Research Experience for Community College Students: Optimizing a Convolutional Neural Network for Autonomous Jetson Image Classification

Michael Gee[1], Eddy Rodriguez[1], Jose Reyna Pardo[1], Jeffrey Mattos[1], Kevin Yamada[2], Nick Langhoff [2], Hao Jiang[2]

[1]Canada College, Redwood City, CA
[2]San Francisco State University, San Francisco, CA

Abstract

This research internship is planned over the course of 10 weeks of summer where community college student interns are assigned a graduate student mentor and a faculty advisor. This paper presents the details of this project, research, educational objectives, results obtained, and the student surveys assessing the outcomes. The goal of this research is to create a deployable autonomous robot for the Jetson TX1 for live image classification. To reach this goal, students will learn how to create a custom data set, how to turn data sets into models using Nvidia DIGITS, and create a custom neural network using caffe architecture. We achieved our goal by producing five iterations of the same datasets, creating a custom multilayer perceptron (SimpleNet), and creating a custom convolutional neural networks (KeviNet). The same dataset was deployed on all three neural networks and tasked to detect human objects at 5ft and 10ft. A standard network, AlexNet, was used as the baseline comparison. From testing, it was concluded that AlexNet was the most accurate with a large overhead, KeviNet has similar accuracy with AlexNet with moderate overhead, and SimpleNet being the least accurate with little overhead.

Introduction

The purpose of this project is to give community college students an opportunity to work with new tools and gain experience outside of the classrooms. The main goal of this project is to create a deployable custom neural network that can be utilized with the Jetson TX1. This custom neural network would emulate the capabilities of an autonomous self-driving vehicle. Through the use of deep learning, a model will be created to detect and classify humans. Additionally, the students will learn how to create a custom dataset for their suited environment. Overall, students will learn the basic structure for deep learning and its current state in the industry.

Background

Jetson TX1

The world's first supercomputer that is on a module, capable of delivering performance and power efficiency needed for computing applications. Created by NVIDIA, the Jetson TX1 runs off of Linux, and provides 1TFLOPS of FP16 computing performance in only 10 watts of power. The Jetson is equipped with the Nvidia Tegra X1, which combines a 64-bit quad-core ARM Cortex-A57 CPU with a 256-core Maxwell GPU. The design of the Jetson was built specifically for AI and deep learning all within a 170 mm x 170 mm size capable of fitting in a

mini-ITX motherboard. In the image below we get to see a visual representation of the Jetson TX1 developer kit. The design of the developer kit allows users to place the Jetson into any compact environment while still producing maximum efficiency and performance rates.

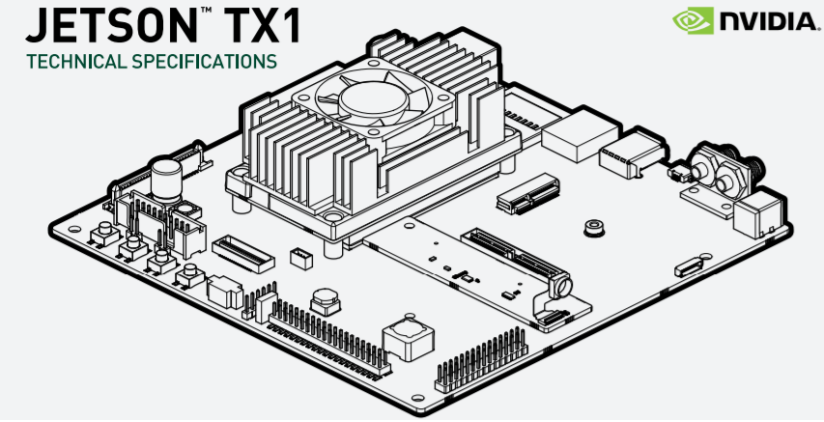


Figure 1. This image showcases the physical design of the Jetson TX1 Developer Kit. The Jetson TX1 itself is located underneath the fan module on the board.

These efficiency ratings are capable of even surpassing top notch CPUs such as Intel's core i7-6700K shown in the image below. This means that for machine learning the Jetson TX1 can compete with even the best CPUs and GPUs on the market, while still being a cost effective unit.

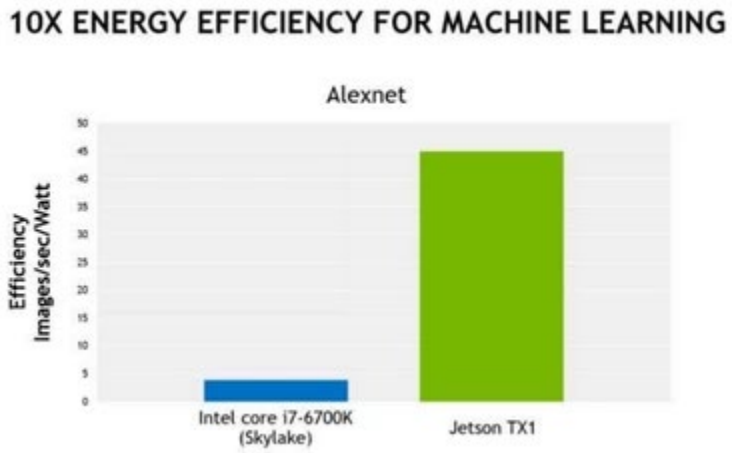


Figure 2. This graph shows the efficiency rating differences from the Jetson TX1 to the Intel core i7-6700K in terms of Images/sec/Watt. The Jetson TX1 and core i7 are both tested via Alexnet.

The point of utilizing the Jetson's strengths is due to the overwhelming advantage it has over other units in efficiency. Given the Jetson's specs, the biggest factor aside from its performance and efficiency is the multi-purpose applications it has in any scenario. Since the TX1 is such a small form-factor, we are able to use it in anything from cars, drones, robots, or even small hand-held devices. This puts the Jetson over all the other CPUs in the market since they require more variables to reach a similar efficiency rating. Our goal is to have a compact

supercomputer capable of processing multiple machine learning tasks in the most energy efficient way possible. The Jetson is perfect for this role because we are able to take its strengths and make the most of our given project.

