Pokémon Detection Using YOLO

Jonathan Henin

DSC680

Bellevue University

May 30, 2020

jhenin@my365.bellevue.edu

Instructor: Catherine Williams

Abstract

Object Detection, while a reasonably easy task for humans, is a challenge for computers. However, we can see examples of their usefulness played out in fictional type scenarios, such as the Pokédex device from the Pokémon world. We test to see if we can achieve similar functionality by training a YOLO(v3) model on Pokémon images and testing them against pictures and videos taken of Pokémon creatures (stuffed animal replicates). This particular model implements detection based on a proposed model for human object recognition called Template Matching Theory (TMT). Therefore, we are testing out both YOLO and TMT. Results of the model do show promise in detecting Pokémon as single objects but struggles to detect them when there are multiple Pokémon, particularly when they overlap; a challenge that is not present in human recognition. While YOLO and other TMT models can show usefulness in Object Detection, it is apparent that TMT fails to appropriately explain the human model.

Introduction

In the world of Pokémon, there is a device that every trainer receives, called a Pokédex. A Pokédex is a handheld device, much like that of a cellphone. One of the essential properties of this device is its ability to recognize Pokémon in the wild just from "seeing" them. Though the world of Pokémon is purely fictional, this particular device might fall within the realm of possibility. The goal of this project is to see if we can get a computer to recognize Pokémon in visual content. In order to accomplish this task, we delve into the world of machine learning and computer vision. To understand how object recognition works, let us first understand how humans are performing the task.

Template Matching Theory

A key aspect in humans is their natural ability to recognize objects within their visual field with remarkable ease and speed; qualities which do not diminish despite variations such as color, size, rotation, position, quantity, and occlusion. This ability stems from our brain's innate talents of pattern recognition. One of the most basic theories of human pattern recognition is called Template Matching Theory. In this theory, our minds store real-world patterns, called templates, which are

formed from past-living experiences (Shugen, 2002). As light enters our retina, the information is translated into electrical signals which get past on to the brain. These signals are then compared with our stored templates to find the best matching template. This is exactly how many computer vision models are implemented, so it stands to reason that these models are a good way to test this theory.

Basics of Computer Vision

Computer Vision is an interdisciplinary scientific field that aims to give computers a high-level understanding of digital images and videos (Ballard, 1982). In other words, we want to give computers the ability to recognize objects in a visual field, such as cars, traffic signals, people, and animals. Like the Template Matching Theory, computers need a set of prior experience templates, or in CV terms, ground truths, which correspond to images or videos with tagged regions. A tagged region is a set of coordinate values that represent the boundary edges around an object, along with a label for what the object is. In its simplest form, a tagged region can be a rectangular box of which the coordinate values can be represented mathematically by four values: t_x min (minimum x coordinate), t_y min (minimum y coordinate), t_w (the width), and t_h (the height). These templates serve as the computer's "experiences", and so when a computer is presented with a novel visual stimulus, what it is doing is comparing the new stimuli to its database of templates. Features are detected by comparing the amount of overlap, which is called the Intersection Over Union, or ICU. If there is a high ICU value then the confidence that what the object the computer is looking at is the same as the one it has it is in memory, increases.

In order to make this comparison, images are converted and stored as tensors, which are a generalized form of a matrix; a matrix being a 2-D grid of numbers, where a tensor can be any dimensional grid. Each pixel is converted into a number representing the color of that pixel (0 to 255). Greyscale images, which are called 1-channel images, can be represented by a 2-D matrix,

where one dimension represents the width, one dimension represents the height, and one for the color value as seen below in Figure 1.

