Convolutional Neural Network – Drive Through Maze

Jesse Simpson
Department of Computer Science
Missouri State University
Springfield, MO USA
simpson145@live.missouristate.du

Rohan Saha
Department of Computer Science
Missouri State University
Springfield, MO USA
rohan2728@live.missouristate.edu

Vignesh Sivanandha Rao
Department of Computer Science
Missouri State University
Springfield, MO USA
vignesh1709@live.missouristate.edu

Abstract –**In this project report, we construct a differential drive robot which can traverse a ROS simulated maze. It uses Convoluted Neural Network (CNN) to classify the maze into left, middle and right classifications. Then using these classifications probabilities traverses to the center of the maze.**

Keywords— Convolution Neural Network (CNN), ROS, classification.

I. Introduction

Image classification is the task of taking an input image and outputting a class (a cat, dog, etc.) or a probability of classes that best describes the image. For humans, this task of recognition is one of the first skills we learn from the moment we are born and it comes naturally and effortlessly as adults. Without even thinking twice, we're able to quickly and seamlessly identify the environment we are in as well as the objects that surround us. When we see an image or just when we look at the world around us, most of the time we are able to immediately characterize the scene and give each object a label, all without even consciously noticing. These skills of being able to quickly recognize patterns, generalize from prior knowledge and adapt to different image environments. But for a computer to classify any image, it takes an image as a matrix of image width and height with its pixel values. The major challenge is to extract features from this image data which allows the computer to distinguish similar object or features in different images. We use Convolution Neural Network (CNN), to generate a model, which can distinguish the maze environment set up in a Robot Operating System (ROS). We also extract the probability values out of the correctly classified images and use these probabilities to drive our differential drive robot to drive through the maze.

II. Convolution

Natural images have the property of being 'stationary', meaning that the statistics of one part of the image are the same as any other part. This suggests that the features that we learn at one part of the image can also be applied to other parts of the image, and we can use the same features at all locations. More precisely, having learned features over small (say 8x8) patches sampled randomly from the larger image, we can then apply this learned 8x8 feature detector anywhere in the image. Specifically, we can take the learned 8x8 features and 'convolve' them with the larger image, thus obtaining a different feature activation value at each location in the image. Once we have the convolved features that are useful in one region are also likely to be useful for other regions. Thus, to describe a large image, one natural approach is to aggregate statistics of these features at various locations. For example, one could compute the mean (or max) value of a particular feature over a region of the image. These summary statistics are much lower in dimension (compared to using all of the extracted features) and can also improve results (less over-fitting). This aggregation operation is called 'pooling', or sometimes 'mean pooling' or 'max pooling'. [1]

III. Convolution Neural Network

A Convolutional Neural Network (CNN) is comprised of one or more convolutional layers and then followed by one or more fully connected layers as in a standard multilayer neural network. The architecture of a CNN is designed to take advantage of the 2D structure of an input image. This is achieved with local connections and tied weights followed by some form of pooling which results in translation invariant features. Another benefit of CNNs is that they are easier to train and have many fewer parameters than fully connected networks with the same number of hidden units.

CNN consists of a number of convolutional and subsampling layers optionally followed by fully connected layers. The input to a convolutional layer is an m x m x r image where m is the height and width of the image and r is the number of channels, e.g. an RGB image has r=3. The convolutional layer will have k filters (or kernels) of size n x n x q where n is smaller than the dimension of the image and q can either be the same as the number of channels r or smaller and may vary for each kernel. The size of the filters gives rise to the locally connected structure which are each convolved with the image to produce k feature maps of size m−n+1. Each map is then subsampled typically with mean or max pooling over p x p contiguous regions where p ranges between 2 for small images (e.g. MNIST) and is usually not more than 5 for larger inputs. Either before or after the subsampling layer an additive bias and sigmoidal nonlinearity is applied to each feature map. The figure below illustrates a full layer in a CNN consisting of convolutional and subsampling sublayers. Units of the same color have tied weights. [2]

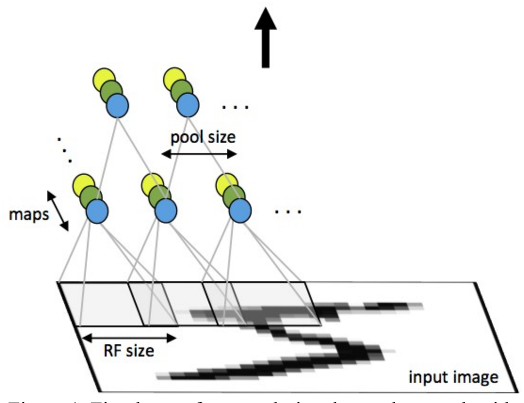

Figure 1. First layer of a convolutional neural network with pooling. Units of the same color have tied weights and units of different color represent different filter maps.

