrated_circuit

[16]D. A. Patterson and J. L. Hennessy, Computer Organization And Design The Hardware/Software Interface. Cambridge, Ma Morgan Kaufman Publishers, 2018.

[17]Onur Mutlu Lectures, “Computer Architecture - Lecture 28: VLIW Systolic Array Architectures (Fall 2022),” YouTube, Jan. 10, 2023. https://www.youtube.com/watch?v=j9hOJZFt6rEab_channel=OnurMutluLectures (accessed Jun. 12, 2024).