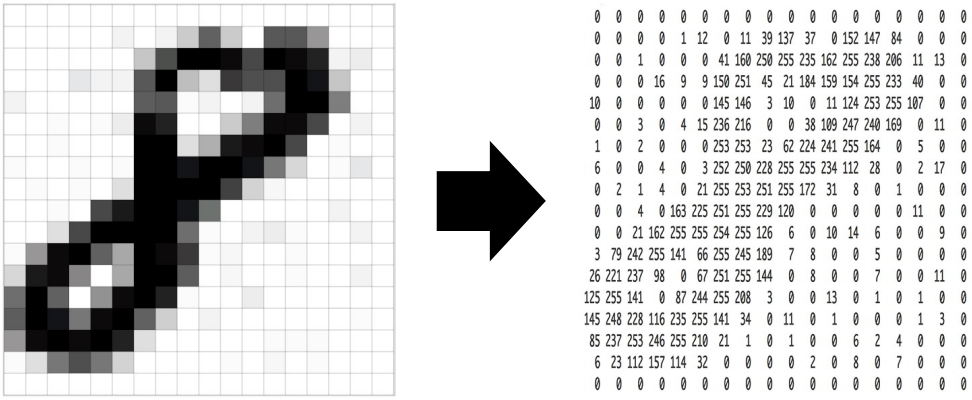

Figure 1: Number 8 as represented by a computer. Source: Geitgey, 2018

For images stored with more channels, each channel is represented by adding depth to a new third dimension, thus a 3-D tensor. For example, an image saved in RGB, or a 3-channel image, has the same two dimensions for height and width, but now has a depth of three, one representing the red channel, one for green, and one for blue, as seen in Figure 2. Again, values representing the RGB values, 0 to 255, are stored in each channel. Some numerical models will apply a squashing function, converting the scale from 0 to 255, to a scale of 0 to 1.

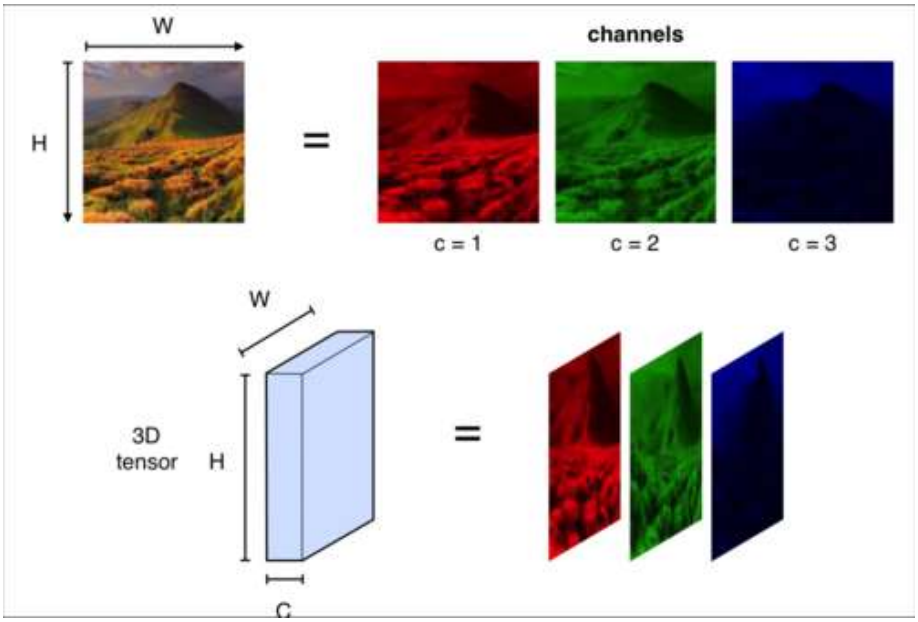

Figure 2: Representation of an RGB image as a 3D tensor. Source: Bursuc, 2017

Once the tensors are generated, comparing patterns becomes a mathematical formula using linear and multilinear algebra. Boiled down, the similarities of values between the ground truth tensor and the new visual image tensor increase the confidence level that the two are the same. However, a single object's tensor can change dramatically with small tweaks, such as changes in color, rotation, skew, occlusion. These changes are referred to in computer vision as feature invariances. In order then for our computer model to be able to handle these invariances, it needs to have representations of these patterns. The need to represent invariances requires that our model has a deep repertoire of templates, which is why CV models are trained using many images.

Object Detection

Computer vision can be broken out into three separate subfields: Classification, which is identifying the label of an object; Object Detection, which aims to identify an objects location; and Segmentation which aims at assigning each pixel to a region. This project will focus on Object Detection.

Figure 3: Subfields of CV, Classification, Object Detection, and Segmentation

Object Detection has the following characteristics:

1. Detect multiple objects within a single visual space.
2. Identify where in the visual field an object is (called localization) most commonly done by drawing a boundary box around it.
3. [Optional] Present the classification label and confidence level for each object.

