Project: Counting Coins

by Jadon Manilall (815050)

Abstract—**In this report, we explore the idea of using a HOG+SVM trained object detector along with a Convolutional Neural Network to identify and classify coins in an input image according to their value. The musical structure of the input is stored in a text file and is encoded using ABC notation. The RNN model is able to read in and generate music directly in this format. This is done by providing the text of the ABC notation to the model as input, and training the model to predict the next character of the tune given all previously seen characters. Then, music can be generated on a character by character basis by simply letting the RNN model predict new characters and then feeding those characters back into the model as an input.**

I. Introduction

THE process of counting coins has been around since the dawn of capitalism. And for the longest time has remained a somewhat manual process.

Recently however, a lot more of effort has gone into automating this mundane task. One of the most widely used of these techniques for achieving this involves using the coins physical attributes such as its weight and area to determine its value.

While this approach enjoys a high level of accuracy, it can be costly to implement. Furthermore, these systems do not have the ability to easily to adapt to a different coin set than what it was designed for.

In this report, we aim to implement a picture based system to automate the process counting coins. The approach implemented takes place in 2 stages. The first is the coin detector stage which takes in an image and uses an object detector (trained to detect coins) to find and segment out each coin in the image. The second stage then takes in those detected coins and attempts to classify the coin according to it's value.

The next section of this report goes on to describe the methods used to solve the problem. As well as highlight some key concepts needed to train both the coin detector and the coin classifier. Then section (III) provides details about the implementation of the of the solution as well as experiments performed which is then followed up by the results obtained from those experiments in presented in section (IV). Finally we conclude by discussing a major issue that this solution faces and how these issues are commonly solved.

II. Methodology

As stated above, the approach taken for this project consists of 2 main stages. The first stage takes in an image & then uses a **HOG+SVM** (Histogram of oriented gradients with a Support vector machine) object detector to identify & crop out all regions of an image that it ”sees” as coins. Then, the second stage uses a **CNN** (Convolutional Neural Network) to take in each of those cropped images and then attempts to assign a value to it.

A. Stage 1: The Coin Detector

Initially, quite a number of methods were considered for this stage of the project. One of these approaches included using a shape detector namely, the Hough Transform tool to detect the circular shape of coins in an image. The thinking was that seeing as coins are usually circular in nature, this approach should be able to detect the circular objects(coins) while ignoring all non-circular objects (rulers, pens, student cards etc.). While this approach seemed to have some level of success in detecting the coins, it quickly showed it's naive nature when it was tested against images in the dataset that had some perspective to it. The perspective made the coins in the images look like ellipses instead of circles which in turn led to a high rate of incorrect detections. Thus, due to the nature of the task at hand, this approach was abandoned altogether in favor for a much more robust technique.

1) The HOG+SVM Object Detector:

In the paper [1] presented by Dalal et al. A new method for object detection was presented. This new method used a combination of HoG Descriptors and a Linear Support Vector Machine (SVM) to achieve nearly perfect results when tasked with identifying human pedestrians on a busy street.

Although the exact implementation of this approach presented in [1] is quite complicated, the general outline can be broken down into the following steps:

- **Step 1:** Given a set of training images containing the objects to detect(eg. coins). Take P positive samples of the object in interest and extract its HoG descriptors.
- **Step 2:** Take N negative samples from a negative training set which does not contain any objects of interest.
- **Step 3:** Train a linear support vector machine on the P positive and N negative samples.
- **Step 4:** Then For each image and each possible scale of each image in the negative training set, apply a sliding window and compute your HOG descriptors and apply your classifier. If your classifier (incorrectly) classifies a given window as an object (and it will, there will absolutely be false-positives), record the feature vector associated with the false-positive patch along with the probability of the classification. This approach is called hard-negative mining.
- **Step 5:** Take the false-positive samples found during the hard-negative mining stage, sort them by their confidence (i.e. probability) and re-train your classifier using these hard-negative samples.
- **Step 6:** Obtain a trained object classifier.

B. Stage 2: The Coin Classifier

For this stage of the project a convolutional neural network was chosen to classify images of coins. The Reason for this choice stemmed from the high amount of variability that could exist even in a single class of the image data(seen in 1), as any two coins with the same value could look vastly different in input image depending on where it's placed, lighting effects etc. thus manually designing the right features to use for the classifier would be an extremely difficult task.

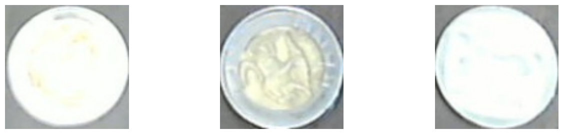
Fig. 1: Three separate coins from the same 5R class

1) ***The CNN Classifier***:

Convolutional Neural Networks have been around for quite some time now and has proved it's effectiveness time and time again.

Go on about how they work...

III. Implementation

This section descibes the details of the implementation of the stages and methods presented in section (II).

A. What Was Used?

Fortunately we live in an age were most of the complexity surrounding the implementation of these methods can be abstracted away with an import statement.Below lists some of the key technologies used to achieve the results in section (IV).