As stated previously, the Jetson uses the NVIDIA's 256-core Tegra X1 processor. The Tegra X1 comes with a dual CPU + GPU combination, which allows the module to be efficient while taking up little to no space. The CPU that is built into the Tegra is the ARM Cortex-A57 meanwhile the GPU is constructed with a full-fledged Maxwell architecture GPU. The two work in combination allowing the Jetson to fully tackle heavy machine learning without using excessive energy consumption. This gives the Jetson a huge lead over other builds as many other computers have either weaker integrated GPUs in their CPUs, or have to use discrete GPUs to provide similar performance. This can prevent computers from reaching maximum efficiency at very low power levels. Inference differences from small to larger GPUs allows us to both see how fast and efficient each processor can be at optimal cases while simultaneously observing their utilization and efficiency in non-optimal cases. The table below compares the Tegra X1 to the Core i7 6700K, giving us the performance, power, and energy efficiency. The Tegra X1's performance to efficiency ratio is significantly higher than the CPU-based inference while also reaching similar absolute performance levels in FP16. This gives the TX1 a huge lead over CPU-based machine learning devices as it can hit the similar maximum performances as top notched processors, but at a quarter of the power consumed.

Network: AlexNet	Batch Size	Tegra X1 (FP32)	Tegra X1 (FP16)	Core i7 6700K (FP32)
Inference Performance	1	47 img/sec	67 img/sec	62 img/sec
Power		5.5 W	5.1 W	49.7 W
Performance/Watt		8.6 img/sec/W	13.1 img/sec/W	1.3 img/sec/W
Inference Performance	128 (Tegra X1) 48 (Core i7)	155 img/sec	258 img/sec	242 img/sec
Power		6.0 W	5.7 W	62.5 W
Performance/Watt		25.8 img/sec/W	45.0 img/sec/W	3.9 img/sec/W

Table 1 Inference performance, power, and energy efficiency on Tegra X1 and Core i7 6700K. We present cases without batching (batch size 1) and with large batching (128 on the GPU, 48 on the CPU). FP16 results are comparable to FP32 results as FP16 incurs no classification accuracy loss over FP32.

Figure 3. This table shows the statistics between the Tegra X1 and Core i7-6700K on AlexNet. The information provided shows difference in inference performance to power used between FP32 and FP16.

Deep Learning

To train the Jetson TX1, we'll be using a method of deep learning. Deep learning is a branch of machine learning and machine learning is a branch of A.I. Deep learning refers to deep artificial neural networks. The word "deep" refers to the number of layers in a neural network. As you increase the number of layers, neural networks have the capacity to become more intelligent. These layers are used to extract features from an object for image processing. With more layers, computational training becomes more intensive which reinforces our choice of the Jetson TX1.

Multilayer Perceptron

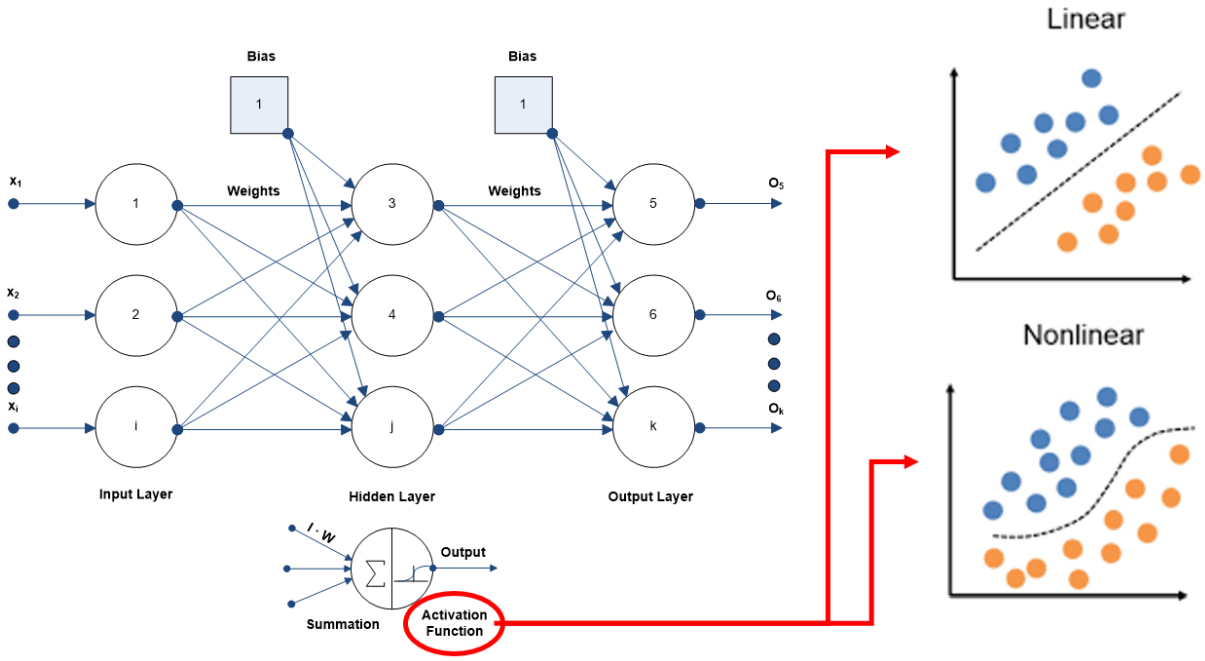


Figure 4. This diagram shows the basic structure of a multi-layer perceptron. Within the hidden layers lies the activation function. The activation function uses algorithms to perform linear and non-linear classifications.

To understand the architecture of a neural network, we need to understand the multilayer perceptron. The multilayer perceptron or MLP is the simplest form of a neural network. Multilayer perceptrons are feedforward networks. You can think of this as a guess and check method. To move forward, the network uses a forward pass function that allows data to move from the input layer, through the hidden layers, and finally to the output layer. The network can also move backwards. To move backwards, the network uses a backward pass function which uses partial derivatives to create gradients. These gradients are then used to optimize the network to achieve convergence. Convergence is where error can no longer increase and becomes stagnant.

Data begins with the input layer and are contained in cells. As these cells move forward, weights and bias can be applied to the cell to adjust for a more desired output. The cell will then move forward into the hidden layers. In the hidden layers, activation functions are used to compete non-linear classifications to find patterns. The MLP also has the ability to compute linear data. Linear data models correlation, meaning the prediction of how likely or unlikely events are to happen based on previous occurrences. Non-linear data models after inference, as in utilizing information about indirect relationships and previous occurrences. A non-linear classification allows the network to readjust itself when data becomes too complex or classification boundaries become complicated and erroneous. For a MLP, training consists of adjusting parameters to minimize error.