While most object detection performs point 3 in the list above, classification is not necessarily a requirement. The exclusion of which can be seen most commonly for tasks such as facial detection. As an extra note, the Object Detection example used in Figure 3 is technically considered Classification with Localization, because there is only one object being detected. This is why point 1 is that the model can detect multiple objects.

With an understanding of what Object Detection is, let's move on to learning about some machine learning models that aim to achieve this functionality and what model would be the best to use for our Pokédex implementation.

Background

One of the earliest successful models in computer vision was one called Histogram of Oriented Gradients (HOG). While the concept for this model was around for some time, the first implementation of it in machine learning was done by N. Dalal & B. Triggs in 2005. The first step of this model divides an image into overlapping squared regions, and then computes a histogram of oriented gradients for each cell. While this approach is great at detecting objects that do not substantially deviate from their standard representation, significant changes, such rotation or changes in orientation can cause issues. Also, this model is pretty computationally expensive, as the information that is being stored on each object creates a rather large feature vector, which makes the model rather slow, and expensive to store.

Following HOG, a new model by P. Felzenszwalb, D. McAllester & D. Ramaman came into the spotlight in 2008 called Deformable Part-based Model (DPM). This model built upon HOG with the goal of trying to solve for feature invariances. The primary way it accomplished this was by breaking an object into its parts and using those parts to assist detection. For instance, a bike could be seen as a collection of two wheels and a frame. There would be a representation for the whole bike (coarse filter), and a representation for the parts (parts filter). Another aspect of this model was using a feature pyramid, which changes the image's scale three times. These models fall into a

group of models called traditional detectors, and DPM represented the pinnacle of these models. However, while this model had high accuracy, the same issues of speed and size of the model that hindered HOG still were issues for DPM. It wasn't until there was a shift to deep learning that models started to see significant gains in accuracy and speed.

Deep Learning

Neural Networks consist of an input layer, a hidden layer where all the functions are performed, and a final output layer. However, this single hidden layer, while powerful, is not powerful enough to learn all the features of an image. However, by adding more hidden layers, we can enable a neural network to learn a much richer feature set; this is precisely what Deep Learning is, a Neural Network with more than one hidden layer. There are a multitude of different names for Deep Learning Neural Networks, with the difference between them being the type of activation function used. One type used heavily in CV is that of the Convolutional Neural Network (CNN). It uses a mixture of convolutional layers (convolution is a mathematical term that simply means the product of two functions) and pooling layers (which downsamples the size of the previous layer).

The first area that of success for CNNs in CV was for the application of image classification in 2012 (Krizhevsky, Sutskever, & Hinton). CNNs, while successful in this task, had a hard time being applied to Object Detection. This is because the way a CNN works is it splits an image into a grid of sub-images, scanning each one with a feature called a sliding window. Each sub-image then gets passed into a CNN for feature detection, aka, checks it against its list of templates to determine if anything it is "seeing" matches one of its templates. A softmax function (a function which picks the most confident class) is applied to classify the image. This has

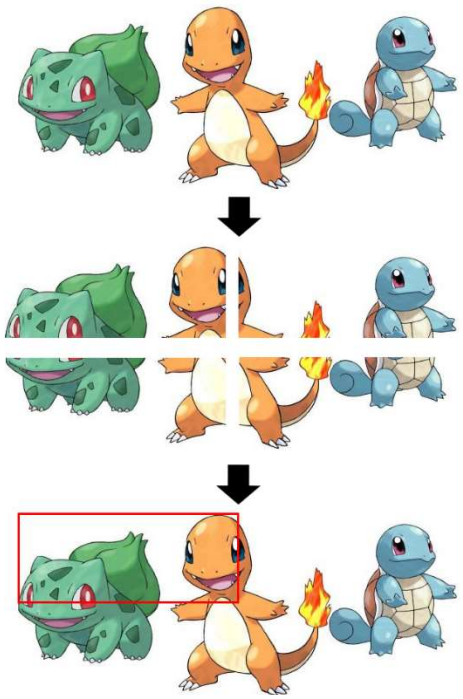

Figure 4: CNN applied to Object Detection

no awareness of where in the cell the object is, it only highlights the entire box. If the object is too small, then highlighting could highlight multiple objects which that are not intended, as seen in Figure 4. The way to solve this is to split the image up into smaller grids. However, the smaller the grid, the more sub-images we have to send through a CNN for feature detection, making it slower. As well, if the object is very large, the highlighted box would be relatively small compared to the object, and the question is then, which box do you highlight.

