EEL 5825 Final Project Report

Michael Fielder
Masters of Computer Vision
University of Central Florida
Orlando, Florida
mi017242@ucf.edu

***Abstract*—In recent years, the field of computer vision has been revolutionized with advancements in deep learning networks like convolutional neural networks (CNNs). These models allow for efficient training and accurate training on several different methods including object detection. In this project, my goal is to create a card detector for the trading card game, Flesh and Blood to hopefully be used by other players. My goal is to use a completely synthetic dataset to bypass the oppressive annotation task that would be required for all the cards in the data. I trained a YOLO-based model on completely synthetic data and was able to detect cards on real test images. This shows that this method has promise in detecting cards in general.**

I. Introduction

Convolutional Neural Networks (CNNs) have revolutionized the field of Computer Vision. Models like VGG, ResNet, and YOLO have allowed for efficient and accurate results on number of tasks including the topic of my project, object detection.

The main focus of my project is to detect trading cards from the Trading Card Game (TCG), *Flesh and Blood*. The game involves two players constructing decks and using those decks against each other. Understanding what cards do is an important aspect of the game. This is easy to overcome in person. If someone wants to know what a card does on the board, they can read it on the table. However, many players play online using a webcam. This can run into problems since the fidelity of the camera can make it difficult to see the cards.

My project intends to solve this problem by creating a card detector using deep learning that can identify cards in an image which would allow players to read them as an image on their computer.

The biggest hurdle with deep learning methods like CNNs is their reliance on data. These kinds of networks are inherently data-hungry which means that a lot of annotated examples are required to train a model. This can be difficult to achieve in this scenario because *Flesh and Blood* currently has hundreds of possible cards to use and is always adding more cards through expansion. To construct a handmade dataset with pictures of real-cards would require a lot of time and money to achieve.

One possible solution is to use synthetic generation to create dataset samples. This would allow for thousands of examples to be generated instantaneously and would allow for the training pipeline to be easily updated when new cards are added through expansions. Synthetic data is not perfect, however. It suffers from a condition know as *domain shift*, which is related to the fact the physics and lighting in a synthetic image are different than one taken from a camera. This can result in a drastic decrease in performance when moving from one domain to the next.

My goal for this project is to see the impact of domain shift when training a model with a synthetic dataset and to see if purely synthetic data can be used to detect real-life cards.

The specifics of how the dataset is generated and processed by the model are explained in the **Method Explanation Section**. A general description of the method can be explained with the following:

1) Generate Synthetic Dataset and Real Dataset
2) Train Model on Synthetic Dataset
3) Test Performance on Validation Synthetic Data
4) Test Performance on Real Images

Synthetic examples will be generated using PNGs of the cards instead of physical ones. The backgrounds that the cards fall on will be a random texture found in the *Oxford Describable Textures Database* [3]. This contains different textures that will simulate the various playing surfaces that players may use when playing. It also makes the model more robust in general. The objective is to make the synthetic samples represent a table where cards have been plopped onto it. The cards are rotated by $-45, 45$ to add some variance. To prevent the cards from being placed right on top of each other, a card is only placed if it doesn't overlap with another card by more than 30%. This makes sure that the cards are still occluded but not impossible to detect. Augmented versions are also generated of these images to create more examples. The augmentations applied include Shear, Blurring, Exposure, and Cutout.

The real dataset is constructed using an iPhone and a webcam. Images taken with the iPhone can be considered "easy" examples while the webcam examples can be considered "hard". Both were taken directly above the table to simulate how they would be used when playing.

In terms of the actual dataset, there are 12 card classes to identify. I chose a subset of cards from a starter deck because of the amount of time available. The hope is that if the model can identify a handful of cards effectively, it can effortlessly extend to the full dataset.