Convolutional Neural Network

A neural network is a set of algorithms modeled after the human brain designed to recognize patterns. A convolutional neural network or CNN is a deep learning algorithm that specializes in image recognition. CNNs have two unique abilities: the ability to assign significance to various aspects of an object within an image and the ability to differentiate an object from one another within the same image.

For our purposes we'll mostly focus on the convolution and pooling layers. The convolutional layer contains a set of learnable filters. These filters are made up of small matrices but are able to extend to the full size of an input image. There are three hyperparameters for the convolutional layer: depth, stride, and zero padding. Depth refers to the presence of edges and color. Stride refers to the number of pixels each filter will move to convolve an image. Zero padding allows for a desired output for the size of an image. After the convolutional layer is the pooling layer. The pooling layer functions to reduce the spatial size and number of parameters the network will need to perform computations. The pooling layer also uses a mac function to determine the region of the image that most likely contains the distinct feature. As the image becomes smaller, the network has an easier time learning distinct features.

Caffe and DIGITS

Caffe is an open framework for deep learning. It was developed by UC Berkeley AI Research PhD student Yangqing Jia. Caffe architecture is made up of a hierarchy of different components. Net, Layers, and Blobs and these are the anatomy of a Caffe model. Forwards and Backwards, Forward pass computes an output for a given input, and Backwards pass computes the loss gradient relative to learning. Loss function, error or objective function maps out the parameter settings dependent on loss. Solver optimizes the model based on Forward and Backwards in an attempt to improve Loss. Interfaces are the programming languages such as Python, C++, and MatLab that Caffe is ran on. Parameters are calculated by Solver gradients. Deep Learning GPU Training System, DIGITS, is the user friendly wrapper for NVCaffe and TensorFlow that is used for training DNN.

DIGITS, unlike Caffe is not a framework but an interface that makes training DNN like AlexNet, GoogleNet, LeNet possible. Within DIGITS the user has the ability to code a custom Neural Network using Python2. DIGITS requires the host computer to have Python2, CUDA, CuDNN, Caffe, and Graphiviz installed in order to work as intended. The DIGITS interface also allows for the creation of Datasets which are an essential part of the model training process. Datasets are typically thousands of images that are used for Model training. Without the DIGITS GUI user friendly interface sorting through thousands of images takes hours as the individual would have to manually place each image into correct class folders and randomize images to prevent overfitting.

Methodology

Figure 5. This diagram showcases the steps used to create, test, train, and modify our neural network.

The work flow for collecting data and testing consists of six steps: environment setup, data collection, custom neural network construction, model training, field testing, and data expansion. In environment setup, selected a location based on the limitations of our environment that would generate a modest dataset. For data collection, videos were recorded through the webcam attached to the Jetson. In custom neural network construction, a multilayer perceptron will created as our foundation to advance towards building a convolutional neural network. In model training, videos recorded were turned into images to be processed and turned into usable models in DIGITS. For field testing, a trained model is created and uploaded onto the Jetson for real-time inference testing. Finally in data expansion, the cycle would repeat with additional data added depending on the performance and accuracy of the real-time inference.

Environment Setup

For the environment setup, selection of an environment was based on reasonable conditions such as brightness, contrast, and vacancy. Due to the given location of our lab, the only suitable environment was the school building hallways. The hallways had ceiling lights every 14 feet from one another. In between each ceiling light, 7 feet apart, are areas with less lighting. This variable needs to be taken into consideration when collecting and processing data. Likewise, this leads to contrast. The lighting of the hallways made it ideal for dark objects but not bright objects. Lastly, is vacancy. The hallways were relatively vacant of objects and made it easier for the Jetson to learn the construct of empty space.

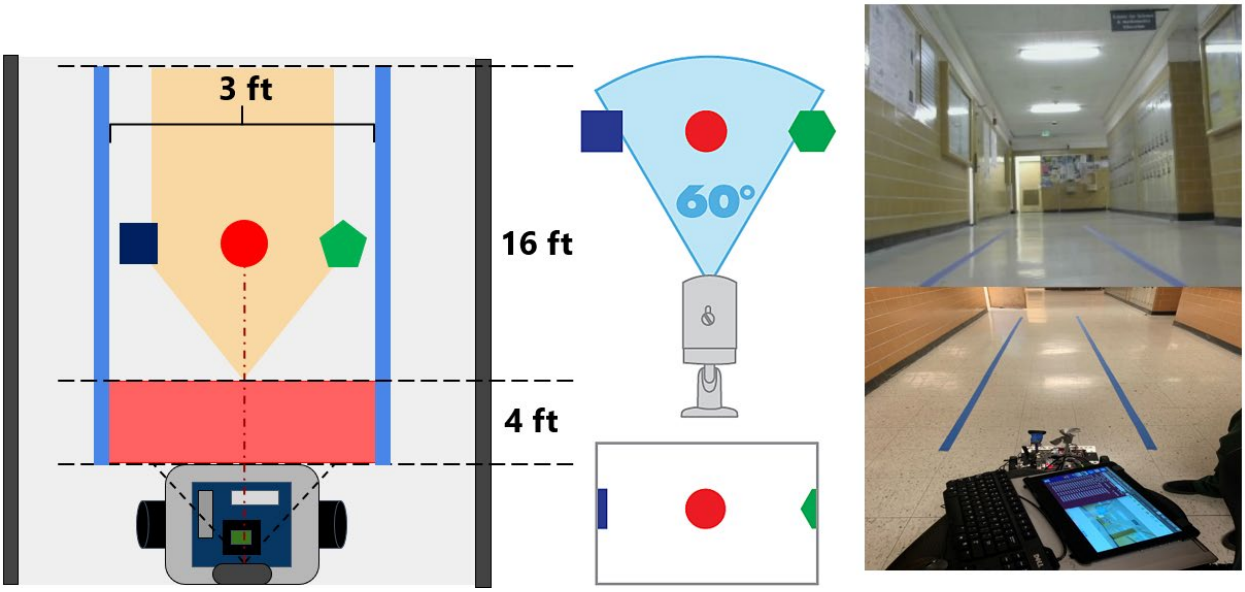


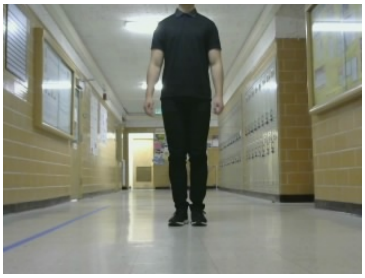
Figure 6. On the left side of the image is a diagram of the environment setup for the Jetson TX1. In the center of the image is a diagram of the Jetson TX1's field of view. On the right side of the image is a real-life depiction and point of view of the Jetson TX1.