In 2014, R. Girshick et al. created a version of CNN specifically for the purpose of Object Detection, called Region-based Convolutional Neural Networks (RCNN). The idea here was to identify regions within an image that consist of superpixels, which are groups of pixels with similar characteristics and patterns such as scale, color, texture, and enclosure; this function is called selective search. The model then chooses 2000 regions to highlight, which are called proposals, and sends each one, independently, through a CNN for feature detection and classification. This method was very successful in proving RCNN for Object Detection, as it far surpassed the older traditional methods. However, the process was very slow, as having to run 2000 CNN for one image takes a long time; a single image could take between 20 and 50 seconds. The original author came out with an improvement to this model, which they called Fast R-CNN. This model simplified the process and rather than test each proposal independently, they reduced it down to a single run, or 2000 CNN to 1 CNN. This reduced the processing time of an image down to 2 seconds, a 10x improvement. While this is a significant increase, one of the goals of Object Detection is to achieve realtime detection in video feeds. One of the most significant limiting factors Of RCNN and Fast R-CNN was their use of selective search to find the bounding box area, as it was rather slow.

In 2015, S. Ren et al. proposed the idea for Faster RCNN, which did away with selective search in favor of what it called a “Region Proposal Network.” This model works similarly to how CNN works, in that it scans an image using a sliding window searching for features. However, in this model, fixed-size boxes are used around the center of the sliding window, known as anchor

points, to generate proposals. This model was the first near-realtime deep learning detector (Zou, Shi, Guo, & Ye, 2019), processing an image in .2 seconds, another 10x gain in performance. For frame of reference, .2 seconds is equivalent to 5 FPS, so while it is much faster than its predecessors, in order to count as realtime, it needs to be able to process at least by 30 FPS (the benchmark for most videos).

All the models we have mentioned thus far are considered two-stage detectors. They are named that because the bounding box and feature detection is done separately from the classification. Boiled down, these models are glorified image croppers, which then use a CNN to do classification. It is clear that in order to go faster still, a model that can do everything in one stage is needed.

YOLO

Around the same time that Faster R-CNN came out, an entirely new type of model was proposed by R. Joseph et al. Their model, You Only Look Once (YOLO), reduced the number of stages to a single stage. The way it did this was by breaking an image into a grid of cells. Each cell is allowed to create B number of bounding box predictions. These predictions are tested for "objectness" by comparing priors (templates), which are generated around anchor points and then checking the IOU values. It is important to note that they are not determining what the object is, just determining the likelihood value that the proposed box contains an object. At the same time, each grid cell is computing the class probabilities using a softmax function. When you multiply the class probabilities by the bounding box confidence levels, we get both bounding boxes and classification. This model could achieve realtime processing speeds, reaching up to 30 to 45 FPS. However, one of the biggest challenges to this model was its accuracy in identifying all objects compared to previous models. The reason for this is that if an object is small or overlaps too much with another object, then the softmax "winner take all" function inhibited the detection of other objects.

Subsequent versions of the model (specifically version 3) focused on improving accuracy. To solve for this, several changes were made. First, the softmax function was changed to a multilabel logistic regression, meaning that a confidence score is calculated for all classes in the model. Second, a feature pyramid was introduced, which changes the scale three times (allowing for finer detail detection). Since each new scale adds additional bounding boxes, and the model has to account for all the classes, the final tensor containing all the predictions has increase increased significantly. This reduced speed slightly from version 1. Version 3 seems to strike a balance between accuracy and speed and has proven itself as one of the top realtime Object Detectors; it is for this reason we have chosen to use this model to build out our own Object Detector for our Pokédex.