Finally, we use Backpropagation algorithm to propagate the error through the pooling layer by calculating the error with respect to each unit incoming to the pooling layer

IV. ResNet Architecture

According to the universal approximation theorem, given enough capacity, we know that a feedforward network with a single layer is sufficient to represent any function. However, the layer might be massive and the network is prone to overfitting the data. Therefore increasing network depth does not work by simply stacking layers together. Deep networks are hard to train because of the vanishing gradient problem—as the gradient is back-propagated to earlier layers, repeated multiplication may make the gradient infinitely small. As a result, as the network goes deeper, its performance gets saturated or even starts degrading rapidly. The core idea of ResNet is introducing of "identity shortcut connection" that skips one or more layers [3].

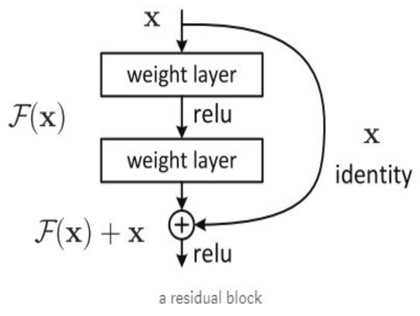
x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu
a residual block

Figure 2. Identity shortcut connection

The difference between ResNet and traditional neural nets architecture is that. Traditional neural nets will learn directly from the output whereas ResNet models the layers to learn the residual of input and output of subnetworks. This gives an option to skip subnetworks by making the output a particular subnetwork into the output of the last subnetwork. Therefore, during backpropagation ResNet can choose to ignore the gradient of some subnetworks and just forward the gradient from higher layers to lower layers without any modification. [4]

V. Experiments and Results

A. Maze Generation

The proposed experimental goal is to navigate a simulated differential drive robot in a square maze created in a gazebo environment. We broke down the experiment process into four steps. In order to create our various testing and training data, a gazebo world was generated. This world is a square-shaped maze that follows a clockwise path towards the center. This maze was constructed using three objects: orange barrier, white barrier, and blue cylinders for corners. Below is Figure 3 which is an aerial view of the generated world used in the training, testing, and for the demonstration.

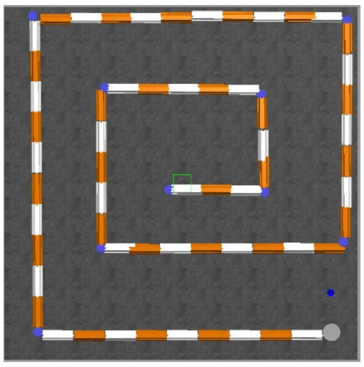

Figure 3. Gazebo Maze Map

B. Testing and Training Data

The generated map in gazebo was then used to generate testing and training data. A movement script was written using a rosnode. This node took user input from the keyboard, allowing the inputs: 'w', 'a', 's', 'd' in order to control the linear X and angular Z values of the differential drive robot. By using the generated maze and movement script, the training and testing data collection was performed. This process involved the user manually controlling the robot through the course. The training data is gathered from three separate sources. Traversing the maze following the left wall, center road, and the right wall are the sources used. As the differential drive robot travels down each source, using rqt image view, the camera data was saved and properly labeled according to the direction the robot should drive to progress through the maze.

C. Building Fastai Model

The process of constructing and training a convolutional neural model from scratch would take an unnecessary amount of time. In order to build a reliable model from our data, we decided to use fastai. Fastai is a library of tools that can be used in order to construct a convolutional neural network to perform image classification. For the purpose of our experiment, we needed to construct a model that had a high accuracy, within the range of 95 to 99 percent. To achieve this goal, experimentation was conducted on the following parameters: learning rate, number of epochs, and network architectures. The first set up used a learning rate of 0.01, 1000 epochs, and two different architectures: ResNet34 and ResNet50. Table 1, shows the accuracy score for different ResNet architecture.