The full dataset is stored on RoboFlow [4] which allows for easy storage and utilization with a model. RoboFlow also allows for augmentations to be applied to images to create

Fig. 1. Some Examples from Data (TOP-LEFT: Synth. w/ No Aug., TOP-RIGHT: Synth. w/ Aug., BOT-LEFT: iPhone, BOT-RIGHT: Webcam)

more samples. This is the feature I use to add augmentations to my image.

Some examples from dataset can be found in Figure 1

The model that I train on is *Ultralytics* YOLOv8 [5]. My reasoning for choosing this model is that it is relatively fast, which allows it to be used with a variety of different devices. This is important for my application since this application should be usable on everything from a computer to a phone.

II. Related Work

My main inspiration for this project comes from the application, *SpellTable* [1]. SpellTable is an application that allows users to play another TCG, *Magic the Gathering* (MTG) remotely with others using their webcams. It has a feature very similar to what my project is trying to achieve and that is the ability to detect cards. Its results are impressive, as it can detect cards reliably with a larger pool of cards (MTG is much older than Flesh and Blood). This serves as direct proof that something like this is possible. The exact method used however is unknown and proprietary. The creators of Magic the Gathering have a lot of resources at their disposal, which makes the collection of real annotated images possible.

Another piece of prior research that I found was the blog post on creating synthetic images for Detection Uno Cards [2]. This involves taking real photos of cards and placing them on a random background from a collection of textures. This method has several differences from my method but served as an inspiration for some of my decisions.

Both of these sources were the main influences on my method and served to guide the structure of my setup.

III. Method Explanation

Ultralytics, the creators of YOLOv8, have not released an official paper explaining the full details of the model. With this in mind, I will focus on the aspects of YOLO [6] model architectures that are ubiquitous.

The models take in as inputs a 3D dimensional tensor which can be defined as $I = D \times D \times 3$, where D is the dimension of the image.

The main advancement of YOLO models is that they divide the image into a series of grids and predict bounding boxes for each grid cell. This limits the total number of detections, which results in more efficient inference.

Formally, it divides the image into $S \times S$ grid cells where it predicts B bounding boxes for each cell and C class probabilities. The predictions are encoded as an $S \times S \times (B * 5 + C)$.

The loss function is the other aspect that is interesting about this model. It can be defined as the following three parts together,

1. Bounding Box Coordinates Loss (L_{coord}):

$$L_{\text{coord}} = \lambda_{\text{coord}} \sum_{i=1}^{N} \sum_{j=1}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_{ij} - \hat{x}_{ij})^2 + (y_{ij} - \hat{y}_{ij})^2 + (\sqrt{w_{ij}} - \sqrt{\hat{w}_{ij}})^2 + (\sqrt{h_{ij}} - \sqrt{\hat{h}_{ij}})^2 \right]$$

This calculates the loss between the predicted and ground truth bounding box coordinates. x_{ij}, y_{ij} and $\hat{x}_{ij}$, $\hat{y}_{ij}$ correspond to the center of the bounding boxes in the (i, j) box. w_{ij}, h_{ij} and $\hat{w}_{ij}$, $\hat{h}_{ij}$ correspond to the height and width of the box.

2. Objectness Confidence Loss (L_{obj}):

$$L_{\text{obj}} = \sum_{i=1}^{N} \sum_{j=1}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(P_{\text{obj},ij} - \hat{P}_{\text{obj},ij})^2 \right]$$

$P_{\text{obj},ij}$ and $\hat{P}_{\text{obj},ij}$ correspond to the probability that an object is in a detected bounding box. $\hat{P}_{\text{obj},ij}$ is either 0 or 1.

3. Class Probability Loss (L_{class}):

$$L_{\text{class}} = \lambda_{\text{class}} \sum_{i=1}^{N} \mathbb{1}_{i}^{\text{obj}} \sum_{c=1}^{C} \left[(P_{\text{class},i,c} - \hat{P}_{\text{class},i,c})^2 \right]$$

Calculates whether or not the model calculated the correct class when looking at a a given bounding box.