Methods

Data

The images for this model came from the Kaggle dataset entitled Pokémon Generation One and was made available for use under a General Public License. However, this data was scraped from the internet, so licensing for all the images is unknown and not guaranteed. The dataset is composed of over 10,706 images, with at least 60 images for each Pokémon. The dataset seems to be provided as is, with no validation to the data. Analyzing the images, we found multiple duplicates and wrong labels. It was decided then to manually clean only a handful of Pokémon given the time scope of this project. The final cleaned up data resulted in 845 images with 1135 tagged regions. Resulting in the following breakdown:

- **Bulbasaur: 317 Tagged**
- Charmander: 13 Tagged
- **Dragonite: 83 Tagged**
- Ivysaur: 11 Tagged
- **Magikarp: 109 Tagged**
- Oddish: 2 Tagged
- **Pikachu: 329 Tagged**
- Pokeball: 52 Tagged
- **Psyduck: 111 Tagged**
- **Snorlax: 87 Tagged**
- Squirtle: 14 Tagged
- Venusaur: 7 Tagged

The bolded items are the initial targets that were chosen from the dataset. There are additional tagged items appear because they appear in images that contain the initial targets.

YOLO requires a very specific format for the data to be processed, consisting of the image name, the bounding box area xmin, ymin, xmax, and ymax, as well as the label. Since the original dataset only contained the images, manual tagging of pictures was required. To do this, we used the Visual Object Tagging Tool (VoTT) from Microsoft. This software allowed us to identify multiple objects within a single image, which is why there are more tagged regions than images.

Model Configuration

For this project, we used the YOLO version 3 algorithm. Initial pretrained convolutional weights for the model were acquired from the darknet53 model. Final weights were built after training the model on our Pokémon image dataset. The number of classes = 12. The input size of images was set to 416 x 416. The optimizer implementation is the Adam algorithm using Keras with a learning rate set to 0.001. The loss optimizer is a custom yolo_loss function. Validation set was set to 10% of the images. There are 9 anchor points set to the following x,y coordinates (10,13), (16,30), (33,23), (30,61), (62,45), (59,119), (116,90), (156,198), (373,326). Non-Maximum Suppression is enabled and set to a value of 0.5. The model is trained for 51 epochs with a set of frozen layers, to help get a stable loss. This initial run has a batch size of 32, resulting in a 13 x 13 grid. After the initial first 51, all layers are unfrozen and training proceeds at batches size of 4 for another 51 epochs (104 x 104 grid), or until the EarlyStopping function stops it. EarlyStopping is configured at monitor = “val_loss”, min_delta = 0, patience = 10, verbose = 1. As well, the ReduceLROnPlateau is used to reduce the learning rate on plateaus for fine-tuning. The configuration settings are set at monitor = “val_loss”, factor = 0.1, patience = 3, verbose = 1. EarlyStopping kicked in for our model at Epoch 75 with a loss value of 14.0876 and a val_loss value of 11.5699.

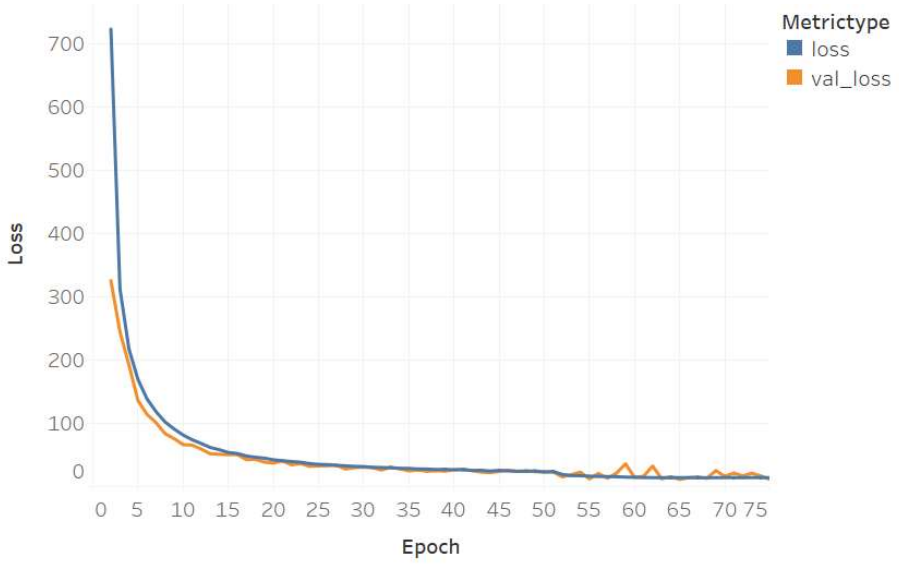

Figure 5: Graph of loss and val_loss across Epochs

Technology

The code was built off a fork of Anton Muehlemann's TrainYourOwnYOLO GitHub project, which is available for use under the MIT license. The code was modified based on the needs of this project, as well as to allow GPU support for TensorFlow. The model was trained in an Anaconda environment of tf_gpu, using Python 3.7.7. In order to get the images and videos processed, a separate build of Python version 3.7.6 was used. There is a fracture in the code where part of it works and then fails, but works in the other build, thus requiring the dual environment.

