Real-Time Trading Card Recognition in Live Video

Kevin Culberg
Department of Computer Science
Stanford University
Stanford, CA 94305
kculberg@stanford.edu

Abstract

Tournaments for physical card games such as Magic: The Gathering attract thousands of online viewers. Despite this success, tournament livestreams are not easily accessible for newer players who have not memorized the many unique cards. I propose an object detection model based on the popular "You Only Look Once" (YOLO) model architecture to detect cards from real time video of tournaments to improve viewer understanding and accessibility. Detection time for my model averaged 57.1 ms per frame while still achieving a mean average precision (mAP) score of 62.09% on card bounding boxes. As the number of unique classes increased the predictive power of the model suffered indicating that the model architecture would need to be changed further in order to be applied in real tournaments.

1 Introduction

The popularity of physical trading card games has increased in recent years with the most popular card game Magic: The Gathering earning over $300 million in 2016 and attracting over 30,000 online viewers to its livestreamed tournaments.[1] Despite this success, tournament livestreams are not easily accessible for newer players who have not memorized the thousands of unique cards. Game rules depend on knowing the text on each card played, but these cards are unreadable on video. In this paper I approach trading card recognition in live video as an object detection problem and propose a model structure to address the challenge of accurately detecting dozens of cards from unique classes fast enough to work on live video.

The object detection model will take in as input a still frame from a video of a tournament. This single frame contains three color channels and is resized to a height and width of 608 by 608 pixels. The network then performs a series of 2d convolutional layers and other operations before outputting a feature map tensor. This feature map tensor has the dimensions $S \times S \times (5 + C)$, where S is the number of cells remaining and C is the number of unique classes. This feature map contains information about all detected bounding boxes and the probability that the box is of each type of class. The feature map can then be translated into a list of detections which can be overlaid on top of the video frame or output in another way to make viewing of the livestreamed tournament easier.

There are a number of advantages to this type of problem that work to make the task easier than many other object detection tasks. First, almost all tournament video is shot from a top down angle of a table of cards with a solid colored background. This results in all objects having roughly the same size, shape, and lighting. This perspective also means that things like skew or background textures play a minimal role in warping the visual representation of the cards. Furthermore, in most cases each class has only one way to represent it. That is to say that each card of the same type will have the same artwork and thus the same physical appearance. This leaves only visual invariances caused by rotation or occlusion that will need to be addressed by the network. Cards may be rotated on the table during the course of play and occlusion is common from a player's hand or another card.

2 Related work

Object detection is the computer vision task of identifying and correctly classifying objects within an image. This task combines the challenge of correctly predicting a bounding box for an object and classifying the object with the correct label. Early work in object detection using deep neural networks relied on re-purposing image classification networks to perform two stages: one to predict the bounding boxes and the second to classify the object within each bounding box.[3][2] These techniques were effective, but also took more time to train and predict due to requiring two or more passes through the neural network to obtain the detections.

Later work such as the aptly named "You Only Look Once" (YOLO) model combined these steps into a single pass through a network designed to be fast and efficient on the task of object detection.[4] YOLO is optimized to detection objects such as animals and people of varying sizes and types. For example, YOLO is prepared to detect objects that may take only a few pixels in the background of an image such as a bird flying in the sky to a large closeup of a dog that may take up the full frame. Additionally, the individual classes that YOLO is trained on can have drastically different physical appearances. Not only are there many different shapes and sizes for "person" objects, but there is also a large amount of variation in the camera angles, lighting, and other factors that can change the pixel representation in an image. YOLO is capable of detecting people in this variety of situations due to its structure. The current version of YOLO, version 3, predicts three different feature maps and each feature map predicts three bounding boxes per cell. [5] These feature maps can be thought of as a grid overlaid on top of the image. Each grid has a different scale with the smallest downsampling the image dimensions by a factor of 8 and the largest by a factor of 32. Another key aspect is that YOLO only calculates the loss from the bounding box location and size if that box has a large enough overlap with a true label bounding box. This way, the network learns to detect a large variety of objects with different cells on different feature maps specializing on objects of different sizes. However, one weakness is that increasing the number of classes greatly increases the size of these feature maps and makes training more difficult. My proposed model is most similar to YOLOv3 but differs because all the objects in card detection have a similar size and shape so only a single feature map needs to be predicted.

