

Convolutional Neural Network Applied to an Image with Performance Measurements

by

Andres Colon

Jimmy Ngo

Johnny Rivera

Mohammad E. Aly

Tyler Solarz

Contact:

johnnyrivera@cpp.edu

jimmyngo@cpp.edu

colon@cpp.edu

tjsolarz@cpp.edu

Abstract:

In this paper, we will discuss the project that our group has worked on, which is a Convolution Neural Network applied to an image. The purpose of this project is to learn how to code a program where it will take in any image and decide what the image is a picture of, for example recognizing a cat in a picture.

The created programs are run in Jupyter Notebook, which is a virtual environment, on a PYNQ-Z1 board as well as a standard desktop and laptop CPU. Next, we then determine the performance of the code that is run on Jupyter Notebook. Lastly, we then perform a comparison on the tested hardware to see which is a better candidate for this application.

I. Introduction

Convolution neural networks are a subset of machine learning known as deep learning, where a piece of computer hardware can perceive and recognize what humans see. The convolution neural network obtains some kind of data stream, in this case an image, and it begins to scan the data with the use of filters in order to recognize whatever is outlined in the code. Filters are what determine the features and special properties for sorting out different classes.

Though convolutional neural networks are known for its machine learning and understanding of concepts, it has a problem with its training. The training part of a CNN consumes a large amount of time based on how accurate the model is intended to be. There are companies that allow training on their servers, but even that is still slow in large scale operations. Upon the first try, a CNN will not recognize anything, or at least it'll be very inaccurate, till it sees the same features in different images multiple times and corrects for the previous error. We hope we can increase the chances of it being more accurate than before, or at least try to.

This paper presents the idea of measuring execution time, CPI, clock rate, and power consumption when running a CNN between different images.

Reasons that we tried to measure all those metrics is because:

1. Execution time: This is important to find because we need to know how fast a program runs. From that result we can try to improve the code to be more efficient and get results in a faster time.
2. CPI: Cycles per instruction is also something that can be used for this project. This is necessary to see how fast the data can be transmitted from the CPU. By knowing that, we can see if there is an increase or a decrease in our results when the code is altered, essentially determining if it can be improved, if at all.
3. The clock rate is another component that is needed. Clock rate is how fast the CPU can output cycles per second, which is in Hertz. Clock rate is often referred to as clock speed, but they are the same thing, the higher the clock rate the CPU can handle, the faster the clock speed. That means the CPU will be able to process the instructions/data at a quicker pace.
4. Power consumption is a key factor that makes or breaks a system. Convolution neural networks can take up a lot of power to run because of countless learning sessions to train it as well as run the program. When it comes

to implementing a trained network the amount of power consumed is then based on the main hardware. For example, running on a Raspberry Pi or PYNQ-Z1 consumes significantly less power than a standard CPU, and can sometimes be even more efficient.

II. Related Works

From the website, “www.towardsdatascience.com”, shows us a similar way to design a convolution neural network with an image like we are right now. This model shows step by step how to prepare the convolution neural network to classify these images, then how the machine takes the data and determine what aspects/details of the image are needed, and then accurately depict what it is. That is the general idea behind this CNN, but they create a module where the images that the CNN has to classify, are formatted into a chart to show the accuracy of its own depiction. Then it creates a matrix where it records the percentage of accuracy of the image classification [1].

The next related work that we looked at is a group where they did their CNN using something called a Simnet. They used this Simnet, which is a network that is trained on pairs of positive and negative images using a novel online pair mining strategy inspired by Curriculum learning [2]. This way provides a more intricate and complex way of doing a CNN with an image, but the results from it helps the CNN to recognize the images faster, more accurately, and output a higher quality of an image.








III. Mathematical Model

The convolution neural network works by taking the image and converting the image to a size that it can read. The network converts the image into layers and assigns it values to the image. The CNN must convert it into numbers because that is the only way it can see it. Once this happens it will start to

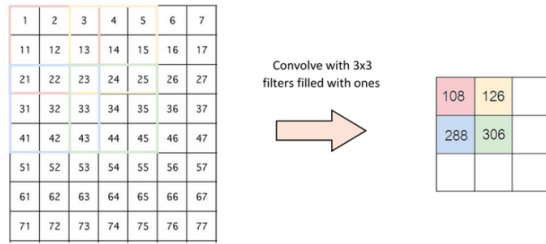
apply the filters or matrix it needs to recognize features of the images to figure out what it is, this is known as feature extraction [4]. How exactly this runs will depend on the model itself or the user input to set the dimension that the CNN wants to do it in. Once it has been through that process, the model will start implementing a filter or kernel matrix. The image will become pixelated to help define the images even more.

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

That filter matrix will go through every part of the image and delegate a number or perform an operation of some kind to show that there is an edge, a curve or something there. For an example:

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

The next part is the strides and padding which help with the images by convolving or shifting around to help sharpen the image so that the filter/kernel matrix can understand the shapes of the image. It places values that are important in the image.



The next crucial step is the pooling method, where inside the pool, it will enlarge and sharpen the areas where the values are large. This allows the pool to maximize at the places where the filter passed values through and falls within the parameters. The image would now slowly start to come together because of the process, which the translation invariant will now become an image when CNN outputs it. From there the CNN will be able to depict the image that user inputted and deduce what it is just by the parameters given; thus looking similar to this.

