Implementations of Convolutional Neural Networks

James Wensel
Computer Science
Kansas State University
jdwensel@ksu.edu

***Abstract*— Machine Learning is one of the newest and most important fields in computer science. It is one of the fastest growing fields, and, with the new interest in artificial intelligence, along with the distribution and use of machine learning models in many different industries; finding the most efficient way to train and test a model is a necessary step to improving the effectiveness and usefulness of machine learning. An important task in machine learning is image recognition, its fast and efficient image operation is the basis for many artificial intelligence systems including health care diagnosis and video analysis. One of the more promising types of machine learning models for image recognition is a Convolutional Neural Network, which uses Convolutional filters to help classify the data. The purpose of this paper is to implement a Convolutional Neural Network on multiple different devices with different specifications to do image recognition on a shared dataset. The neural networks will be tested for time taken to build, train, and test the model, as well as accuracy of the model when implemented on different devices.**

I. Introduction

As early as the 1950's, a machine learning algorithm was created by A. L. Samuel, at the time an engineer for IBM, that could learn to play checkers [1]. While this was not the complex deep learning neural networks that have since become extremely popular, it created a method of game-tree search called Alpha-Beta pruning which is still used to this day. While he did not use deep learning, Samuel did mention a method "which might be called the Neural-Net Approach" [1], showing even as early as the 1950's, Deep Learning and the most modern machine learning methods were being investigated.

Machine learning, in particular neural networks, have incredible amounts of utility today. Neural networks are best applied to problems of pattern recognition and can often perform better than humans, and better then what humans could otherwise create, in those areas [2]. Neural networks can accomplish these tasks faster, more efficiently, and with greater accuracy then other methods. While this may at first seem significant, pattern recognition is one of the most basic tasks for main different problems. From stock market analysis [3] to disease diagnoses [4], pattern recognition is central to many important and growing fields. Because of this, neural networks have been applied to problems in every industry and field, relying on the strength of already created methods to gain increases in performance that would otherwise not be possible. Neural networks are now being applied to almost every industry, making machine learning one of the largest and fastest growing technologies in the world.

According to the statistics portal Statista, the AI market will grow to be a 126 billion dollar industry by 2025 [5]. With an industry this big, it is necessary to investigate the most efficient ways to both create and use neural networks, trying to maximize time, resource utilization, and model efficiency. This paper seeks to do just that by creating an implementation of a neural network on multiple different platforms, and timing and comparing the results in order to show the performance of machine learning on readily available technology, and show the potential for growth with other, more powerful technologies.

The main purpose of the neural network created for this paper will be to image classification. This is a central problem of machine learning, as utilization of machines can lead to significant increases in both accuracy and speed of image classification. It can be utilized for many purposes, from reading patient X-rays to help diagnose Rheumatoid arthritis [6], to activity recognition using traffic surveillance footage on edge devices [7]. Even these two different industries have very drastically different resources, with the model trained for X-ray image classification being a computer with many more resources then an edge device in fog computing, which mostly utilizes the internet of things. With such varied distinction between both the uses and the resources that are available for a neural network, and the importance of the uses of image classification, it is an obvious choice as the main purpose of the implementation of a general neural network on multiple different devices.

The specific type of neural network that will be used for this task is a Convolutional Neural Network (CNN). A convolution neural network is a network that applies a square filter of different weights to the image, sliding over every pixel in the image, as can be seen in Fig (1). These filters help to perform feature extraction, essentially allowing the neural network to "learn" different features of an image. This "learned" feature extraction makes CNNs perform better than other man-made methods, as other man-made image classification systems need a hand-engineered filter to be created to perform feature extraction [8]. This takes much more time, effort, and a deep understanding not only of the problem space, but also of the image and feature types to create adequately. CNNs help to circumvent this problem entirely, while still managing to be relatively simple neural networks, both in concept and design. The leads to CNNs having very good performance on most image recognition and classification tasks.

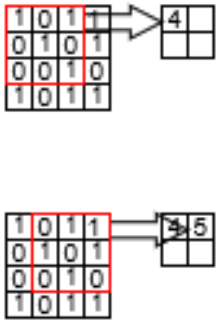

Fig. (1)

II. Related Research

A. Convolutional Neural Networks

Convolutional Neural networks are a subject that has been researched regarding many different existing problems and datasets. They are used in feature extraction for determining food safety [9], detecting facial manipulation in videos from Deepfakes [10], and even forecasting the future of COVID-19 [11]. Clearly, CNNs have been used in a wide variety of fields, all having their own unique problems and datasets. They have been shown to be rather effective as well. For example, the network developed to find Deepfakes became up to 99% effective [10] at identifying when videos had been modified.

The effectiveness and prevalence of this technology make it a prime target for this paper, as the purpose of this paper is to build on this previous work and show the meaningful ways in which CNNs can be improved. Showing the ability for improvement in CNNs could potentially lead to new improvements in the industries that already have functional CNN applications.

B. Neural Networks on Edge Devices

Programming on edge devices, in the case of this paper a Raspberry Pi 4, is a newer concept related to Cloud Computing and Specifically Fog Computing. According to Munir et. al. [12], Cloud Computing is the use of many devices to pass data to a central server for computations, while Fog Computing is the use of edge devices for calculations and data passing, and a central server is used for more data intensive tasks as well as overall diagnostics. In a fog computing environment, a majority of the tasks will be completed on edge nodes, which shows a clear need for more research into the efficiency and effectiveness of different machine learning methods on edge devices.

