Playing Card recognition using Convolutional Neural Network

Gregory Poisson, Bradley Winters and **Joshua O'Dell**
Colorado State University
Fort Collins, CO 80523

Abstract

One of the most important technological advancements in the last few years has been the availability of image recognition algorithms and libraries. This paper will explore an image recognition system which utilizes a convolutional neural networks to recognize images of playing cards. We discuss the use of such a neural network, and the issues that arose as we began to implement it. We will touch on the problems that we solved, as well as the problems for which a solution was not found. Finally, we will give a summary of our experience, discuss the current efficacy of the technology, and remark upon different applications of our results.

Introduction

In this paper we aim to implement a playing card image recognition system. This system could have numerous uses, allowing for automated players, or assistants, to identify and potentially interact with cards on the table. In this way, computerized opponents might be made which could play more than simple graphical card games, but also physical ones using paper cards. The image recognition industry has progressed quickly, there are several tools available for doing image recognition. Our paper will focus on a neural network with convolution layers.

Methods

To convey exactly how this system is designed, we present a general description of the implementation we used.

Homemade Neural Network

We started the study with a handwritten neural network class. It was a simple class written in Python with the ability to create convolutional layers. The results of using this class were unsatasfactory, however. The training time took days to train on anything with more then a single convolutional layer. We therefore decided to look at other commercial and open source libraries which had been developed by companies with far more resources then our team.

TensorFlow

One of the libraries that first came up was Google's TensorFlow library. This library is freely available and has the ability to produce simple, easily understand assets. We settled on using this library. Tensorflow features a pretrained model named inception V3. This model has been trained with 1.2 million images representing 1000 classes. Fully training an image classifier on thousands of images would have taken weeks or months. We opted instead to retrain the final 2 layers of the inception V3 model for playing cards. These layers consist of a bottleneck or autoencoder layer and a final classifcation layer. In essence, the classifier is identifying playing cards with features learned from other images.

Training Data

To train our neural network, we needed images of all 52 playing cards in a standard deck of playing cards. We started by manually taking approximately 20 photographs of each playing card in a deck, then placing them in separate folders with a naming convention that TensorFlow could use. We then copied and manipulated these images, skewing, rotating, and cropping them using an automated script. Our dataset in total contained approximately 6,500 images.

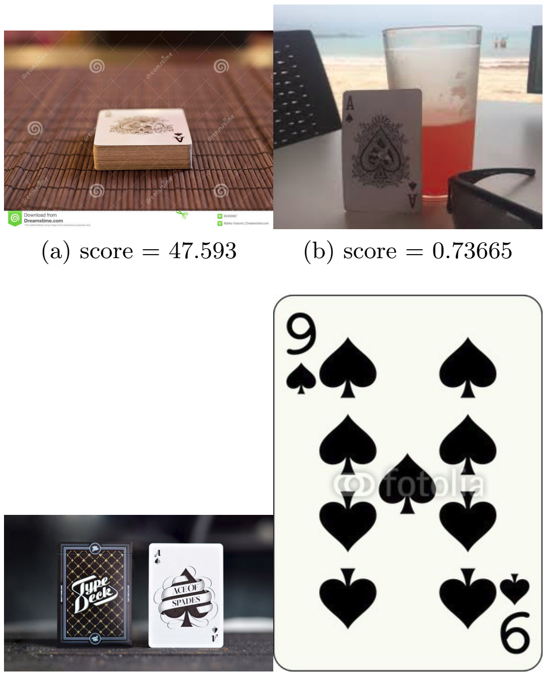
Initial tests

We began by training our neural network on approximately 1,000 images, just to see how accurate our initial predictions would be. As shown in Figure 1, results varied for cards with complex backgrounds, but the model was confident about cards with little to no background image data.

We hypothesized that the network needed more training data data, and more training steps. 1,000 images was not a large dataset, and the fact that we were seeing results already was promising.