4. Total Loss (L):

$$L = L_{\text{coord}} + L_{\text{obj}} + L_{\text{class}}$$

The grid-based bounding boxes and the specific loss function are the two big advancements YOLO models give for object detection.

IV. Pseudocode

In this section, I show the pseudocode for the synthetic data generator. This was the development of the project, so laying it out is more significant than the code for the model training. This is because Ultralytics have provided users with an easy to use interface to train and deploy models. Their is nothing novel about the model training pseudocode and re-stating the obvious would not be beneficial for the report.

Algorithm 1 Generate Synthetic Dataset

function GENERATEDATASET(N)
  $backgrounds \leftarrow$ list of N background paths
  $coco \leftarrow$ COCO JSON object
  for b **in** $backgrounds$ **do**
    $background \leftarrow$ load_img(b)
    $c\_bboxes \leftarrow$ list of card coordinates
    $coco$.add_image()
    $C \leftarrow$ # of random image
    for c **in range** C **do**
      $card \leftarrow$ load_card()
      $card \leftarrow$ rotate_card($card$)
      $card \leftarrow$ resize_card($card$)
      $x, y \leftarrow$ random o card
      check_overlap(x, y, c_bboxes)
      $background \leftarrow$ paste_card($bg, card, bbox$)
      c_bboxes.append($bbox$)
      $coco$.add_annotation($bbox$)
    end for
    save_img($background$)
  end for
  return $coco$
end function=0

V. Experiment Settings

The generated data set contains 6848 images. 6375 training images were purely synthetic data with augmentations, 375 synthetic images with no augmentations were used for validation, and 98 images were hand-annotated real images taken with an iPhone and a webcam were used for testing purposes to see if the method would work correctly. The dataset is stored as *RoboFlow* which allows for ease of storage, but also facilitates the application of augmentations to the image. All images are resized to a size of 416×416 to ensure consistency and

I trained the model in a *Google Colab* notebook using the provided **T4 GPUs**. The model was trained for 25 epochs using the SGD optimizer at a learning rate of 0.01. A batch size of 64 was used.

VI. Experiment Results

Below are the results of my experimentation. Tables I and II show the results of the model when detecting synthetic images and real images respectively. The metrics used for evaluation are: Box(P), Box(R), mAP50, and mAP50-95. I also show examples of the model's bounding box predictions for a collection of real test images in Figure 2

VII. Result Discussion

The results achieved by the model are surprisingly good considering that it it was only trained on synthetic images. The top row of Figure 2 shows several examples where the model was able to accurately detect all the objects in an image. All these examples show accurate bounding boxes and classification labels with high confidence. This is proof that the method can work effectively on synthetic data with several augmentations applied. Another thing to note is that a large majority of the cards that were taken with the iPhone were able to be correctly idenfified. This shows that a common camera like the one found on a phone, has more than sufficient fidelity to identify a synthetically trained model. It also shows that when the fidelity is sufficient, the domain shift between synthetic and real images is low enough that it can be used effectively.

The bottom row of Figure 2 shows the misclassified images. The main way the model fails is in not identifying a bounding box with a high enough confidence (only boxes with a conf. > 0.25 are displayed). Something interesting to note is that most of the misclassified examples come from webcam photos. This makes complete sense since the webcam photos are

Class	P	R	mAP50	mAP50-95
bittering_thorns	1	0.984	0.995	0.992
brutal_assault	0.997	1	0.995	0.994
edge_of_autumn	1	0.999	0.995	0.995
flying_kick	0.998	1	0.995	0.993
head_jab	1	1	0.995	0.995
ira_crimson_haze	0.998	1	0.995	0.994
lunging_press	0.997	1	0.995	0.995
salt_the_wound	0.995	1	0.995	0.995
scar_for_a_scar	1	0.999	0.995	0.995
springboard_somersault	0.998	1	0.995	0.995
torrent_of_tempo	0.997	1	0.995	0.995
whirling_mist_blossom	0.998	1	0.995	0.995
all	**0.998**	**0.998**	**0.995**	**0.995**