Data Collection

Before data collection, we needed to determine classes to categorize our objects. The first class is an empty class. This class holds different variations of our hallway without any objects. The next class is humans, this class holds different variations of humans standing in particular positions in the hallway. However, we eventually changed from object-oriented classes to distance-oriented classes. With the prearranged Jetson TX1, this system lacked the functionality of sensors to detect objects in close proximity. To compensate for the lack of sensors, we replaced our human class with a 5 feet and 10 feet class to emulate distance.

For data collection, we recorded video using computer vision through a usb Logitech webcam with the following capabilities: 720p resolution, 30 frames per second, and 60 degree field of view with vertical adjustability. Videos recorded were converted into 320 x 240 resolution images. Looking at **Image 6**, the rear positioning of the camera was chosen to capture equals amounts of the floor, hallway, and ceiling. Due to the rear positioning of the camera, the Jetson has a blindspot of 5 feet. The 5 feet class begins where the boundary of the blindspot ends. Additionally, a 10 feet class was chosen as both a precautionary measure and test for the performance of incoming objects at a distance.

Table 1. This table shows examples of the three following classes for our dataset

		
Empty	5ft	10ft

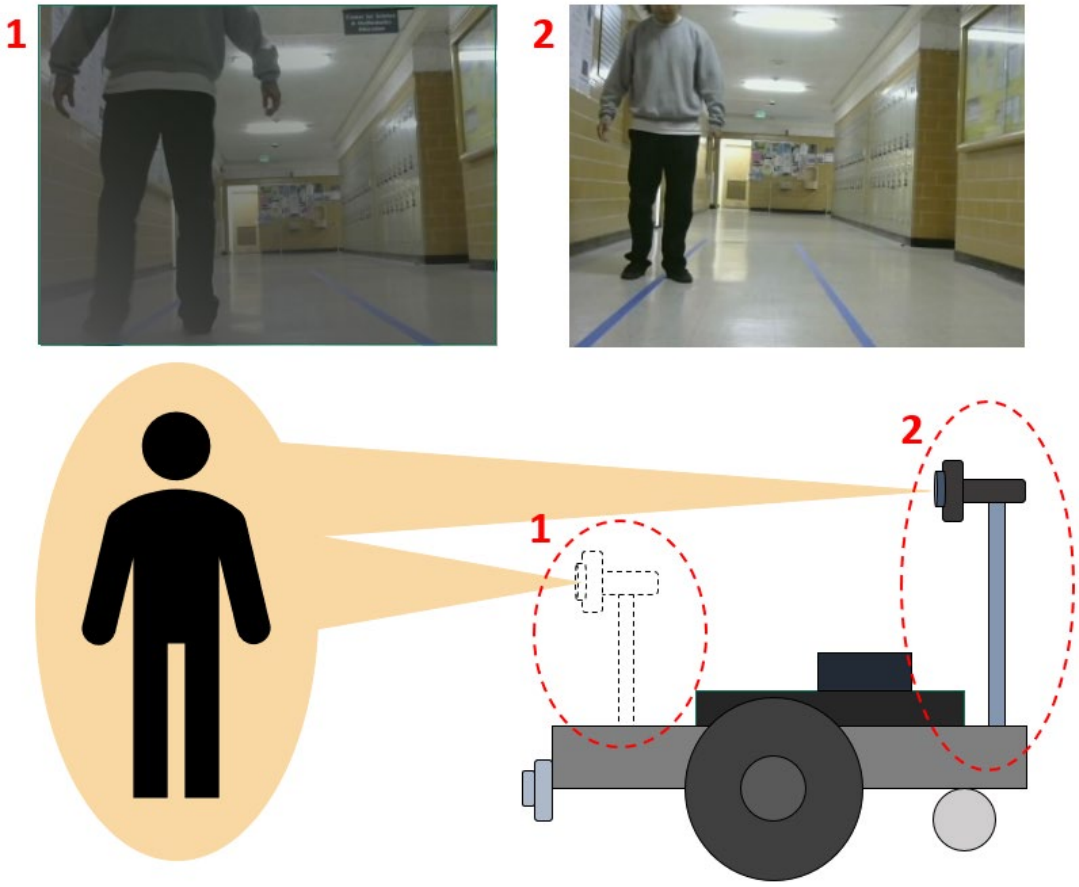


Figure 7. This diagram depicts the qualities of two different viewpoint positions, the (1) front and (2) rear.

Looking at our collection of data, there are six variables that need to be taken into consideration: viewpoint, truncation, illumination, clutter, clarity, and size. Looking at Image #, we have a diagram showcasing two viewpoints. Analyzing both images, we notice that the viewpoint affects the six variables. In the front viewpoint, the 5 feet blindspot was removed and focus is shifted more towards the floor. Also, the top half torso has been truncated and obstructs the light source. With less illumination, the clarity of the object decreases and impedes the distinct features that make up the shape of a human. Switching to the rear viewpoint,

illumination and clarity increases while truncation and size decrease, and clutter remains neutral in both viewpoints. It is clear that with a rear viewpoint, the features of an object become more pronounced and recognizable. By comparing the two viewpoints, the rear position become the standard for our data collection. From start to finish, we've generated five versions of our dataset with each subsequent iteration the number of images increases. As the number of images increases, our neural network has more to learn from.

Table 2. This table shows the number of images in the database of each iteration

Version	Number of Images
1	4,091
2	11,687
3	34,529
4	51,221
5	60,662

Custom Neural Network Construction

SimpleNet (Multilayer Perceptron)

Before constructing the custom neural network, we created a simple multilayer perceptron named SimpleNet to use as our foundation. First, we researched the caffe architecture of a neural network using an online resource created by Yangqing Jia and developed by Evan Shelhamer at University of California, Berkely. Then, we used the standard neural network, LeNet, as a reference model to create each layer of the multilayer perceptron.