It was decided to use Keras as the code base as that is what we are familiar with. It should be noted that since YOLO was built using Darknet, which is its own separate library, it is necessary to rely on bridge libraries. These libraries are often coded using specific builds and can make working with the code challenging; highlighting potential challenges when working with Python and code that keeps evolving.

Images were tagged using a locally installed build of VoTT, version 2.1.0.

Results

To test the model, I took a series of photos and videos of my son's stuffed Pokémon toys. By taking the images myself, it ensured that the model could not have been trained on them prior. So let's see how the model fared.

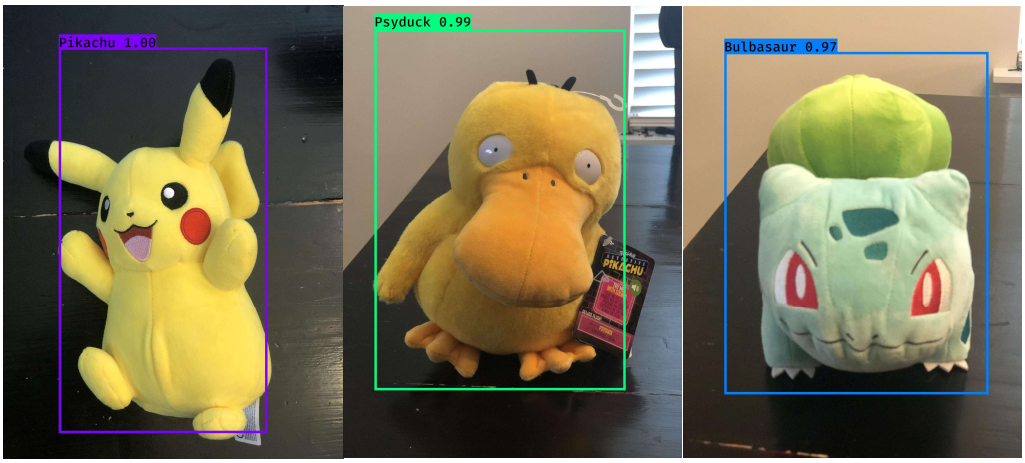

Figure 6: Single Object Detection Results

For single object detection, the model did fairly well, getting all but one image correct out of a total of ten single object images. The one that it got wrong was a Pokéball, which was only trained on 52 images (to which it thought it was a Psyduck, pretty odd).

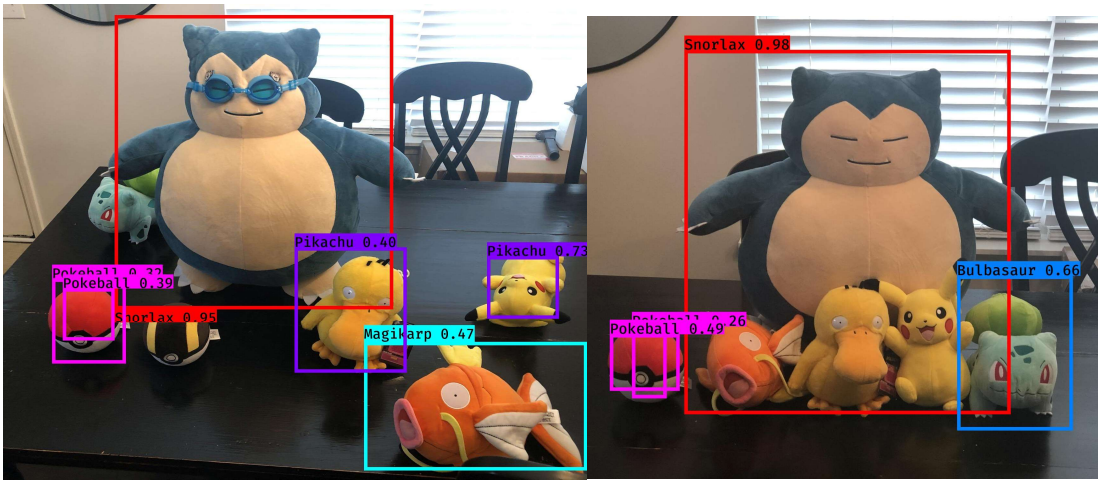

Figure 7: Multi Object Detection Results

When it came to multiple object detection, results were fairly mixed. When objects were separated, as seen on the left in Figure 7, the model was generally able to detect objects with