3 Dataset and Features

The datasets used for training and evaluating the baseline consists of full color images taken from tournament video with labeled bounding boxes for every card within the frame, Figure 1. A single image's label includes the bounding boxes represented as the x and y coordinates of the center point of the bounding box as well as the width and height, each as a percentage of the image height and width. Each bounding box is accompanied by a class number to identify which card is present within the bounding box. A single image may have as few as no objects or as many as dozens of unique objects.

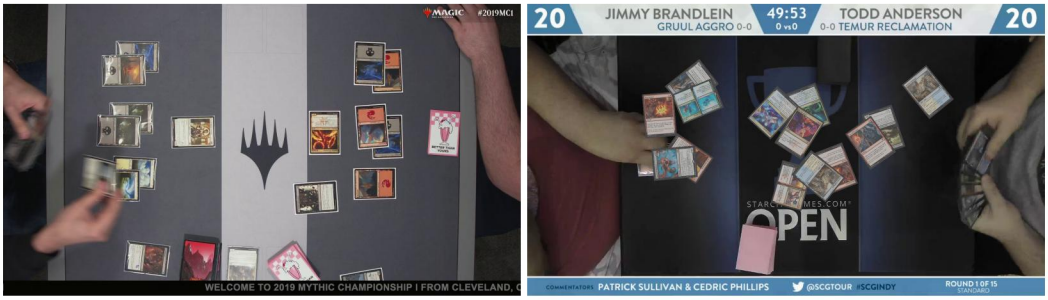

Figure 1: An example of a single frame from a tournament livestream (left) and an augmented image created by inserting images of cards onto a frame (right). These images are resized to 608×608 before being passed in as input to the network.

No labeled dataset previously existed that contained images from a tournament video with labeled bounding boxes. Due to similar presentation of tournament video in respect to lighting and background it is possible to generate a dataset which appears very similar to real images. This process involved

Table 1: Model structure

	Start			Region 1		Region 2		Region 3	
Type	Conv2d	Conv2d	Res x1	Conv2d	Res x2	Conv2d	Res x8	Conv2d	Res x8
Filters	32	64	32/64	128	64/128	256	128/256	512	256/512
Size	3	3		3		3		3	
Stride	1	2		2		2		2	

	Head						Output
Type	Conv2d	Conv2d	Conv2d	Conv2d	Conv2d	Conv2d	Conv2d
Filters	256	512	256	512	256	512	5+C
Size	1	3	1	3	1	3	1
Stride	1	1	1	1	1	1	1

taking frames of tournament video with no cards present to use as a background image and then inserting card images into the image so that the bounding box locations and labels are known. Card images are modified to have their lighting and color match those of a real image and they are randomly rotated and placed on the image to simulate invariances due to rotation or occlusion.

Three full datasets were created from the same base of 302 hand-labeled images with only real objects containing 51 unique class ids. These images were then split int a train/validation/test dataset containing 62/120/120 images, respectively. The images in the train split where then further augmented in the same manner as described above by artificially inserting cards from either the same pool of 51 classes or a larger pool of 264 classes. This process yielded three different training datasets: Dataset 1, Dataset 51, and Dataset 264, named after the number of unique classes they contained. These datasets could then be used to measure the impact that the number of classes had on the performance of the models. Datasets 51 and 264 contained 20,000 images total in their training set, of which 10% contained real frames with some augmentation and the rest were fully augmented. Dataset 1 contained only 10,000 images in the training set and used the label of "card" for all objects instead of a unique card name.

4 Methods

The model is implemented as a deep convolutional neural network (CNN) to perform detection in a single step similar to YOLOv3's darknet 53.[5] The key difference is that only a single feature map is predicted for the input image with a downsampling of 16. This means that an input image of 608×608 pixels will produce a feature map of 38×38. See Table 1 for full model structure. Each convolutional layer is a 2d convolution with filter size of 1×1 or 3×3. Each convolutional layer also uses batch norm with a decay of 0.9 and leaky ReLU as the activation function with $\alpha = 0.1$. No pooling layers are used throughout the model. Every residual block consists of two convolutional layers with the first layer having a filter size of 1 and the second having twice the number of filters (the same number as the input to the residual block) and a filter size of 3. The output of a residual block is the output from this second layer added to the input to the residual block.