Starting with the data layer, we used a batch size of 1024 for training data and 128 for validation data. Due to the size of our dataset, we determined that a batch size of 1024 and 128 was optimal for training a small network. Moving on, we have our first fully connected layer with 100 neurons. Within the hidden layers, we have the Sigmoid Activation function. The Sigmoid was an atribary activation function to test the functionality of the neural network. A second fully connected layer is added to support the capacity of computations that the neural network will need to perform. Here only 3 neurons are utilized. It then splits to the softMax layer to determine the expected loses in the training data, loss layer to measure inconsistencies between predicted values, and accuracy layer to measure how accurate the network is based on inference.

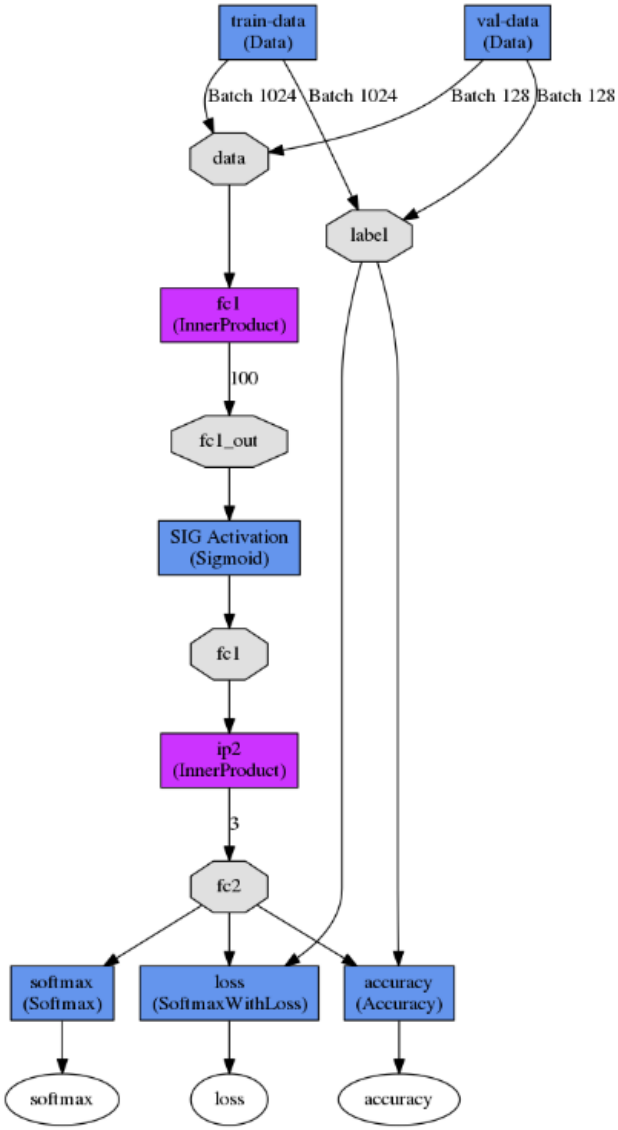


Figure 8. Diagram of SimpleNet (multilayer perceptron) using a Caffe architecture generated by Nvidia DIGITS

KeviNet (Custom Convolutional Neural Network)

The custom convolutional neural network named KeviNet is comprised of SimpleNet with a convolutional layer and pooling layer. Compared to the data input of SimpleNet, we adjusted the batch size of KeviNet to 64 and 32. A smaller batch size was chosen to control the accuracy of the error gradient, speed, and learning process of the neural network. For the convolutional layer, it had a kernel size of 10, stride size of 1, and zero padding of 0. Since input images were 320x240, a kernel size of 10 with a stride of 1 was deemed sufficient through testing. For the pooling layer, we decided to decrease the kernel size but increase the stride to be twice as fast. Additionally, ReLU was used instead of Sigmoid due to better optimization. All other components were equivalent to the multilayer perceptron.

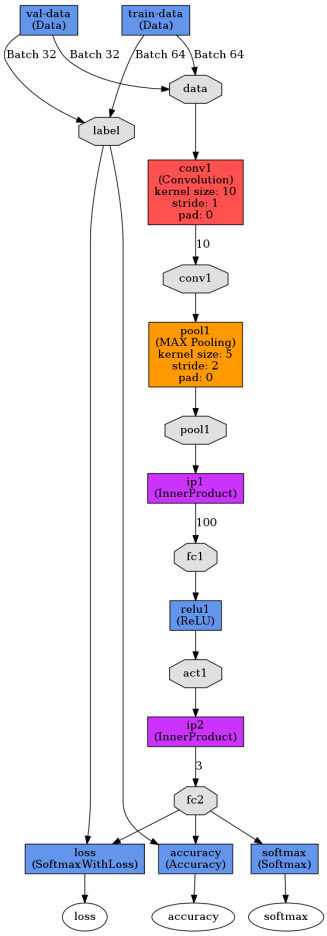


Figure 9. Diagram of KeviNet (custom convolutional neural network) using a Caffe architecture generated by Nvidia DIGITS

Field Testing

For field testing, we tested the accuracy and reliability of our dataset with real-time inference. We used four test subjects. Two subjects were controlled and part of the training data. The other two subjects were uncontrolled and wore clothing with colors that the neural network was unfamiliar with. The average height of the trained data was about 5’ 5’’ with mostly black, blue, grey, and white colored clothing

AlexNet	**Table 3. This table displays the real-time inference distance for the 5 feet class**					
	Height	Clothing	Trial 1	Trial 2	Trial 3	Average
	5’ 10’’ Controlled	White shirt Blue Jeans	4’ 5’’	5’ 8’’	6’ 2’’	5’ 5’’

	5’ 8’’ Controlled	Grey Shirt Black Jeans	4’ 8’’	4’ 6’’	4’ 8’’	4’ 7.3’’
	5’ 6’’ Uncontrolled	Black shirt Green shorts	4’ 4’’	4’ 0’’	4’ 5’’	4’ 3’’
	5’ 9’’ Uncontrolled	Blue Shirt Khaki Jeans	4’ 6’’	4’ 8’’	4’ 5’’	4’ 6.3’’

Simple Net	**Table 4. This table displays the real-time inference distance for the 5 feet class**					
	Height	Clothing	Trial 1	Trial 2	Trial 3	Average
	5’ 10’’ Controlled	White shirt Blue Jeans	4’ 9’’	4’ 9’’	4’ 9’’	4’ 9’’
	5’ 8’’ Controlled	Grey Shirt Black Jeans	4’ 8’’	4’ 9’’	4’ 7’’	4’ 8’’
	5’ 6’’ Uncontrolled	Black shirt Green shorts	3’ 9’’	4’ 9’’	3’ 3’’	3’ 11’’
	5’ 9’’ Uncontrolled	Blue Shirt Khaki Jeans	5’ 4’’	5’ 9’’	5’ 1’’	5’ 4.7’’