This paper will serve as an entry point into this field, as a Convolutional Neural Network will be trained and tested on a Raspberry Pi 4, which is itself an edge device, but also has comparable power to other edge devices. The different results gained from the neural networks on the Pi could serve to influence future fog computing technology.

III. Methodology

In order to complete image classification with a neural network, a neural network must be built and trained using a sufficiently large, labeled dataset. For this paper, a CNN was developed using the MNIST dataset. There are two key steps that must be taken before training the neural network: data cleaning and model creation. Once these two tasks have taken place, the devices to be used are selected, the model is then put onto these devices and tested. The testing of the models will be discussed in detail in the next section.

A. Data Cleaning

The MNIST dataset is a collection of 70,000 handwritten digits (60,000 training images and 10,000 test images) stored as 28x28 pixel images [13]. A few of these images can be seen in Fig (2). Each image has an associated label stating which digit (0-9) the image is of. This collection of data and labels allows for obvious translation and use of this data in supervised learning for image classification. For supervised learning, it is necessary to have labeled data so the predictions of the neural network can be tested against the correct label for the data, and the difference between these two values (calculated by a loss functions) can be used to update the neural network to make better predictions in the future. The main benefit of the MNIST data set, other than its size and direct translation to the problem task, is that it has already received a majority of the preprocessing steps that would be necessary for a fresh dataset. All images have been size-normalized and centered [9], meaning there are no additional steps needed to preprocess any of the images. Normally, one of the first data preparation steps is splitting the data into a training and testing set, but the data is already split up into these sets, so that is not necessary either. The only data cleaning steps necessary are data reforming and normalizing for our CNN.

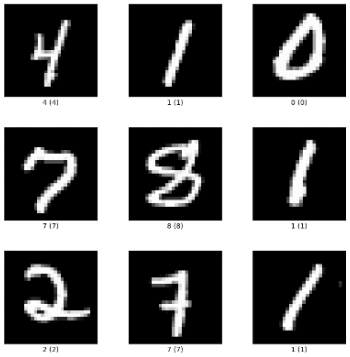

Fig (2)

With the original preprocessing of the images done, the first necessary step is reformatting the dataset. The dataset must be formatted correctly to be trained by a neural network. With 28x28 images, it must be fed into the neural network in an array of shape (28, 28, 1). This is because the CNN is expecting each value in its own array, so there must be an outer array (one for every row), containing 28 inner arrays (one for every column), each containing a single data point. The pixel data from each image must be formatted in this shape, and the overall dataset is stored in an array of shape (60,000, 28, 28, 1), containing all 60,000 training images. Once the data is in the shape, it can be processed by the CNN.

The last data processing step that is needed is normalizing the data values. Each pixel value is initially an int between [0, 255], but neural networks deal much more easily with values between [0,1]. To achieve this, each value is converted to a float to allow for maximum precision, and divided by 255, giving the

desired value range. Once this is completed, the data has been processed and cleaned, and is now ready for training the neural network.

B. Model Creation

The steps required to create a neural network are deciding which software to use, deciding on the type of model, determining the number of layers and the kind of layers, and determining the batch and epoch sizes.

For the neural network used in this paper, Keras was used as the main software to build our model. Keras is a python API built on TensorFlow 2.0 [14] that allows for easy and efficient neural network model creation. It comes with CNN functionality, as well as functionality for all other layers necessary to create a CNN neural network. It is the obvious choice for the purposes outlined in this paper. The next step, determining the type of model, has already been completed, and a CNN was selected as the type of model for its efficiency and effectiveness at image classification and recognition.

The general setup of a CNN involves any number of convolutional layers. Each different convolutional layer generally has a different number of filters being applied to extract different features. The number of layers and number of filters per layer can be changed for different results, and testing of this is shown in the experiments section. The CNN layers feed into a pooling layer which reduces the output feature size from the filters. This helps to create more manageable data sizes for the following layers. Next is a flattening layer that formats the data into a flat array so it can be input into a Dense layer. Dense layers are general, fully connected neural network layers, and there can be any number of them. This general structure can be seen in Fig (3). The number of dense layers used is 2, 1 for linear learning and another for determining the output. The number of dense layers could have been changed, but for simplicity they were left the same. There are also dropout layers added after the convolutional layers and after the dense layers. These decrease immediate performance of the model but prevent overtraining on the training dataset and were included because neural networks used in image recognition would most likely be trained with dropout layers.

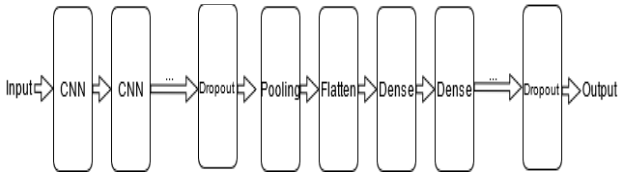

Fig (3)

The final decision is batch size and number of epochs for the training data. Batch size is the number of data points put through the neural network with each step, and epoch size is the number of times to train on the data. For larger batch sizes, more data will be used at a time and less will be learned from each individual data point, but the time required for training the model will be reduce, and the process of testing this will be described in the next section. For multiple epochs, the data will be trained on multiple times, meaning all 60,000 training data points will be fed through the network once per epoch. A larger number of epochs will allow the dataset to be artificially increased in size, but can cause overtraining issues, showing the importance of dropout layers. The experiments used to test this are detailed in the next section.