Refining the Training

With the success of the initial tests we began to train the network in ernest. We supplemented the pho-



(a) score = 47.593 (b) score = 0.73665

(c) score = 0.45144 (d) score = 0.91995

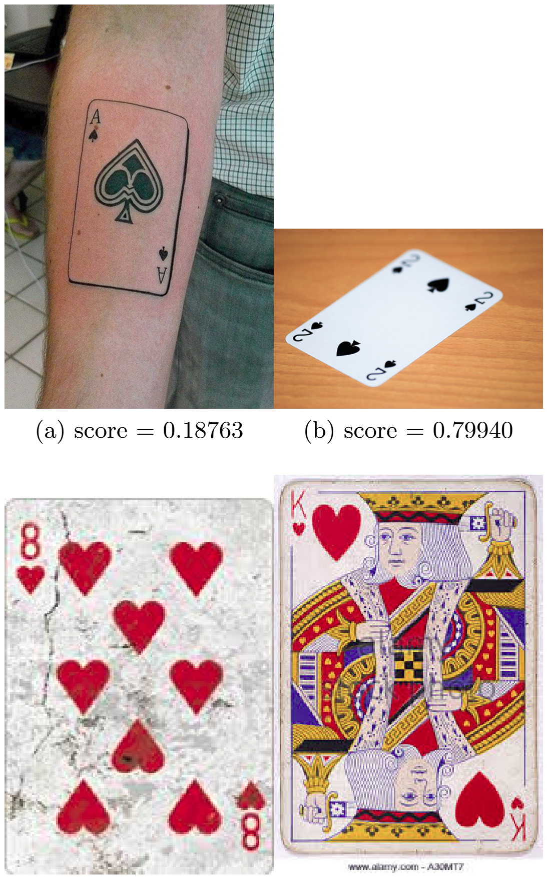
Figure 1: Correctly classified images with class probabilities; initial classifier

tographs in our dataset with card images obtained from Google. We added 8 such images for each card (class), then trained the network on a set of 1,800 training images using 1,600 training steps. This training took several hours, but the results were even better. The network was now able to more accurately classify images with various backgrounds. Figure 2 shows some examples of the images with cards that were now correctly classified, as well as examples of the training images without backgrounds, which greatly refined our model.

OpenCV

One of our initial goals was to be able to recognize as many situations as we could, including cards that were not oriented in a way that our network was trained to handle. We also wanted to be able to recognize multiple cards within the same image. What we found was that our network was able to recognize cards oriented horizontally or vertically, it had a hard to with rotated cards, skewed cards, and was terrible with multiple cards on the same image.

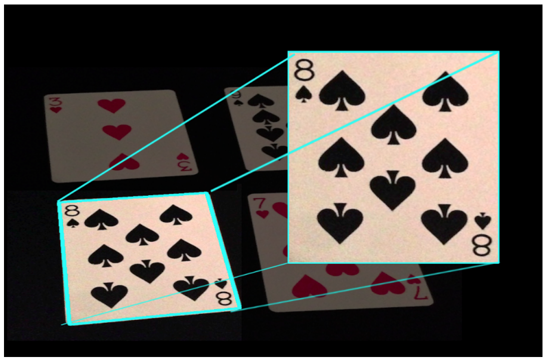
Our solution was to pre process the images. First the cards are isolated in the image, and then they are transformed to be flat rectangular images, which are then classified through the trained network. We used the OpenCV library to locate and transform the card into a clean, flat piece of data prior to classification. Figure 3 shows an example of pre processing and pulling out the playing card portion of the image.



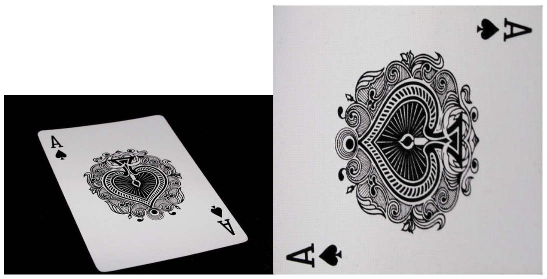
(a) score = 0.18763 (b) score = 0.79940

(c) score = 0.36798 (d) score = 0.69048

Figure 2: Top row - images with complex backgrounds correctly classified; Bottom row - images with no backgrounds, used for training



(a) OpenCV extracting the 8 of Spades from an image with multiple cards



(b) Pre processed Ace of Spades

(c) Post processed Ace of Spades

Figure 3: OpenCV pre processing on images

Results and Discussion

Upon implementing OpenCV and expanding the training dataset to include images which did not contain backgrounds, we trained a classifier over 32,000 epochs. This training was performed on a CPU and ran overnight. The classifier we trained exhibited a 90% accuracy on test data. Figure 4 shows the confusion matrix for this model with respect to all 52 classes.