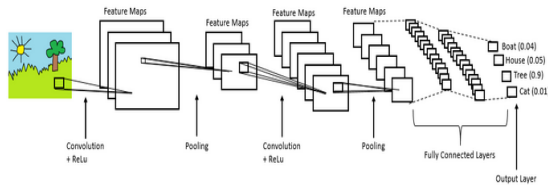


Figure 10 : Complete CNN architecture

Once the image is compiled, the CNN will determine what it is, like a dog, cat, airplane, car, etc, and provide details on how each category relates to it after feeding it the data already. Given the value of how similar the image that it is trying to determine is compared to the data of a general one, the accuracy can be obtained. CNN then obtains the highest ranked class and labels the picture as such.

IV. Implementation/Constraints

The whole idea of doing this is to learn, study, and understand what a convolution neural network can do, how else can be applied, and how it can be

improved upon. This project we are comparing different types of processors to implement CNN. These processors are all run in the same virtual environment, Jupyter Notebook. This is the main and easiest way to interface with the PYNQ-Z1 board and is therefore used throughout all the testing on all hardware platforms.

The first hardware platforms run on a standard CPU found in a desktop and laptop. For this testing purpose we are using an AMD Ryzen 7 3800X and an Intel i7-8550U. These general units allow us to run python programs and to use the libraries that are required to make the CNN work. This will allow us to gather basic measurements on it while trying to implement our basic performance metrics.

The second type of hardware is the PYNQ-Z1 board which uses the ARM architecture, while also being able to implement hardware FPGA (Field Programmable Gate Array) overlays to accelerate a function. This board runs Jupyter Notebook as well where the python code will be implemented. We then will run some performance benchmarking on the PYNQ-Z1 board and compare, which is faster to implement if we had to choose. From there we can see the actual difference in performance and make infer decisions about using convolution neural networks on python or through PYNQ-Z1 or other means.

Convolution neural networks have merits when it comes to learning about machine learning or even artificial intelligence. However, there are some constraints that were found while working on the project: that CNNs had a particular set of libraries needed for the program to work on Python, the application needed to be restarted to work again, and overfitting.

- 1) Libraries: Using CNN there are certain libraries that must be downloaded or if already

downloaded we must import them. That can become tedious because sometimes downloading the libraries doesn't work, thus it can put a hold on the project.

- 2) Application Restart: The application restart is geared towards the virtual coding of python. When logging back to the jupyter notebook, the program tends to lag or disconnect from their server every once in a while. That means restarting your program is necessary and must start from the beginning again; a problem that takes time away from the main project.
- 3) Overfitting: This happens when data in the set doesn't require any modification, but CNN assumes that it has to. Therefore, CNN's own performance is affected because of that image that it thought it needed to improve but didn't [2].

V. Results

After running the performance metrics on the PYNQ board and through the virtual environment on our own computer, the results were astounding. We see that when running on a different CPU whether on a laptop or on a desktop, the performance of it has little correlation between power, speed, and accuracy. The same can be said with the PYNQ-Z1 board in which, if we run on the board just the ARM CPU, the performance is similar. The components that make up CNN can drastically change the accuracy of it, but in this model all the results ran around 70 percent. When the FPGA component in the PYNQ-Z1 board is run, the speed of the images processed are increased significantly.

For example, the AMD Ryzen 7 3800X is on a desktop PC which consumes more power than an Intel i7-8550U, which is done on a laptop, and the speed in which it processes the images is a little more than 3x

faster yet the power consumption is around 13x higher. When run only on the PYNQ-Z1 board the performance was similar to the Intel CPU, yet it consumes even less power. The result that outperformed the others is the implementation of FPGAs. In this test when a CNN is ran on the FPGA component of the PYNQ-Z1 the performance was over 200x faster. The main reasoning for this is that the CPUs are all general purpose whereas the FPGA is application specific so therefore that is the only thing it can do but that means it is more efficient than Ryzen 7 3800X, Intel i7-8550U, and ARM CPU. The takeaway from this is that CNN is useful and an interesting topic to learn if wanting to know more about machine learning. If wanting to create a better and faster network, having a dedicated hardware component is needed to get the desired results.

VI. Conclusion

In conclusion, what we have learned about convolution neural networks is that it is a big next in learning the basic information about machine learning. We know that different processors do change the performance of how the CNN reacts in determining what the image is. Though having the best processor speed or clock rate doesn't always mean it will increase productivity because sometimes the data can alter its performance on the model also.

VII. Acknowledgement

We want to thank Xilinx for providing us the PYNQ-Z1 board to allow us to test our python programs on it to see if it can compete with other more mainstream hardware.

VIII. Reference

- [1] I. Mamun, "A Simple CNN: Multi Image Classifier," *Medium*, 08-Apr-2019. [Online]. Available: <https://towardsdatascience.com/a->

simple-cnn-multi-image-classifier-31c463324fa.

[Accessed: 27-Apr-2020].

- [2] J. Brownlee, “Overfitting and Underfitting With Machine Learning Algorithms,” *Machine Learning Mastery*, 12-Aug-2019. [Online]. Available: <https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/>. [Accessed: 27-Apr-2020].
- [3] P. Skalski, “Preventing Deep Neural Network from Overfitting,” *Medium*, 11-Feb-2020. [Online]. Available: <https://towardsdatascience.com/preventing-deep-neural-network-from-overfitting-953458db800a>. [Accessed: 27-Apr-2020].
- [4] K. Sorokina, “Image Classification with Convolutional Neural Networks,” *Medium*, 26-Feb-2019. [Online]. Available: <https://medium.com/@ksusorokina/image-classification-with-convolutional-neural-networks-496815db12a8>. [Accessed: 27-Apr-2020].