C. Devices

To test the performance of CNNs using the MNIST dataset, different devices are used to both train and test values. The tests are more thoroughly outlined in the next section. The devices used for these tests are Google Colaboratory (Google Colab), a laptop, a desktop, and a Raspberry Pi 4. Google Colab is a tool for running online Jupyter notebooks, which are online interactive python files that allow for segmentation and selective running of different sections. The notebooks run on Google servers that allocate 12.78 GB of RAM and 107.77 GB of disk space, and runs the files using Google's Cloud Computing Backend. The laptop used has an Intel Core i5-7300HQ CPU running at 2.50 GHz with 8 GB of RAM. The Desktop has an Intel Core i7-9700F CPU running at 3.00 GHz with 16 GB of RAM. The Raspberry Pi 4 has a Cortex-A72 processor with 2GB of RAM. These devices have vastly different resources which serves to illustrate the potential devices that could be used to train a model in real-world applications Google Colab shows cloud computing, the laptop shows the power of a basic computer, the desktop shows the power of a higher-end computer, and the Raspberry Pi 4 shows the use of a model on an edge device. Testing of these devices is outlined in the next section.

IV. Experiments

There were 3 main categories of tests performed. First was testing the model layout by changing the number of Convolutional layers. Then second involved changing the batch size. Finally, the number of epochs was changed. All three test categories were performed on each device, calculating the loss and accuracy each time, and timing how long it took to build the model, train the model, and test the model.

To test the different number of convolutional layers, each neural network was trained with a batch size of 32 and run for 15 epochs. These numbers were chosen as baselines and kept consistent among all the devices. Each device was tested on the performance difference of 1 vs 2 convolutional layers. The Desktop was also tested on 3 convolutional layers, as well as tested with 0, 1, and all dropout layers, and tested with only one dense layer. This was only done on the Desktop to show the changes caused by these different neural network setups, they were not done on any of the other devices as the dropout layers are critical to preventing overtraining, and the changing of the dense layers would have added increased complexity to tests performed that did not have an associated benefit.

To test the effect of different batch sizes, each device was tested with 2 convolutional layers over 15 epochs, except for the Raspberry Pi 4 where only 1 convolutional layer was used for reasons that will become apparent in the next section. The batch size was changed from 8 to 1024, being doubled each time.

The final test, testing different number of epochs, was done in a similar manner to the previous test. It was done with 2 convolutional layers (again except for the Raspberry Pi 4), and with a batch size of 32. The number of epochs was varied from 5-20 in increments of 5, and then each model was trained on 100 epochs. The Desktop was also trained with varying numbers of epochs from 30-90, increasing by 10 each test to better show the transition from the lower number of epochs to higher numbers. The results of these tests are shown and discussed in the next section.

V. Results

A. *Model Layout Experiments*

After training each model with a batch size of 32 and 15 epochs, the results of changing the model layout are shown below in Figs (4-7). The results are measured by loss (the average distance between the predicted and true label), accuracy (the percent of correct guesses from the model), build time (the time taken to build the model in seconds), train time (the time taken to train the model in minutes), and test time (the time taken to test the model in seconds):

Conv Layers	Google Colab				
	Loss	*Accuracy*	*Build (s)*	*Train (m)*	*Test (s)*
1	1.18353	79.42%	0.36738	7.6697	1.8329
2	0.55903	86.11%	0.23461	28.9617	4.2357

Fig (4)

Conv Layers	Laptop				
	Loss	*Accuracy*	*Build (s)*	*Train (m)*	*Test (s)*
1	1.11661	81.56%	0.16181	4.8503	1.0013
2	0.61088	85.53%	0.9821	17.9576	2.617

Fig (5)

Conv Layers	Desktop				
	Loss	*Accuracy*	*Build (s)*	*Train (m)*	*Test (s)*
1	1.25254	78.69%	0.04225	2.4453	0.4999
2	0.64695	85.19%	0.05005	8.71949	1.0898

Fig (6)

Conv Layers	Raspberry Pi 4				
	Loss	*Accuracy*	*Build (s)*	*Train (m)*	*Test (s)*
1	0.0364	98.73%	9.187	110.7237	25.85
2	0.0286	99.15%	9.4542	1189.27	264.9

Fig (7)

The first thing that becomes apparent from these results is that 2 convolutional layers results in both less loss and a higher accuracy then a single convolutional. This can be visualized in Fig (8). However, this increase comes at the cost of training and testing time. These increases can be seen in Fig (9). For all but the Raspberry Pi 4, the accuracy increases between 4-9%, which is significant, but the training time also takes a significant hit, increasing between 6-20 minutes. The training time is generally increased by 4 times its single convolutional layer training time. With testing time, the increase is only .5-3 seconds, again for all but the Raspberry Pi, which is not a significant increase. Because the only time that had significant increases is the training time, it will be the only time metric focused on for the remainder of the analysis. Also, accuracy will be the only correctness metric used after this section for simplicity, but loss is equally valid and could be used instead.