The first few layers of the network consist of two convolutional layers with 32 and 64 filters, a filter size of 3, and stride of 1 and 2, respectively. This is followed by a single residual block which results in the base image height and width cut in half. Residual blocks allow for the outputs from earlier layers to be combined with later layers through addition. The body of the network can be thought of as three regions. The first layer of each region is a convolutional layer with filter size 3 and stride 2 that doubles the number of filters from the previous block and halves the height/width of the input. This is followed by a number of residual blocks so that the output maintains the same dimensions throughout the block. The number of residual blocks in each of the three regions are 2, 8, and 8. The output dimensions from the body of the network are a tensor with dimensions $38 \times 38 \times 512$, which is then passed to the head of the network to be converted into a feature map.

The head of the network consists of six convolutional layers alternating between filter size of 256 and size 1 and 512 with size 3. This is then passed through a final convolutional layer of size 1 with the number of filters corresponding to $5 + C$ where C is the number of classes to produce the final feature map. Each cell in the feature map can be thought of as containing a feature vector of length $5 + C$. The first two elements of this vector are t_x and t_y and correspond to the predicted bounding box center x and y position as a ratio of the original image. The next two elements are t_w and t_h and correspond to the predicted bounding box width and height as a ratio of the original image. The fifth

element is the probability that the cell has predicted an object which is referred to as its objectness score. The final C elements are a probability vector that sums to 1 with the probability that this object is one of C classes. This feature map can then be translated to a list of detections where each detection much meet some threshold for its objectness score such as greater than 0.5.

To convert between the feature map predictions and bounding box center coordinates use equation 1 and for bounding box size apply equation 2:

$$b_x = (\sigma(t_x) + c_x)/S \tag{1}$$

$$b_w = \sigma(t_w) * (w_{\text{max}} - w_{\text{min}}) + w_{\text{min}} \tag{2}$$

where b_x and b_w are the bounding box center x coordinate and box width; c_x is the x index of the cell, S is the number of cells, w_{max} and w_{min} are the maximum and minimum widths of a bounding box in the dataset and are calculated once over the entire dataset and not changed. These equations can be applied to the y coordinate and box height by swapping the variables.

The loss function used during training combines the loss from each aspect of the predicted bounding boxes, see equation 3. The loss from the predicted bounding box coordinates and size are calculated using sum of squared error between the predicted values and the true values. This loss is masked so that it is only calculated for those cells that have an object present in the true label. The loss from these predictions is also weighted with $\lambda_{\text{coord}} = 5$ and $\lambda_{\text{size}} = 5$. The loss for correctly predicting if a cell contains the center point for an object is done with binary cross entropy and is weighted with $\lambda_{\text{noobj}} = 0.5$ so that failing to predict no object has a lower impact to training because many more cells will not contain an object during training. The loss for correctly predict the object's class id is calculated by applying softmax and cross entropy error.

$$\begin{aligned} L = & \frac{\lambda_{\text{coord}}}{S^2} \sum_{i=0}^{S} \sum_{j=0}^{S} \mathbb{1}_{ij}^{\text{obj}} ((x_{ij} - \hat{x}_{ij})^2 + (y_{ij} - \hat{y}_{ij})^2) \\ & + \frac{\lambda_{\text{size}}}{S^2} \sum_{i=0}^{S} \sum_{j=0}^{S} \mathbb{1}_{ij}^{\text{obj}} ((w_{ij} - \hat{w}_{ij})^2 + (h_{ij} - \hat{h}_{ij})^2) \\ & - \frac{1}{S^2} \sum_{i=0}^{S} \sum_{j=0}^{S} \mathbb{1}_{ij}^{\text{obj}} d_{ij} \log \hat{d}_{ij} - \frac{\lambda_{\text{noobj}}}{S^2} \sum_{i=0}^{S} \sum_{j=0}^{S} \mathbb{1}_{ij}^{\text{noobj}} (1 - d_{ij}) \log (1 - \hat{d}_{ij}) \\ & - \frac{1}{S^2} \sum_{i=0}^{S} \sum_{j=0}^{S} \mathbb{1}_{ij}^{\text{obj}} \sum_{c \in \text{classes}} p_{ij}(c) \log \hat{p}_{ij}(c) \end{aligned} \tag{3}$$

The baseline model tested so far is an implementation of YOLOv3 (github) with a 608 x 608 pixel input to match my proposed model. [4] This model was selected as a baseline due to its proven speed and accuracy as a state of the art object detection network. This model is also capable of performing the full detection and classification task end to end as well as comparison on the first stage task of bounding box prediction.