KeviNet	**Table 5. This table displays the real-time inference distance for the 5 feet class**					
	Height	Clothing	Trial 1	Trial 2	Trial 3	Average
	5’ 10’’ Controlled	White shirt Blue Jeans	4’ 3’’	4’ 9’’	4’ 11	4’ 7.7’’
	5’ 8’’ Controlled	Grey Shirt Black Jeans	4’ 5’’	4’ 4’’	4’ 6’’	4’ 5’’
	5’ 6’’ Uncontrolled	Black shirt Green shorts	3’ 0’’	3’ 2’’	3’ 7’’	3’ 3’’
	5’ 9’’ Uncontrolled	Blue Shirt Khaki Jeans	3’ 8’’	4’ 11’’	7’ 10’’	5’ 5.7’’

AlexNet	Table 6. This table displays the real-time inference distance for the 10 feet class					
	Height	Clothing	Trial 1	Trial 2	Trial 3	Average
	5’ 10’’ Controlled	White shirt Blue Jeans	13’ 4’’	13’ 5’’	11’ 11’’	12’ 10.7’’
	5’ 8’’ Controlled	Grey Shirt Black Jeans	8’ 10’’	8’ 11’’	8’ 5’’	8’ 8.7’’
	5’ 6’’ Uncontrolled	Black shirt Green shorts	9’ 1’’	12’ 7’’	10’ 9’’	10’ 9.7’’
	5’ 9’’ Uncontrolled	Blue Shirt Khaki Jeans	9’ 3’’	11’ 5’’	12’ 1’’	10’ 11’’

Simple Net	Table 7. This table displays the real-time inference distance for the 10 feet class					
	Height	Clothing	Trial 1	Trial 2	Trial 3	Average
	5’ 10’’ Controlled	White shirt Blue Jeans	9’ 5’’	9’ 2’’	9’ 10’’	9’ 5.7’’
	5’ 8’’ Controlled	Grey Shirt Black Jeans	9’ 5’’	10’ 0’’	9’ 7’’	9’ 8’’
	5’ 6’’ Uncontrolled	Black shirt Green shorts	8’ 7’’	9’ 11’	8’ 6’’	9’ 0’’
	5’ 9’’ Uncontrolled	Blue Shirt Khaki Jeans	8’ 9’’	9’ 0’’	9’ 2’’	8’ 11.7’’

KeviNet	Table 8. This table displays the real-time inference distance for the 10 feet class					
	Height	Clothing	Trial 1	Trial 2	Trial 3	Average
	5’ 10’’ Controlled	White shirt Blue Jeans	14’ 3’’	13’ 0’’	14’ 0’’	13’ 9’’
	5’ 8’’ Controlled	Grey Shirt Black Jeans	12’ 2’’	12’ 1’’	12’ 5’’	12’ 3’’

	5’ 6’’ Uncontrolled	Black shirt Green shorts	13’ 9’’	13’ 7’’	15’ 3’’	14’ 2.3’’
	5’ 9’’ Uncontrolled	Blue Shirt Khaki Jeans	12’ 5’’	13’ 3’’	14’ 1’’	13’ 3’’

Analysis and Results

Importance/Analysis of Datasets

Although there are pre-built data sets we can obtain from the internet, we wanted to create our own dataset to have a better understanding of the different factors and procedures required for creating a reliable dataset. This allowed us to understand any types of errors we may have encountered during the process of creating the dataset. By creating our own dataset we gain the ability to customize the data set or create one that will fit best for a specific task.

One factor that was detrimental to the performance of the neural network was the angle of the camera. Any amount of adjustment to the angle would significantly decrease the accuracy to the point of complete failure. To combat this, a fixture was built to keep the angle locked through data collection and field testing. Another factor was the subject’s body proportions. The larger the test subject, accuracy increased. The skinner the test subject, accuracy decreased. We fixed this issue by adding more variation of body types and heights into the dataset. The last factor was the environment. If the Jetson TX1 were to be deployed anywhere other than the environment it collected data from, exposure to new objects would be completely inaccurate. This is mainly due to the type of classification we used in our neural network. For our convolutional neural network, it was trained for image classification and not object classification. Due to time constraints, we decided to use image classification over object classification. Ideally, object classification would be able to accurately detect objects without environment restrictions.

Standard vs Custom

The three Neural Networks, AlexNet, KeviNet and SimpleNet, were trained and tested on inference DIGITS. AlexNet was the standard Convolutional Neural Network, SimpleNet was the MultiLayer perceptron and KeviNet was the custom Neural Network. When training the Neural Networks we used the same Dataset V5 which contained 90% training images, 10% validation images and 1% test images. When looking at theoretical performance SimpleNet, MLP, gave us the highest accuracy of about 98%. This theoretically means that when classifying the 3 different classes either empty hallway, person at 5 feet, and person at 10 feet it would have the highest confidence. AlexNet had the lowest accuracy of about 94% making it theoretically the least accurate at classifying the 3 different classes. KeviNet gave an accuracy of about 96%.

When field testing for accuracy SimpleNet was the least accurate. On average when classifying humans at 5 feet SimpleNet stopped at 4.212 feet which gave a STD of 1.552 feet. The average for 10 feet was 13.228 feet and the STD was 1.063 feet . It’s poor accuracy was due to it having the least parameters out of all 3 Neural Networks. AlexNet gave the best field testing results making it the most accurate. On average when classifying humans at 5 feet AlexNet stopped at 4.212 feet which gave a STD of .728 feet. The average for 10 feet was 10.805 feet and the STD was 2.053. In the middle ground was KeviNet which was our custom CNN. Its

performed fair as it didn't have as many parameters and layers as AlexNet but wasn't as simple as the MLP; SimpleNet. On average when classifying humans at 5 feet KeviNet stopped at 4.657 feet which gave a STD of .753 feet. The average for 10 feet was 9.398 feet and the STD was .555.

Overall when considering field testing, theoretical performance, and actual performance, our custom Neural Network, KeviNet, performed best at classifying humans at 5 feet and 10 feet. When field testing from beginning to end KeviNet took about 25 seconds to complete 1 cycle this was because it had a moderate overhead. Which was faster than AlexNet which would often stutter due to it's large overhead that created bottlenecking in the Jetson TX1 causing it to take about 35 seconds to complete 1 cycle. SimpleNet was the quickest due to its small overhead giving it a real time inference time of 21 seconds to complete one cycle. SimpleNet would often have a hard time detecting humans at 5 feet.