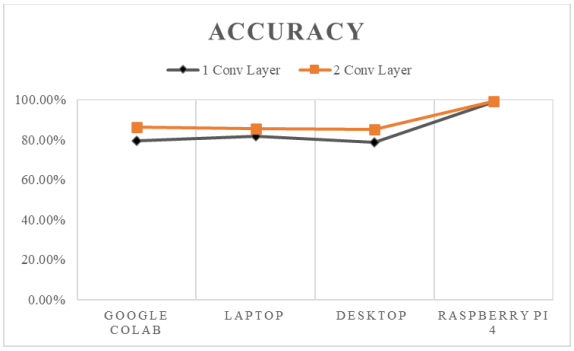

Fig (8)

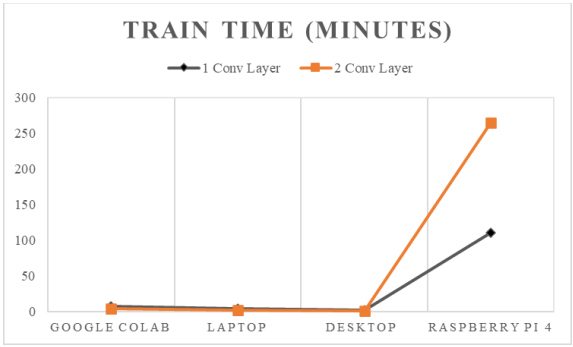

Fig (9)

For the Raspberry Pi, the results of the tests are much worse. The accuracy only increases by .42%, which is an insignificant amount of increase, while the training time increases by 1000 minutes (a full 16 hours). The testing time also increased by 220 seconds, or 3 and a half more minutes. These increases are extremely large for little to no gain in performance. For this reason, the Raspberry Pi will be using a single convolutional layer for all future tests and will also be considered separately from the rest of the devices. The results of the devices without the Raspberry Pi outlier are shown in Fig (10-11). It is also important to note that the Raspberry Pi had the highest accuracy by far, with its 1 convolutional layer accuracy being over 12% higher than the best 2 convolutional layers set up for the other models, meaning the worse version of the Raspberry Pi 4 was better than the best version of the rest of the models.

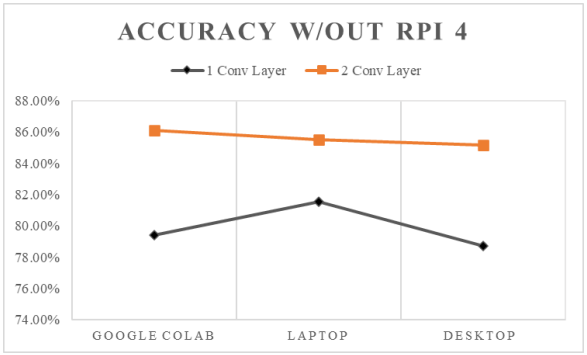

Fig (10)

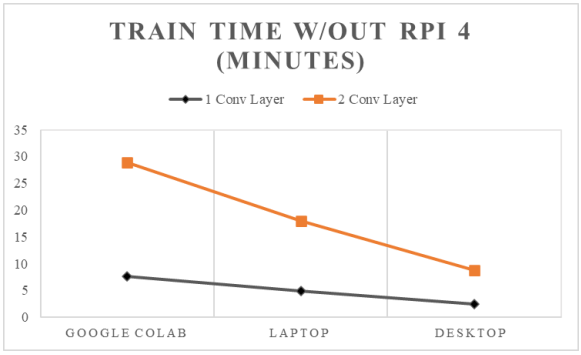

Fig (11)

After these initial tests, there was an additional test done with the Desktop to show the accuracy and training time of 3 Convolutional layers, and the results can be seen in Fig (12-13).

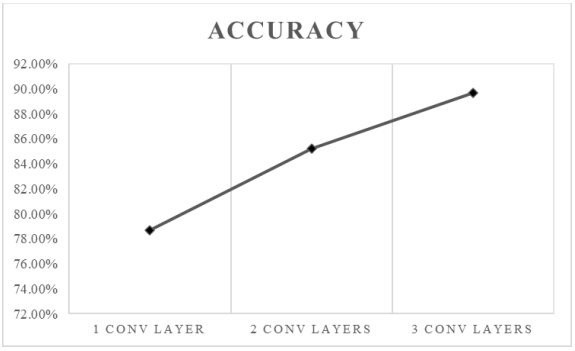

Fig (12)

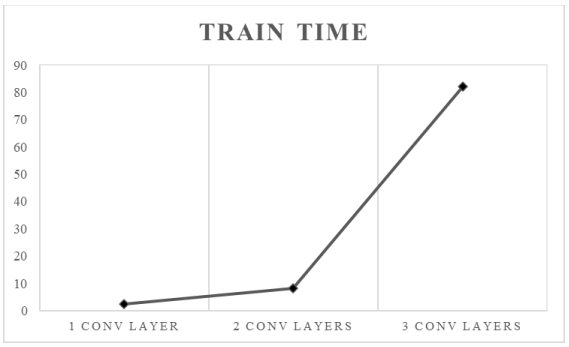

Fig (13)

As can be seen and was expected, there is an increase from 2 to 3 convolutional layers, but it is less significant than before, and the increase in time from 2 to 3 convolutional layers is also much more drastic, taking more than 10 times longer to train. Two convolutional layers is a good compromise between efficiency and effectiveness. It produces better results than a single convolutional layer but takes more than 8 times less time than 3 convolutional layers. Because of this, it will be used for all following tests, except on the Raspberry Pi where a single convolutional layer will be used.