We also experimented with classifying images which contained more than one card. Figure 3(a) gives an example of this. The process which worked best was to convert the image to black and white, based on a pixel intensity threshold, and to then isolate and rotate the cards before classifying them one at a time.

Conclusions

The learning architecture of the neural network in TensorFlow began to show a capacity for classifying playing cards almost immediately. However, by including enhancements such as training data without extraneous information, and the pre processing capability of OpenCV, we found we were able to substantially increase the classification power of our model. With enough training, we found that our dataset and model structure allowed us to achieve a maximum out of sample accuracy of 90%. The model accuracy could be further improved improved by fullly retraining all of the convolution neural network layers with playing cards.

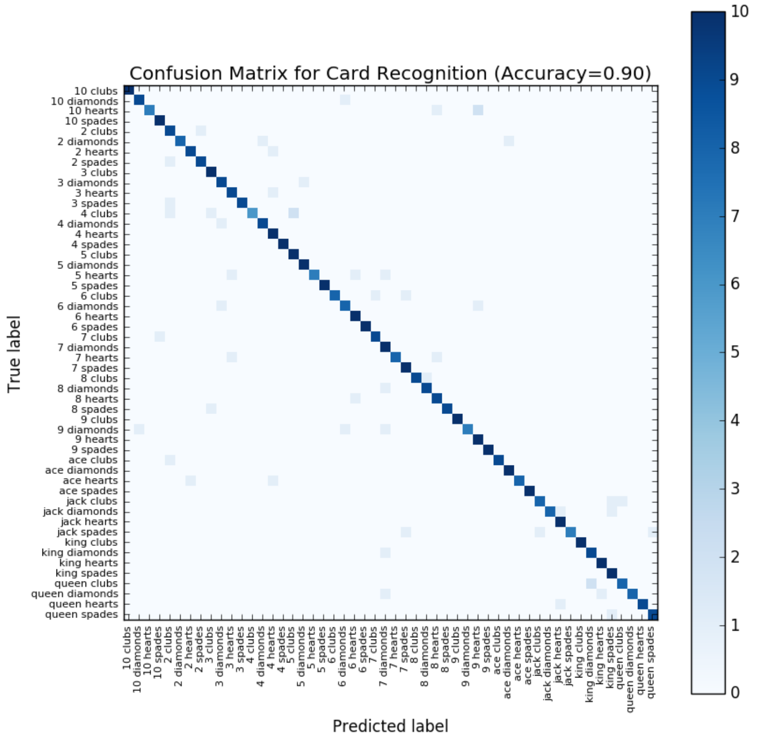


Figure 4: Confusion matrix for classifier after 32,000 training epochs with a test set size of 520 cards (10 per card)

There is no current state of the art in card classification, but image classification itself has become a booming field. Our accuracy is not quite production ready, but clearly demonstrates the potential of this method as a proof-of-concept that could be implemented in an interface to allow computers and humans to play various card games.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., … Zheng, X. (2015). *TensorFlow: Large-scale machine learning on heterogeneous systems.* Retrieved from `http://tensorflow.org/` (Software available from tensorflow.org)

NANDI, A. (n.d.). *I suck at 24: Automating card games using opencv and python.* http://arnab.org/blog/so-i-suck-24-automating-card-games-using-opencv-and-python.

Nielsen, M. A. (n.d.). *Neural networks and deep learning.* http://neuralnetworksanddeeplearning.com/chap6.html.

Tensorflow. (n.d.). *Using gpus - tensorflow.*

Theano. (n.d.). *Using the gpu theano 0.8.2 documentation.*

tmbdev, T. (n.d.). *ocropy.* https://github.com/tmbdev/ocropy.