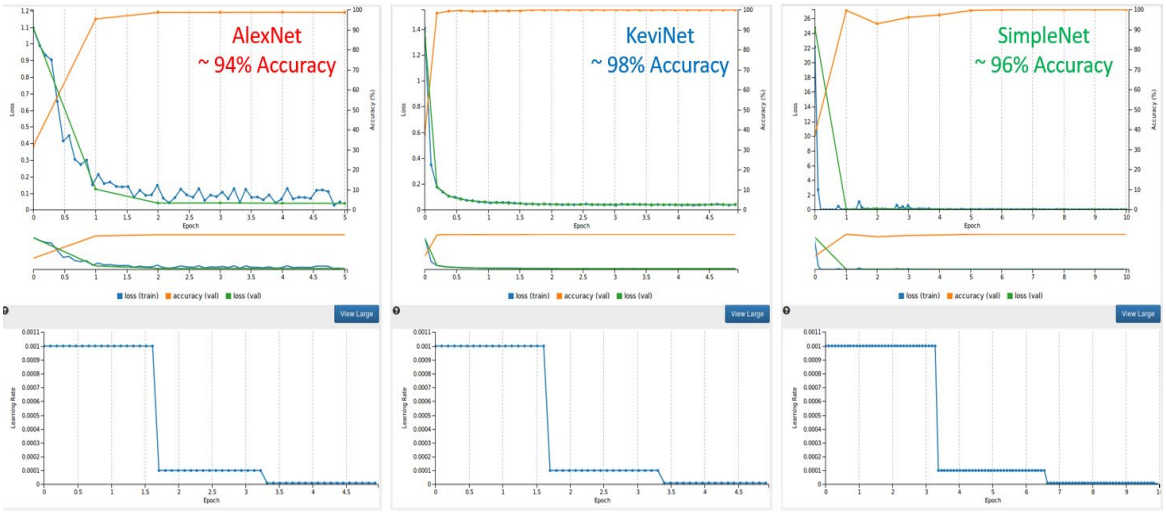


Figure 10: The top graphs indicate the accuracy of each Neural Networks at different Epochs. The bottom graphs indicate the learning rate decay of each Neural Network with respect to Epoch iteration.

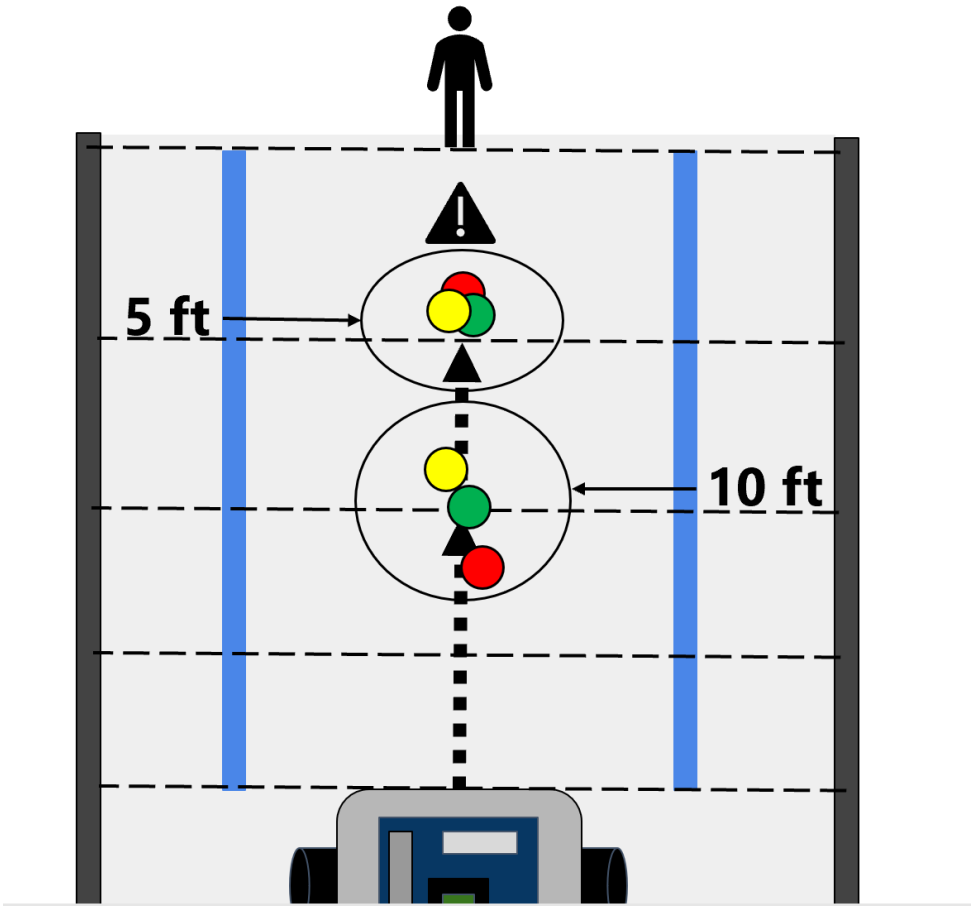


Figure 11: This was the environment set up for the Custom Neural Network vs Standard Neural Network. At one end sat the autonomous Jetson TX1 and at the other end a human subject stood.

Neural Network	5ft averages	10ft averages	5ft STD	10ft STD
AlexNet	4.694	10.805	0.728	2.053
KeviNet	4.657	9.398	0.753	0.555
SimpleNet	4.212	13.388	1.552	1.063

Table 9: This data was measured in feet. It contains the average real time inference for 5 feet and 10 feet from 2 know subjects to our dataset and 2 unknown subjects..

Conclusion

In order to make this project possible 5 goals were created. Goal 1: Build and train custom dataset, Goal 2: Deploy and optimize a standard neural network on the Jetson TX1, Goal 3: Deploy and optimize parameters autonomous field testing, Goal 4: Enlarge custom dataset with

more environmental variation, and Goal 5: Create and compare custom Neural Networks with a standard Neural Network for autonomous driving. We were able to reach 5 out 5 of our goals.