The final experiments done regarding the structure of the neural network focused on changing the number of dropout layers. As stated in the methodology section, dropout layers are essential to help prevent overtraining, but they reduce the accuracy in the training data, and this can be seen in Fig (14). The training, testing, and build times had little change between these different implementations and are not shown.

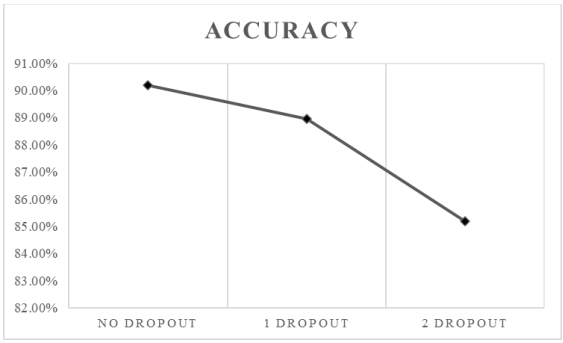

Fig (14)

As Fig (14) shows, the dropout layers do decrease the accuracy, from over 90% to over 85%. While this drop is important to note, the benefit of not overtraining is much greater than the downside of a slightly less accurate model on the training set.

B. Batch Size Experiments

For the batch size experiments, the batch size was changed from 8 to 1024, doubling each time. For Google Colab, the Laptop, and the Desktop, the accuracy results can be seen in Fig (15) and the timing results can be seen in Fig (16). The accuracy and timing data for the Raspberry Pi 4 can be seen in Fig (17) and Fig (18). The accuracy and timing data can be seen in Appendix A Tables (1-2).

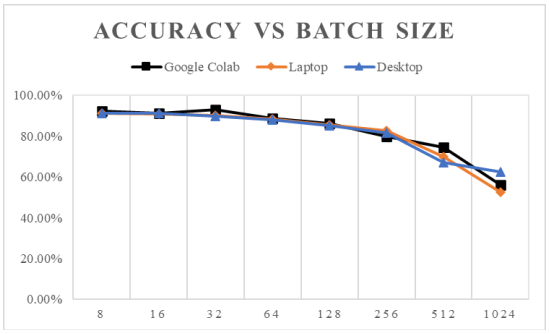

Fig (15)

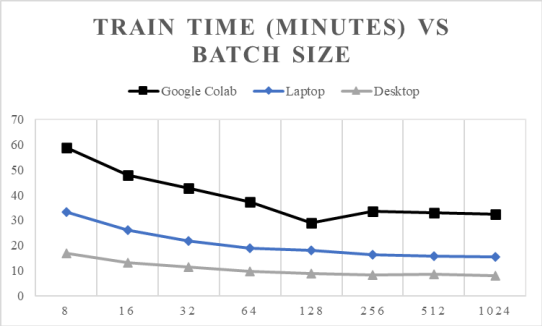

Fig (16)

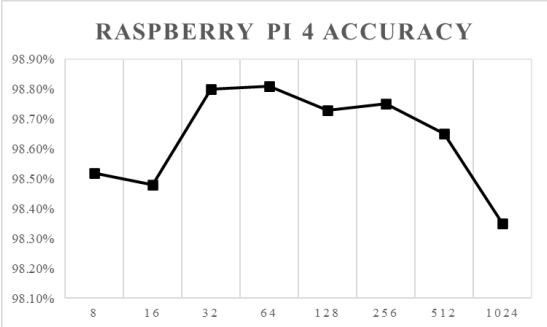

Fig (17)

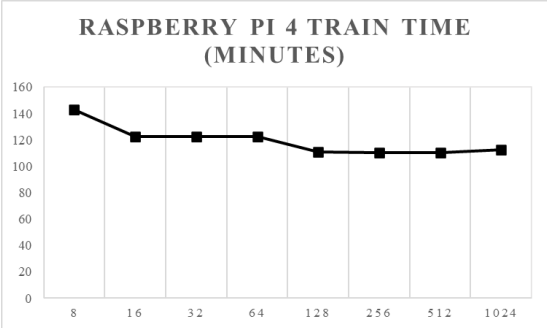

Fig (18)

For the Non-Raspberry Pi group, with each increase in batch size there is a decrease in accuracy and a decrease in training time. This is due to the impact of increasing the batch size. The higher the batch size the larger the number of datapoints considered at the same time, which causes larger steps to be taken. This ultimately causes a decrease in accuracy. However, the decrease in training time for each is also significant: the Google Colab model decreased its training time by 50 minutes, the Laptop decreased its training time by 15 minutes, and the Desktop decreased its training time by 9 minutes, each reduced its original training time to under half its original value. Therefore, it is necessary to select a value that optimizes accuracy while considering the time available and the system being used. For the last experiment, a batch size of 128 was used, as the results at that batch size still have high accuracy while not requiring excessive times to train.

