CSCI 374: MIRA

Brandon Banda, Quinn Barker-Plummer

December 15, 2017

1 Introduction

Our motivation for pursuing this project was to gain a deeper understanding of how neural networks function, and to gain practical skills on the cutting edge of machine learning to help us in our careers as computer scientists. Our project aimed to create a convolutional neural network that processed a large dataset of images, and made predictions on a randomly selected subset of images from the dataset. Our report contains an analysis of the problems involved in image recognition, a detailed description of our own solution to this issue, a summary of how our implementation was able to solve the issue (including image pre-processing and randomization), and an analysis of the results that we attained from our neural network. Overall, our neural network performed better than we had expected, with accuracies on predictions of the random subset of images ranging from 60-63%, with relatively low computation times for thousands of training runs. We also have ideas on how to improve on the performance of our neural network in the future.

2 Problem

MIRA was designed to address the problem of intelligent image recognition, which has been a topic of discussion in the field of computer science for decades. It is only in recent years that even the most advanced algorithms, such as those used by Google and Facebook, have shown accuracy comparable to basic human recognition. Recently, Google reported a facial recognition accuracy of 86%, and a general recognition of 94% with a neural networking model that attempted to generate image labels in natural language. Facebook has also reported a 98% facial recognition accuracy, beating the FBI's own facial recognition by 13%. These are promising results, especially for a problem with such far reaching applications in everything from astrophysics, to self-driving cars, to accessibility for the visually impaired.

Despite this, "the performance of machines in providing rich semantic descriptions of natural scenes from digital images remains highly limited and hugely inferior to that of humans" (Fleuret). However, in "Model-based Machine Learning" by Christopher Bishop, Bishop shows how making a model for image recognition can result in dramatically improved performance. Specifically looking at the Xbox Kinect, he states that "the parameters of the system, in this case, the selected features and thresholds at the nodes of the decision trees, as well as the depths of the trees themselves, are determined in the laboratory during the training phase" (Bishop). Using this model, the Kinect is able to track the human body with amazing accuracy, but only due to "[millions of] depth images of human body poses, each of which is labelled with body parts" (Bishop). As such, it models the incoming images in segments, which, Bishop claims, improves the results. Fleuret agrees, claiming that model-free machine learning in image recognition performs poorly, on average, compared to human performance. In short, image recognition requires a model, usually involving breaking images into segments the same way that humans do in the visual field.

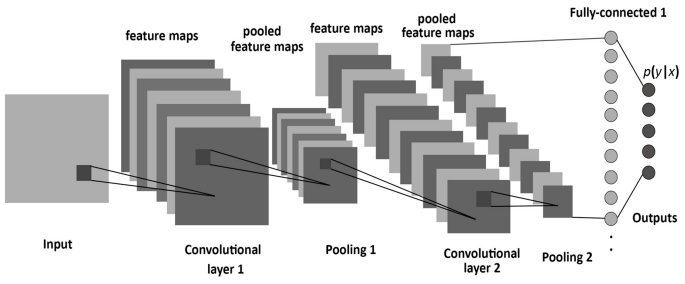
For this reason, we modeled the problem using a convolutional neural network. These have shown promise in the past for image segmentation and recognition due to the ability to break images down into smaller sections and analyze the sections individually. We also used other tools such as NumPy and Pillow to help process the images. (More discussion of this in the Solution section.)

The data set used was the CalTech-256 Data Set (https://authors.library.caltech.edu/7694/), a standard data set of images used to test image recogni-

tion models. It contains roughly 100 images for each of 256 different labels, such as "airplane," "american flag," and "eiffel tower". From these, we selected 10 labels to use as our limited range data set. We maintained the full RGB color of the images, but scaled each image down to 32×32 in size, to limit the amount of the incoming data. (More discussion of this in the Experimental Setup section.)

3 Solution

Figure 1: An example of a convolutional neural network, as applied to image recognition.
Image: mdpi.com



In order to solve the problem of image recognition, we decided to use a pre-existing open-source neural net software called Tensorflow to help us in developing a convolutional neural net. Tensorflow and its libraries implement all of the algorithms such as backpropagation, naïve gradient descent, and image segmentation, all of which we needed to create our convolutional neural net. We also used Pillow, a Python Imaging Library branch that is compatible Mac OS and newer versions of Python 3, to resize and transform the images in our dataset. We then used NumPy to vectorize our images and data labels for use in our convolutional neural net. Our implementation comprised of image pre-processing and vectoring, splitting of image datasets into a training and testing set, and finally feeding the training set into the neural network, and evaluating the testing set at the end of training. We manually implemented the image pre-processing ourselves using Pillow. We tried out a range of implementations such as scaling images up to a large square and grayscaling them, scaling images down and retaining RGB to reduce computation time and retain key image data, and even scaling images up to the most average-sized, most square image in our dataset to keep image distortion to a minimum. We eventually settled on downscaling the images, but retaining their RGB data, as this reduced computation time and gave good results. We then used NumPy to vectorize the images we had transformed in our image pre-processing. This converted PIL image objects into arrays of discretized pixels for use in our neural network. We then created a class which took in a vectorized image, a one-hot encoded label for the image, and the original label of the image for readability of our results. To create our convolutional neural net, we used a pre-existing template which was provided to us by Tensorflow's documentation website. We modified the dimensions used by their template to take into account our vectorized RGB images, re-implemented the way that labels were one-hot encoded in their template, and modified several parameters, especially in the pooling and dense layers of the neural network, which determine how segmented and 'convoluted' the original image will be by the time it is processed by the neurons of the neural net. We decided to have 2 convolutional layers to theoretically allow for our neural net to identify shapes within our images for more accurate predictions. We tweaked these parameters throughout our process until we were getting relatively high accuracy scores (above 50%) for our testing set, and stopped tweaking once we reached past 60% accuracy in our results. We do believe that more improvements could be made to the parameters of the neural net (more information in Results & Analysis). We will explain our parameter choices, as well as which optimizations we made to our neural network in the Experimental Setup section.