TABLE I. Accuracy score for different ResNet Architecture

Architecture	Learning Rate	Epoch's	Accuracy	Trained Images
resnet34	0.01	1000	90.63%	120
resnet50	0.01	1000	89.06%	120

The accuracy achieved in the two models were decent. The ResNet34 architecture performed approximately 1.5% better than ResNet50; however, the overall accuracy was not at the optimal percentage. A proposed modification was made to see the effects of changing learning rate and the number of epochs ran. The second set up used a learning rate of 0.02, 500 epochs, and ResNet34/50. Table 2, shows the result of accuracy score on a different learning rate.

TABLE II. ACCURACY SCORE FOR DIFFERENT RESNET ARCHITECTURE AND LEARNING RATE

Architecture	Learning Rate	Epoch's	Accuracy	Trained Images
resnet34	0.02	500	90.62%	120
resnet50	0.02	500	88.28%	120

Comparing the first and second experiment results, by altering the learning rate and epochs, the overall accuracy of the model did not improve. The promising information gained from the secondary set up showed that using half as many epochs compared to the first experiment. Applying the information gained from the previous two experiments, the third set up used a learning rate of 0.025, 1000 epochs, and ResNet34/50. Table 3, the experiment set up and results of accuracy rate on increasing learning rate to 0.025.

TABLE III. ACCURACY SCORE FOR DIFFERENT RESNET ARCHITECTURE AND LEARNING RATE

Architecture	Learning Rate	Epoch's	Accuracy	Trained Images
resnet34	0.025	1000	96.09%	120
resnet50	0.025	1000	91.40%	120

The results of the third experiment provided us with a satisfactory model using the resnet34 architecture at 96.09% accuracy. Before we concluded the third experiment model as the primary project classifying model, we decided to do a slight modification to the learning rate to observe any accuracy changes. Applying the previous experiment results, the final set up used a learning rate of 0.03, 200 epochs, and ResNet34/50. Table 4, shows the experiment set up and results of accuracy rate on increasing learning rate to 0.03.

TABLE IV. ACCURACY SCORE FOR DIFFERENT RESNET ARCHITECTURE AND LEARNING RATE

Architecture	Learning Rate	Epoch's	Accuracy	Trained Images
resnet34	0.03	200	96.31%	120
resnet50	0.03	200	87.50%	120

The final experiment set up peaked our model accuracy at 96.31%. We noted that accuracy would increase as the learning rate was modified to a slightly larger value. Epoch value rarely provided a substantial increase or decrease in accuracy when the model ran more than 200 epochs. Therefore, the simulation test will be using the fourth model trained with the ResNet34 architecture with more than 95% accuracy.

D. ROS and Python

The last requirement for the experimental set up is to create the python script and ROS environment. This task was done using two separate files. The first file is a python script that is called within an anaconda3 environment. Fastai requires python 3; however, ROS packages used for the experiment required python 2.7. Within this python script, all fastai modules are loaded and the fourth model is loaded. This will allow the script to classify incoming images saved onto the hard drive by ROS. The second file is our rosnode that is used to subscribe to the differential drive robot's camera data, as well as publish movement commands based on the camera's data classification. These two files are used to control the robot to the center of the maze. Below in Figure 4 is an rqt graph, showing the relationship of our rosnode and gazebo.

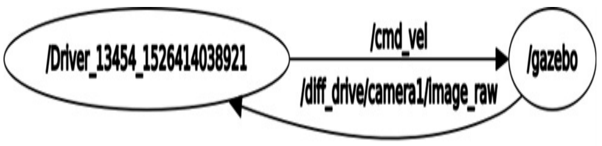

Figure 2. Identity shortcut connection

VI. CONCLUSION

Using the trained model obtained from fastai, the simulated differential drive robot was then operated in the gazebo environment. By using a convolutional neural network to classify the directions the robot traverses, it could successfully traverse the maze and arrive at the center point. The rosnode file used the classification probabilities and performed movement actions based on those probabilities. This required manual modification of values in order to allow the robot to successfully complete the maze. We used the trial and error method to find a semi-optimal linear and angular speed. Despite, the robot successfully traversing the maze with the given speed, it can be improved to traverse it faster and safer. This leads to several possible expansions and improvements in future work. First, the use of another image processing algorithm can be implemented to assist the classifier, such as semantic analysis. Second, using the process of trial and error, the implementation of a neural network can be used to find correct values for robot speed, weights on the probability values, to ensure the highest accuracy of maze traversal.

REFERENCES

[1] http://ufldl.stanford.edu/tutorial/supervised/Pooling/

[2] http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/

[3] https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035

[4] https://wiseodd.github.io/techblog/2016/10/13/residual-net