C. Epoch Experiment

For this experiment, the number of epochs was changed from 5-20 with steps of 5, then 100 to show the impact of training multiple times on the same data. The results from the non-Raspberry Pi 4 group can be seen in Fig (19-20), while the results from the Raspberry Pi 4 can be seen in Fig (21-22). The data from these tests can be seen in Appendix B Tables (3-4).

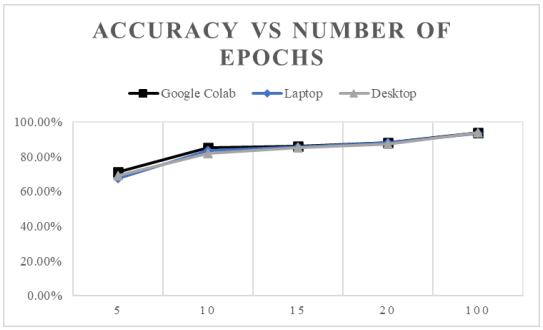

Fig (19)

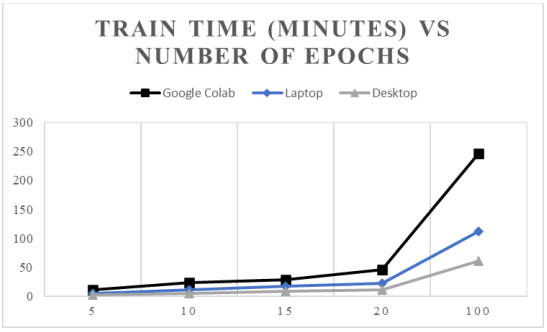

Fig (20)

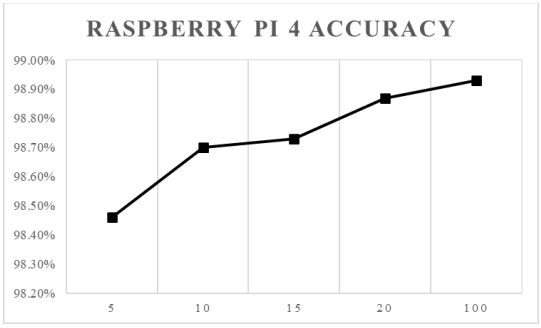

Fig (21)

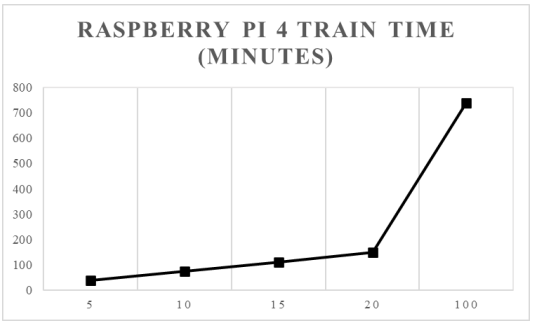

Fig (22)

As expected, all three non-Raspberry Pi devices experienced drastic increases in accuracy. All three begin at approximately 70% accuracy, and they end at approximately 93% accuracy, a drastic improvement. The increase in time for Google Colab is sporadic, but the Desktop and Laptop models increase in time almost linearly, going up 1 minute per epoch and 40 seconds per epoch, respectively. For the Raspberry Pi 4, the results are similar to the results of the last experiment, with the increase in accuracy between 5 and 100 epochs being .5%. However, the time is drastically different between the two tests, with 100 epochs running in about 12 hours while 5 epochs run in under 40 minutes.

The final tests done on the Desktop used 30-90 epochs increasing by increments of 10, and the results of this are shown in Fig (23-24), while the data is in Appendix B Table (5).

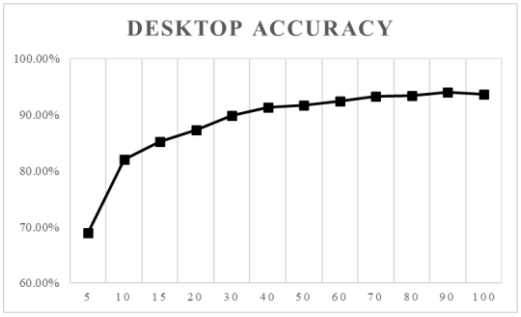

Fig (23)

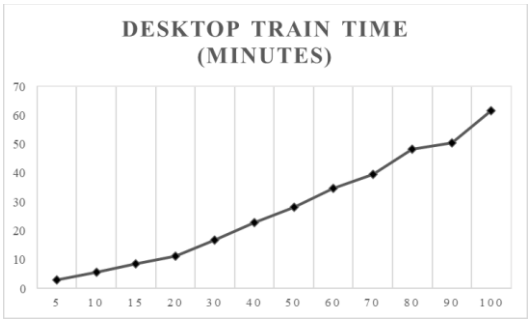

Fig (24)

This extra data serves to reinforce the ideas discussed after the first set of experiments. It shows the diminishing return of increasing the epochs much more clearly, and models the linear increases with the number of epochs very well.