4 Experimental Setup

We evaluated our model primarily by its classification accuracy on the data set, for a set number of iterations. Starting with a set 1000 iterations (about 5 minutes of runtime), we modified the learning rate, the segmentation density (the number of segments per image), as well as the size and overlap of the segments. In addition, we modified the drop rate to prevent overfitting, or fitting to bad or outlying data, with varying success. Because the drop rate determines what proportion of the farthest outlying data is ignored by the model, it is difficult to assign a single drop rate that both creates a good learning

model and does not cause overfitting, so some experimentation was required to improve the value. This issue may be helped by using a larger, more varied data set, because we could then set the drop rate high, and let the size of the data set itself prevent overfitting.

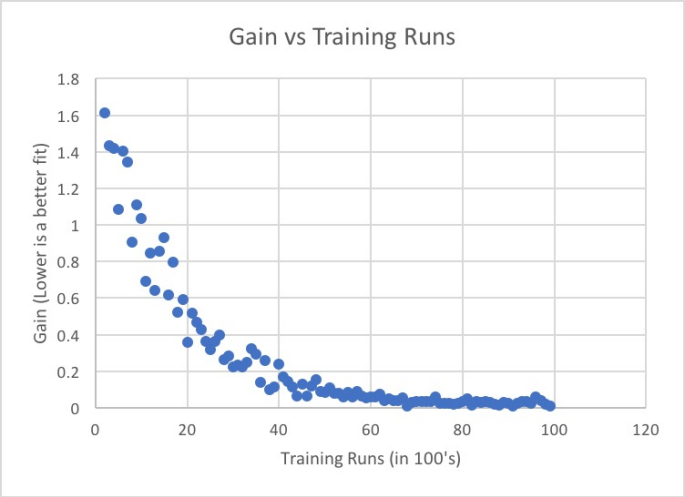
It is also possible that we could improve the model by performing fewer transformations on the images. For the purposes of this model, we scaled the images down to 32×32 in order to limit the data inflow to a manageable level. We also deliberately chose labels that were the most visually distinct, to give the network the largest number of distinguishing features between labels. We could have opted to grayscale the images as well, but chose not to as it would represent significant data loss. For certain labels, such as "american flag," color is the primary identifying feature, and removing it would significantly impact the accuracy of the network. We also used anti-aliasing for the same reason, to preserve image quality as much as possible while scaling. In future, we could modify the network to scale images less or not at all, but that would require much longer runtime and larger computational power.

5 Results

After passing our image dataset through our neural network several times with different random seeds to determine the training and testing sets, we determined that our neural net has an average accuracy of around 60-63% within a 95% confidence interval. This confidence interval only applies for 1000-iteration training runs, as higher training iteration runs yielded different results. Figure 2 represents the gain over 10,000 training iterations for our neural network using the random seed 1234, and segmentation and step numbers of 2. We used 70% of the dataset as training images, and the rest as testing images. Our learning rate in this training run was 0.1%.

Figure 2 shows that our neural network beings to fit the data in our training set extremely well over time, learning the intricacies of each image with a relatively high accuracy. However, with an accuracy of only 58.24%, this graph serves to show that our neural network tends to over-fit the training set the longer it runs. This overfitting is something that we believe has arisen as a result of our decision to

Figure 2: A graph of Gain vs Training Runs, for an average of 58.24% accuracy over 10.000 iterations.



make our images 32×32 in dimensions, as the data in such small images may not catch on the more intricate details of what the original image contained. Thus, there is a hard limit on how accurate our neural network can be while working with the small image sizes, which is tradeoff that we were willing to make to spare ludicrous computation time. Furthermore, this overfitting may also be a result of the limited number of convolutions that can be made using such a small image. A 3rd layer (something that we attempted to make) failed in implementation, as convolutions on the small segments from our second layer introduced a significant amount of noise to neural net, dropping our accuracy to below 30%. Another interesting note about our neural network is that the learning rate did not significantly affect our accuracy results, which suggests that our image dimensions are small enough that a neural network can learn the intricacies of them in a relatively short amount of time. In recognizing that the image dimensions are of key importance in our neural network, we may try to increase the dimensions of each image in the future to allow for a 3rd layer in our neural network, as well as allow for more neurons in our network. We may also want to split our training for the neural net across several systems to reduce the computation time using a supercomputer, an array of computers/processors, or a graphics card to speed up matrix operations. Furthermore, a way to possibly improve on our results is to have a dataset comprised of purely square-dimensional images, as to reduce the distortion and inaccuracies in images that needed to be scaled disproportionately both ver-

tically and horizontally. Such square images might better retain key information about the image itself, even when scaled to a small dimension such as 32×32, allowing for resizing to have less of an effect on our results, while maintaining a manageable computation time for our images.

6 Works Cited

Francois Fleuret, Ting Li, Charles Dubout, Emma K. Wampler, Steven Yantis, and Donald Geman. "Comparing machines and humans on a visual categorization test." PNAS 2011 108 (43) 17621-17625; published ahead of print October 17, 2011, doi:10.1073/pnas.1109168108

"Model-based machine learning," Christopher M. Bishop. Phil. Trans. R. Soc. A 2013 371 20120222; DOI: 10.1098/rsta.2012.0222. Published 31 December 2012