5 Experiments/Results/Discussion

The two models were evaluated on three separate datasets by measuring the mean average precision (mAP) of the detections they generated as well as their detection speed per image. For each dataset, my model was trained for 10 epochs with a learning rate of $1e^{-3}$ and batch size of 2. The loss function was weighted with $\lambda_{\text{coord}} = 5$, $\lambda_{\text{size}} = 5$, $\lambda_{\text{noobj}} = 0.5$, and $\lambda_{\text{class}} = 1$. The YOLOv3 baseline model was also trained for 10 epochs with a learning rate of $1e^{-3}$ and batch size of 1.

Full results for both models on each metric are available in Table 2. Both models performed well within requirements for single frame prediction speed with my model being slightly faster to predict objects for a single frame at 57.1 ms. With this speed it would be possible to predict over 15 frames per second when requirements for assisting viewers of live tournaments would only need a single frame per second. My model also outperformed YOLOv3 on mAP for the datasets with multiple

Table 2: Results

	Mean Average Precision (mAP)			Speed
	Dataset 264	Dataset 51	Dataset 1	(ms)
me	**5.31%**	**6.78%**	62.09%	**57.1**
YOLOv3	0.10%	2.20%	**86.08%**	67.4

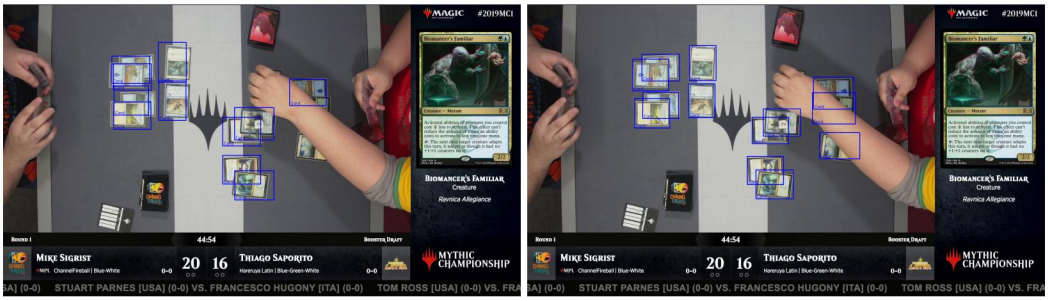

Figure 2: Predicted objects on a single frame from Dataset 1 produced by my model (left) and the YOLOv3 baseline (right).

classes. However, both models performed very poorly on these datasets and produced results that would not be useful for real tournament video. YOLOv3 outperformed my model on Dataset 1 with a mAP of 86.08% compared to my model with 62.09%. Both of these scores would be acceptable for real tournament video with the main difference being how tight the predicted bounding boxes are to the card border. Figure 2 contains side by side examples on the same frame from the test set. From this is it clear that almost all cards are detected by both models, but the outlines for the bounding boxes are messier from my model. The cards that are missed by my model are those that are heavily obscured by the players arm. Looking closer at the bounding boxes for Datasets 264 and 51 it is clear that as the number of classes increases both models do worse at predicting both the correct class and at the bounding box shape. There was also a large amount of training instability that is likely caused by the multiple aspects of the loss function pulling the model in different directions. It is possible that readjusting the weights of the various parts of the loss function could correct for this in a small way and improve results.

6 Conclusion/Future Work

I believe that the failure to predict cards when multiple classes were used in the training set is due to the large amount of time needed to successfully train the network and the small size of the dataset. However, the number of unique classes will continue to restrict the model's ability to scale to thousands of unique cards, which would be required for many real tournament applications. Results were promising when limited to only predicting card outlines even when presented with multiple overlapping cards and other forms of occlusion such as player's hands. It is likely that building a separate network to do card image classification from the region of the video frame located within the predicted bounding box would offer a way to obtain accurate classification results. This would allow developing a network with a structure specialized for that task such as those used for facial recognition, which is another problem involving predicting thousands or more classes that all have similar size and shape. This would also help the card outline detector model to work better by having a loss function with greater stability due to not being pulled in other directions by the class prediction loss. Due to the quick bounding box detection time I believe that the addition of another stage network would not cause the end to end time for detection and classification to be too slow for realistic use.

7 Contributions

Kevin Culberg worked independently on this project. Source code is available on Github - https://github.com/culk/StreamSight

References

[1] CML. The problem with magic: the gathering no one is talking about. *The Daily Dot*, 2016.

[2] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[3] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[4] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[5] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018.