D. Best and Worst Possible Models

After all other tests had been completed, the best and worst possible networks were created on the Desktop to see the contrast between the two. The best network consisted of 3 convolutional layers, 2 dense layers, and dropout layers, and ran with a batch size of 8 and 30 epochs (more could have been used but after 30 the returns were small enough, they were considered insignificant). The worst network had 1 convolutional layer, 1 dense layer and dropout layers, and ran with a batch size of 1024 and 5 epochs. The results of the two models are shown in Fig (25-26).

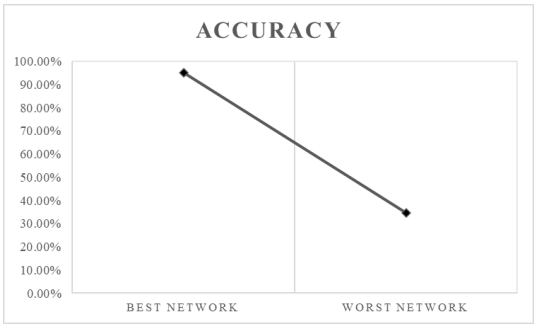

Fig (25)

Fig (26)

The results show the expected relationship, with the best network achieving 95.15% accuracy, and trained in 74 minutes while the worst network had an accuracy of 34.59% and trained in less than a minute.

VI. Conclusions

The results of the experiment shows important principles that should be used when creating models. The first is shown by the consistency of building and testing times amongst the different models, batch sizes, and number of epochs. This tells us that regardless of the setup of the model we are using, we can expect certain speeds when building and testing our models. This is an important result. Building a model only occurs once, but testing a model occurs every time the model is used to predict future results. This means that every time after the initial training a model is used, the time it will take to give a result is constant. This allows for pretrained models to be distributed to different environments with an expected testing time and can even allow for real time systems as the testing time is predictable.

The next important result is the impact of all the different metrics on accuracy and training time. For all the performed tests (except those performed on the Raspberry Pi 4 which will be addressed later), when a change in model design or training resulted in a more accurate model, it also resulted in an increase in training time. This relationship shows the necessity for an understanding of both the problem space and design restrictions before creating and training a model. If it is necessary to have the highest possible accuracy, a large amount of time will be required to train the model. If time is a restriction, there will be a hit to accuracy to achieve the necessary speeds. As was shown, there are many different

ways to change the speed and accuracy, and each change had different levels of impact on the result, but the process followed in this paper can be used to determine an approximate accuracy and training time for any networks created on similar devices.

Another very important result is the impact of process capabilities and RAM on performance. As could be expected, the more powerful the processor and the more ram, the more quickly the model trained. For every model combination, batch size, and number of epochs, the devices ran in times relative to their components. This means one way to achieve faster model training, other than through changing of model parameters, is through improving the device the model is being trained on. It is extremely important to note that the increase in computing power had no impact on the accuracy of the model. This means that, while a model can be trained faster on a faster device, the accuracy cannot be improved by means other than improving the neural network in some way.

The final results to discuss are the results seen by the Raspberry Pi 4. It had running times so slow that even its fastest running model was only reached by the other devices when they were running much more processor intensive model designs. However, it had extreme accuracy. The cause of this excessive accuracy could not be determined, however. With that in mind, if the goal is to achieve the most accurate possible model regardless of time, then a Raspberry Pi seems to be the answer. Another very important result from the Raspberry Pi is its testing time. While it was not uncommon for the Raspberry Pi to train for multiple hours at a time, the testing with 1 convolutional layer was consistently 25 seconds long. This means that a model, trained on the Raspberry Pi or not, could be used on the pi for any number of possible calculations, and as long as 25 seconds is an appropriate wait time, it can still be used as the controller for a complex system.

VII. Future Research Directions

The results from the experiments outlined in the paper have two obvious extensions. The first is doing similar testing on a different model type, specifically some kind of Recurrent Neural Network or a Deep Neural Network. The timing results from these different kinds of networks could provide similar further insight into the training time vs accuracy relationship, as well as showing if the static testing time is consistent among multiple kinds of neural networks or is simply a byproduct of a Convolutional Neural Network.

The next extension of this paper could be a similar setup with a similar model but using different metrics, such as changing the activation functions, the kernel size, or the number of filters. These metrics would all provide more insight on the convolutional layers and their effectiveness on this dataset. Another dataset could also be used to see if the results outlined in this paper are consistent for all data, and if there are any different data dependent results that can be found.

Finally, more research should be done into neural networks on edge devices. The extensions in this could involve different models applied to a Raspberry Pi, finding the testing time on edge devices of model trained on non-edge devices, and training of similar models on different types of edge devices. All of these would provide more insight into the results generated by the Raspberry Pi 4 in this paper.

VIII. References

[1] A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers," *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210–229, 1959.

[2] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[3] K. Pahwa and N. Agarwal, "Stock Market Analysis using Supervised Machine Learning," *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, 2019.

[4] T. Karatekin, S. Sancak, G. Celik, S. Topcuoglu, G. Karatekin, P. Kirci, and A. Okatan, "Interpretable Machine Learning in Healthcare through Generalized Additive Model with Pairwise Interactions (GA2M): Predicting Severe Retinopathy of Prematurity," *2019 International Conference on Deep Learning and Machine Learning in Emerging Applications (Deep-ML)*, 2019.

