Flesh and Blood Object Detector (CAP 5415 Final Project)

Michael Fielder

December 5, 2021

Note that this project was coded using Google Colab. Exports of the `.py` and `.ipynb` can be found in the zip provided.

1 Abstract

In this project, I utilize transfer learning and PyTorch's Detectron2 library to create an object detector for the card game Flesh and Blood using a subset of released cards. While not completely satisfactory, the results show that it is possible to create a card detector using the Detectron2 library.

2 Introduction

Flesh and Blood is a Trading Card Game similar to Magic the Gathering or the Pokemon TCG. A key feature of Trading Card Games (TCG) is the ability to collect a wide variety of unique cards or differing variety. This allows the creators of these games to inject new game play mechanics and monsters into already existing meta. Due to the COVID-19 pandemic or a lack of a local scene, many players choose to play the game online using a webcam to display their cards so that the opposing player can see. Playing the game online has some problems however. The biggest issue is webcam resolution, which makes the card difficult to read on the screen. This problem has been addressed in software such as SpellTable which presumably uses computer vision and image segmentation to allow players to click the cards on screen and receive a high quality digital version. SpellTable only works with Magic the Gathering, which means that a new model needs to be constructed in order to detect Flesh and Blood cards. This is the main objective of my project. However, due to time constraints, I only train it on a subsection of the released cards and simply calculate the bounding boxes for the objects.

3 Method

I decided to train my object detector on a dataset that contains a subset of cards from the TCG. Specifically, they are cards found in the beginner deck given out for free at card game stores. This deck contains multiple copies of 12 different cards from the set, which are the 12 classes tested for in my model.

Figure 1: Flesh and Blood Cards Used in Dataset

Due to the limited amount of time available it was necessary to only use a select number of cards because I wanted to source my dataset from a webcam to create a dataset that was similar to that found when playing the game online. To capture images, I used a webcam and my iPhone to capture 98 images of a random number of cards on the table. I then uploaded the images to the service Roboflow. This website allows me to annotate and apply preprocessing to my images. In the dataset, there are 279 annotations across the 12 classes, which results in 2.8 annotations per image on average. Each image is resized to a size of 416 x 416. Horizontal and vertical flips are applied to each image resulting in 3 unique training images for each image and 191 total images. The dataset is then split into 161 testing images, 20 validation images, and 10 test images. Figure 2 is a count of each class in the dataset.

Figure 2: Table of Card Counts

Classes	Precision
edge_of_autumn	9
ira_crimson_haze	9
scar_for_a_scar	27
whirling_mist_blossom	27
bittering_thorns	27
flying_kick	27
lunging_press	27
springboard_somersault	17
brutal_assault	26
head_jab	27
salt_the_wound	27
torrent_of_tempo	29
Total	279

Figure 3: Examples of Annotated Images

To train the model, I used transfer learning and the Detectron2 library in PyTorch. Transfer learning is an effective way to leverage a pre-trained network to identifying a new collection of objects. This is because a pre-trained network has already formed common features within images, which speeds up future training. I used the Faster RCNN model trained on the COCO dataset. It was trained in Google Colab using their provided GPUs. I trained for 1500 iterations and a batch size of 64 images. It took 30 minutes to train the network.

4 Results

Figure 4 displays the loss of the network during training. The loss of the network decreases consistently overtime and reaches a lowest loss of 0.093.

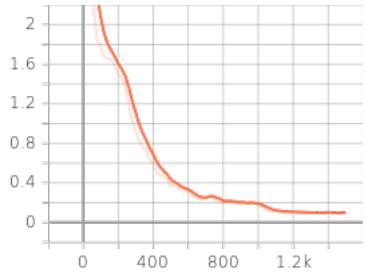

Figure 4: Loss of Network (X: Iterations, Y: Loss Value)

Figure 5 displays the Average Precision of the model during training. It increases steadily through training and reaches a maximum of 91.09%.