- The code was written in **python3** using the **spyder** IDE.
- A combination of **openCv** and **skimage** was used for image manipulation.
- The **dlib** toolkit (for python) by Davis King was used to build and train the HOG+SVM object detector.
- **Keras** with a **Theano** back end was used to build and train the Convolutional Neural Network.
- **Google's Cloud Platform** was used to train the CNN's

B. About The Data

The input dataset for this project consisted of 207 .jpg images. In order to be able to train as well as test weather our implementation is actually working we first had to split the original dataset into 3 separate datasets, a test set, training set and validation set. The test set consisted of about 15% of images while the training set consisted of about 65% of the images. The validation set was made up of 20% of the images and was kept away during the training of both stages.

Further more, some of my own images (12 of them) were added into the training dataset to increase the size of the dataset and help prevent over fitting.

C. Training The Coin Detector

Creating a HOG+SVM object detector from scratch is a challenging and extremely time consuming process. Thankfully dlib provides a simple API to do just that!

All that's required is a set of positively annotated training images. Annotating an image simply means drawing a rectangle around the object in an image you want the detector to learn to detect.

So to train dlib's object detector to detect coins, we took the set of training images (all 143 of them) and manually drew rectangles around each coin in each image (shown in 2).

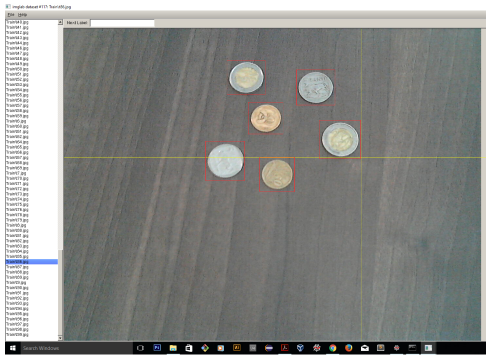
Fig. 2: Manually annotating coins using dlibs imglab

Approximately 2 hours and 13 hand cramps later we get an .xml file containing each image in our training set along with co-ordinated specifying the position of each coin in an image.

This .xml file is then used to train the coin detector. Figure 3 shows the learned HOG filter.

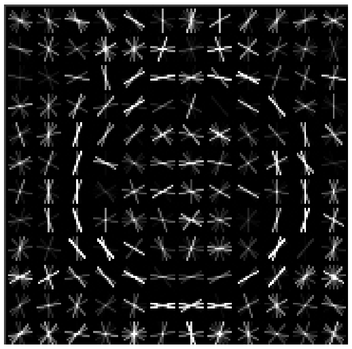
Fig. 3: Learned HOG filter from training data

D. Training The Coin Classifier

Training the convolutional neural network was a much more involved process, as a large amount of labeled training images had to be gathered.

The accumulation of these images were done in a few ways.

- First the 106 labeled images provided were used as masks against the corresponding 106 original images in order to crop out each coin in the images. After that process was complete, each coin was then labeled individually according to its actual value. However, some coins were extremely difficult to distinguish between.

- Next the learned coin detector was run on both the training and test set. Which the segmented and saved each coin as its own image.

IV. Results and Analysis

V. Conclusion

In this report we have demonstrated the power that these RNN models have over sequential and/or variable length input data. In particular, we implemented a single layer RNN which takes as input a chunk of raw text and builds up a probabilistic language model to generate similar sequences of text.

Although the model that we chose to implement may look very simplistic in the context of the larger class of RNNs. Our trained model still managed to produce desirable melodic output.

However, these models are not without their drawbacks. One of the main motivations established for using RNNs mentioned in section (I), is their potential to relate previous information to a given current task under consideration. But in some cases where more information is required, it becomes entirely possible for this 'information gap' between the relevant information and the point where it is needed to become very large. In theory, RNNs are absolutely capable of handling these long-term dependencies. but unfortunately, as that information gap grows, RNNs become unable to learn to connect the related information between time steps. This problem, commonly known as the 'vanishing gradient' problem was explored in depth by Bengio, et al. in [2], who found some fundamental reasons why it might be difficult to learn these long term dependencies using ordinary gradient descent.

Fortunately, these difficulties associated with training RNNs are much better understood[3]. Long Short Term Memory networks (LSTMs) are a special kind of RNN, which were explicitly designed to avoid the long-term dependency problem. They were introduced by Hochreiter et al. in [4], and were later refined and popularized by many researchers because of their ability to perform well on a variety of problems.

Acknowledgments

Andrej Karpathy's blog post in which he demonstrated the effectiveness of a character-level language models based on RNNs.

The abc music project developed by Chris Walshaw, which made converting between a plain text file to a midi file and vice versa an extremely simple task.

The Nottingham Music Database maintained by: James Allwright and Eric Foxley which contains over 1000 Folk Tunes stored in the abc format.

References

[1] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*, ser. CVPR '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 886–893. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2005.177

[2] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *Trans. Neur. Netw.*, vol. 5, no. 2, pp. 157–166, Mar. 1994. [Online]. Available: http://dx.doi.org/10.1109/72.279181

[3] R. Pascanu, T. Mikolov, and Y. Bengio, "Understanding the exploding gradient problem," *CoRR*, vol. abs/1211.5063, 2012. [Online]. Available: http://arxiv.org/abs/1211.5063

[4] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735