[5] S. Liu, "AI market size 2018-2025," *Statista*, 07-Dec-2020. Available: https://www.statista.com/statistics/607716/worldwide-artificial-intelligence-market-revenues/.

[6] A. Tashita, K. Morita, M. Nii, N. Nakagawa, and S. Kobashi, "Automated estimation of mTS score in hand joint X-ray image using machine learning," *2017 6th International Conference on Informatics, Electronics and Vision & 2017 7th International Symposium in Computational Medical and Health Technology (ICIEV-ISCMHT)*, 2017.

[7] T. Hussain, K. Muhammad, A. Ullah, J. D. Ser, A. H. Gandomi, M. Sajjad, S. W. Baik, and V. H. C. D. Albuquerque, "Multi-View Summarization and Activity Recognition Meet Edge Computing in IoT Environments," *IEEE Internet of Things Journal*, pp. 1–1, 2020.

[8] S. Saha, "A Comprehensive Guide to Convolutional Neural Networks-the ELI5 way," *Medium*, 17-Dec-2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. [Accessed: 08-May-2021].

[9] Y. Liu, H. Pu, and D.-W. Sun, "Efficient extraction of deep image features using convolutional neural network (CNN) for applications in detecting and analysing complex food matrices," *Trends in Food Science & Technology*, 2021.

[10] Z. Xu, J. Liu, W. Lu, B. Xu, X. Zhao, B. Li, and J. Huang, "Detecting facial manipulated videos based on set convolutional neural networks," *Journal of Visual Communication and Image Representation*, vol. 77, p. 103119, 2021.

[11] K. N. Nabi, M. T. Tahmid, A. Rafi, M. E. Kader, and M. A. Haider, "Forecasting COVID-19 cases: A comparative analysis between Recurrent and Convolutional Neural Networks," 2020.

[12] A. Munir, P. Kansakar, and S. U. Khan, "IFCIoT: Integrated Fog Cloud IoT: A novel architectural paradigm for the future Internet of Things.," *IEEE Consumer Electronics Magazine*, vol. 6, no. 3, pp. 74–82, 2017.

[13] "THE MNIST DATABASE," *MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges*. [Online]. Available: http://yann.lecun.com/exdb/mnist/. [Accessed: 08-May-2021].

[14] K. Team, "Simple. Flexible. Powerful.," *Keras*. [Online]. Available: https://keras.io/. [Accessed: 08-May-2021].

Figures

[2] "mnist : TensorFlow Datasets," TensorFlow. [Online]. Available: https://www.tensorflow.org/datasets/catalog/mnist. [Accessed: 08-May-2021].

IX. Appendix

Below are the tables for both the batch size and epoch number tests.

A. Batch Size

Batch Size	Google Colab		Laptop	
	Accuracy	*Train(m)*	*Accuracy*	*Train (m)*
8	92.15%	58.8756	91.28%	33.35556
16	91.23%	47.9029	90.96%	26.1843
32	92.86%	42.7417	90.20%	21.8015
64	88.93%	37.2201	88.52%	18.7666
128	86.11%	28.9617	85.53%	17.9576
256	79.99%	33.6919	82.57%	16.293
512	74.67%	32.8466	69.81%	15.7629
1024	56.11%	32.4688	52.55%	15.4952

Table (1)

Batch Size	Desktop		Raspberry Pi 4	
	Accuracy	*Train(m)*	*Accuracy*	*Train (m)*
8	91.28%	16.801	98.52%	143.16
16	91.22%	13.002	98.48%	122.624
32	89.99%	11.5266	98.80%	122.311
64	87.96%	9.8005	98.81%	122.593
128	85.19%	8.70949	98.73%	110.724
256	81.54%	8.29683	98.75%	110.108
512	67.01%	8.48852	98.65%	110.265
1024	62.59%	7.98701	98.35%	112.306

Table (2)

B. Epochs

Epochs	Google Colab		Laptop	
	Accuracy	*Train(m)*	*Accuracy*	*Train (m)*
5	71.02%	11.7741	67.38%	5.74339
10	85.30%	23.6064	83.44%	11.2725
15	86.11%	28.9617	85.53%	17.9576
20	87.88%	46.6865	17.9576	22.4327
100	93.82%	245.935	93.48%	112.669

Table (3)

Epochs	Desktop		Raspberry Pi 4	
	Accuracy	*Train(m)*	*Accuracy*	*Train (m)*
5	68.93%	2.91558	98.46%	37.1815
10	81.96%	5.60669	98.70%	74.1668
15	85.19%	8.70949	98.73%	110.724
20	87.27%	11.3323	98.87%	147.941
100	93.66%	61.6459	98.93%	737.591

Table (4)

Epochs	Desktop	
	Accuracy	*Train(m)*
5	68.93%	2.91558
10	81.96%	5.60669
15	85.19%	8.70949
20	87.27%	11.3323
30	89.79%	16.9026
40	91.29%	22.9273
50	91.66%	28.3107
60	92.38%	34.7734
70	93.20%	39.5345
80	93.33%	48.2187
90	94.00%	50.52
100	93.66%	61.6459

Table (5)