Future work

As of now the Nvidia Jetson TX1 only has one camera. This that the computer only sees in the 2 Dimension making it hard for the autonomous TX1 robot to analyze depth when looking for people at 10 feet and 5 feet. There are a few techniques to overcome this one of them is by adding a second camera to the autonomous robot. This second camera can be used as a reference point and by using 3D software could triangulate the object. The second technique that can be applied to tell depth in a 2D picture can be by using a camera with lenses. This single camera can then take a picture from the same position at two different depths. The last but not final technique that could be to implement would be adding infrared distance sensor. This could work in conjunction with the camera in order to validate information being feed.

Further work needs to be done to the custom Neural Network in order to make it more robust. By adding and taking layers away we could make it more efficient at object detection. Increasing datasets to include different environments would add in variation. Our datasets were collected at different points of a hallway at SFSU and this was due to the Jetson TX1 needing to have internet connection in order to operate. By making the Jetson Tx1 able to operate without internet connection we could then expose the autonomous vehicle to different environments overall making our entire system more robust.

Literature Review

Using deep learning to investigate the neuroimaging correlates of psychiatric and neurological disorders: Methods and applications

In the past two decades, neuroimaging studies of psychiatric and neurological patients have relied on mass-univariate analytical techniques. By utilizing Deep Learning, the neuroimaging community can develop algorithms that discovers trends and patterns in existing data and use the information to make predictions on new data. Deep Learning methods are statistical inferences at a single subject level, meaning that it can be used to inform diagnostic and prognostic decisions of individual patients, overcoming the limitations of previous techniques. DL techniques such as Support Vector Machine works by estimating an optimal hyperplane that separates two classes. When they are not linearly separable, SVM uses kernels that maps the original data to make them linearly separable. Despite being in its initial stages, the application of DL in neuroimaging has shown promising results for fundamental advances in searching for imaging-based biomarkers of psychiatric and neurologic disorders. Due to the complexity of DL models, many studies must favor sample sizes of larger quantities to train the machine to adapt to newer information.

Small Neural Nets Are Beautiful: Enabling Embedded Systems with Small Deep-Neural-Network Architectures

When creating a custom Neural Network our main concern was making it with the fewest amount of parameters possible while maintaining the highest accuracy. To accomplish this computation overhead had to be taken into consideration. The inspiration to make a small but precise CNN came from the research paper *Small Neural Nets Are Beautiful* by Forrest Iandola and Kurt Keutzer. At the UC Berkeley Lab they created a Deep Neural Network called SqueezeNet. SqueezeNet requires as little as 480KB of storage for its parameters. AlexNet which is a popular Convolutional Neural Network uses about 60 million parameters that require 240MB, and in comparison SqueezeNet holds about 1.2 million and require about 480KB which is a 50x size reduction in parameters. SqueezeNet's smaller overhead means that the Deep Neural Network could be trained much faster and would require less computational overhead to operate making it more efficient. A small Deep Neural Network is more practical to use when working in collaboration with the software Computer Vision and embedded systems that have constraints. Embedded systems that have constraints such as lack of computational power could potentially lead to bottlenecking in the system. Less parameters means less updates the system has to periodically change overall making it easier on low power computer such as the Nvidia Jetson TX1 that we were using to control the autonomous vehicle.

High performance vegetable classification from images based on AlexNet deep learning model

With China being one of the largest vegetable producers in the world, there is a high demand in the creation of a machine that can pick vegetables autonomously and decrease the amount of manual operation and reduce the amount of labor force. The first step into creating this robot is by creating one that can identify and classify vegetables individually. With the help of the Caffe deep learning framework, and the AlexNet neural network, it is possible to create and train data sets of different vegetables that will give the machine an ability of identifying vegetables based on the data sets. With AlexNet, there is a higher accuracy than that of the BP neural network with BP having 78% accuracy and AlexNet having 92.1%.

GPU-Assisted Learning on an Autonomous Marine Robot for Vision-Based Navigation and Image Understanding

A GPU-based integrated robotic system incorporated on a single underwater vehicle capable of collision avoidance, navigation, and image understanding. The unit deploys observational tasks such as coral reef health assessments using operations of multiple image analyzation tasks in close proximity areas. The device also must navigate through these tight fit areas utilizing object detection and collision avoidance. By having GPU integration the device is capable of leveraging deep neural networks for each of the automated tasks. The CPU processes images in order to perform visual Simultaneous Localization and Mapping (SLAM for short). The state-of-the-art marine applications can leverage deep networks for sensor processing, while in this work the focus on image processing using Deep Convolution Neural Networks for environmental assessments. Having a DCNN for data analysis and measurement solutions can have a major advantage in both performance and system efficiency over classical methods.

However, a huge disadvantage of using machine learning as the primary sensing modality is carrying out computer vision underwater, and the computational costs that come with it.

Deep Learning

Deep Learning is a form of machine learning that can identify objects and classify images. It is used in most aspects of modern society and has been a big factor in the world of Artificial Intelligence. With the help of the machine learning form called Supervised Learning, we are able to detect objects such as cars, houses, cats, dogs, or a person. It is a form of machine learning that consists of datasets with images of the object we wish to be detected. With the help of convolutional neural networks, we are able to process the data which can be trained and tested for evaluation. Deep learning has a bright future ahead of it with the amounts of data being collected and shared which will improve artificial intelligence.

Yann Lecun, Yoshua Bengio, Geoffrey Hinton, "Deep Learning"

Bibliography

Retrieved from https://ieeexplore.ieee.org/abstract/document/8604645

Retrieved from http://deeplearning.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/

/@_sumitsaha_. (2018, December 17). A Comprehensive Guide to Convolutional Neural Networks - the ELI5 way. Retrieved from https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Caffe. Retrieved from https://caffe.berkeleyvision.org/

Iandola, F., & Keutzer, K. (2017). Small neural nets are beautiful. *Proceedings of the Twelfth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis Companion - CODES 17*. doi: 10.1145/3125502.3125606

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks fromOverfitting. *Journal of Machine Learning Research 1*.

(2017, January 10). Using deep learning to investigate the neuroimaging correlates of psychiatric and neurological disorders: Methods and applications. Retrieved from https://www.sciencedirect.com/science/article/pii/S0149763416305176

Zhu, L., Li, Z., Li, C., Wu, J., & Yue, J. High performance vegetable classification from images based on AlexNet deep learning model. Retrieved from https://ijabe.org/index.php/ijabe/article/view/2690