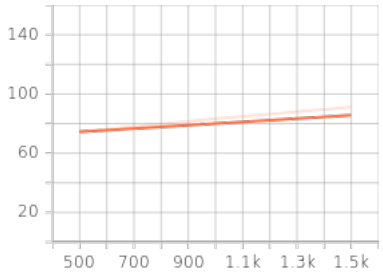

Figure 5: mAP of Network (X: Iterations, Y: mAP)

The average precisions for the network during testing can be seen in Figure 6,

Figure 6: Precision Table

Classes	Precision
edge_of_autumn	90.000
ira_crimson_haze	53.267
scar_for_a_scar	90.000
whirling_mist_blossom	91.683
bittering_thorns	95.545
flying_kick	82.525
lunging_press	100.000
springboard_somersault	90.000
brutal_assault	90.000
head_jab	NaN
salt_the_wound	96.634
torrent_of_tempo	97.624
mAP	88.84

Included below are images of the output produced by the model,

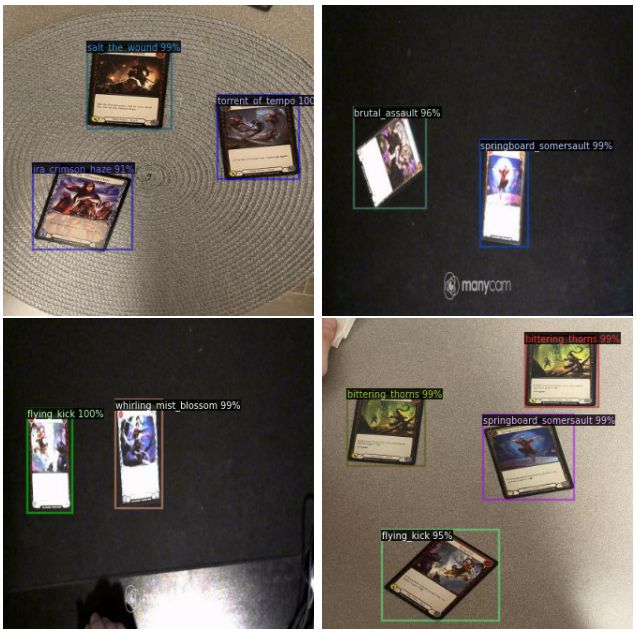

Figure 7: Images of Bounding Boxes Generated by Model

5 Discussion and Analysis

Considering the limited amount of time present for this project, I would consider it a successful experiment. There are however several issues with the results. First, the mAP for the Test set is low considering the methods I used. This can most likely be attributed to the low accuracy when attempting to detect the card `ira_crimson_haze`. An accuracy of 53.267% is most likely due to the model being trained on only 9 samples. This happened because there's only one example of that card in the free starter deck, resulting in the model encountering it less frequently. This can be remedied by including more annotated images with the card. Another issue is that the testing set did not contain a annotation containing the `head_jab` class. This makes the test set less representative of the actual dataset making the results less accurate. If I were to go back, I would refine the dataset and make sure that there is at least one example of every class.

If there was more time to perform this project, I would look into implementing a synthetic dataset and apply preprocessing steps to imitate different webcam conditions. A random number of cards would be placed on a collection of different textures to simulate varying playing surfaces. The main benefit of using a synthetic dataset, is time efficiency and extensively. It takes less time to generate thousands of images and when a new expansion set comes out, it is easy to add those cards to the generation set.

6 Conclusion

I would say that my project was a success in its goal to successfully detect multiple Flesh and Blood playing cards on a table. As addressed in the previous example, I do have some concerns with the precision values of some of the classes, but those can easily fixed. All in all, I believe this project taught me a lot about constructing custom datasets, the process of annotation, and running a model using transfer learning and a custom dataset.

7 Sources

1. https://blog.roboflow.com/how-to-train-detectron2/
2. https://towardsdatascience.com/detecting-set-cards-using-transfer-learning-b297dcf3a564
3. https://blog.revolutionanalytics.com/2016/08/deep-learning-part-2.html
4. https://detectron2.readthedocs.io/en/latest/tutorials/datasets.html
5. https://blog.roboflow.com/how-to-create-a-synthetic-dataset-for-computer-vision/