TABLE I
PERFORMANCE ON SYNTHETIC VALIDATION SET

Class	P	R	mAP50	mAP50-95
bittering_thorns	0.757	1	0.939	0.614
brutal_assault	0.951	0.923	0.97	0.677
edge_of_autumn	0.84	0.587	0.865	0.588
flying_kick	0.886	0.926	0.965	0.653
head_jab	0.987	1	0.995	0.711
ira_crimson_haze	0.683	1	0.94	0.653
lunging_press	0.971	1	0.995	0.734
salt_the_wound	1	0.921	0.978	0.667
scar_for_a_scar	1	0.97	0.995	0.639
springboard_somersault	0.941	0.931	0.949	0.666
torrent_of_tempo	0.968	0.897	0.934	0.652
whirling_mist_blossom	0.984	0.926	0.957	0.678
all	**0.914**	**0.923**	**0.957**	**0.661**

TABLE II
PERFORMANCE ON *REAL* TEST SET

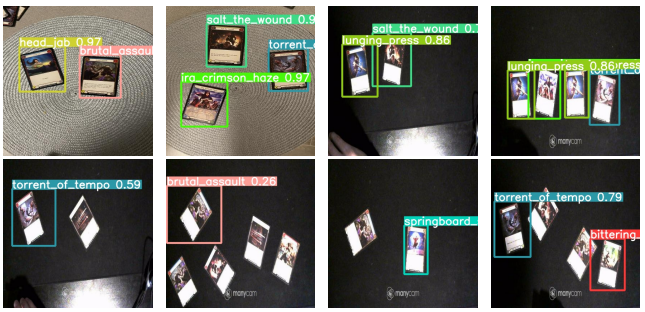

Fig. 2. Inference Results for Model. TOP: Correct Predictions, BOTTOM: Incorrect Predictions

lower fidelity and overexposed, making it harder to detect cards accurately. This is unfavorable since there are a variety of different possible cameras that people could use during playtime. Even in the misclassified images, the model can place accurate bounding boxes around a detected object with a medium level of confidence on some cards.

When comparing the mAP50-95 between the validation and test sets, it can be seen that there is drop-off. This is not due to overfitting, but most likely due to the difficulty of the webcam examples.

Several methods could be taken to improve the performance of difficult examples like the ones from the webcam. The first could be to add more possible augmentations to the image. This could provide more examples to create a more robust model. The application of these augmentations might also be another method. Bounding box-level augmentations have been shown to improve performance [7]. Samples were created in a simulation with something like Unity or Unreal could create samples that resemble real-life lighting conditions more accurately. The final thing that could be done is to put examples of real images into the training set, which could help bridge the domain gap. A combination of all these techniques could be used to bolster the training set with more examples to achieve better performance.

Overall, I believe that the results show a lot of promise for synthetic data being used to solve this particular task, and can most likely be scaled to be trained on ALL the cards present in the card database.

VIII. Conclusion

In conclusion, my experimentation worked and a simulated synthetic dataset was able to detect cards accurately detect real cards. If I have time, I intend to expand the synthetic data generation and test it with real players to see if it works in practice. I feel like the material I learned in this course helped me complete this project effectively. To improve performance, I'm going to try some of the improvements explained above to see if they help. Thank you for the great class professor!

References

[1] https://spelltable.wizards.com/
[2] https://blog.roboflow.com/improving-uno-with-computer-vision
[3] https://robots.ox.ac.uk/vgg/data/dtd/
[4] https://universe.roboflow.com/flesh-and-blood-dataset/full-dataset-ktr4z
[5] https://github.com/ultralytics/ultralytics
[6] https://arxiv.org/abs/1506.02640
[7] https://arxiv.org/pdf/1906.11172.pdf