varying degrees of success. Snorlax seemed to have the most success at being detected, as his bounding box takes over most of the images. This can be seen in the image on the right, as we can see that when the objects were grouped together, it had a harder time recognizing each individual Pokémon. This is due to Snorlax's large bounding box (quite fitting actually for a Snorlax). The only objects that were detected were the objects on the edge of the Snorlax's bounding box, the Pokéball and the Bulbasaur.

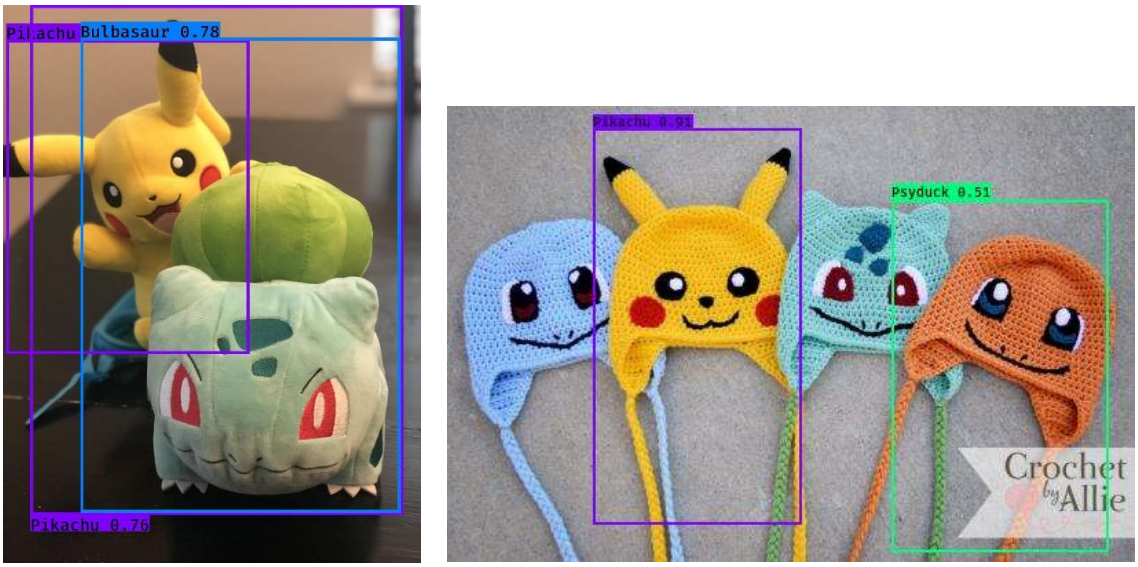

Figure 8: Multi Object Detection with Occlusion and Generalization Results

Testing on multi object detection without a Snorlax seemed to fare better. In the image on the left in Figure 8, we can see the model was successful in detecting the Pikachu behind the Bulbasaur despite the partial occlusion of the Pikachu. However, the overlapping hats in the image to the right, only identified one hat correct, the Pikachu hat.

Discussion and Future Work

Our stated goal was to get a computer to recognize Pokémon, which was demonstrated to be a fairly successful endeavor. Despite not living in the world of Pokémon, the core concept of a Pokédex to recognize Pokémon (or perhaps real animals) by computer sight is not far off. This is made possible through machine learning, specifically in our case, by using the YOLO model. The model did reasonably well at recognizing Pokémon, despite the fairly low level of images trained

on. Increasing the number of training images would surely help to increase accuracy. Since the model seems to be very reliant on the whole body of the Pokémon, perhaps tagging some images that are just parts of the Pokémon could help, such as just the faces, very similar to the Deformable Part Models. Finding the appropriate number of images to train on could be a future goal. One of the other challenges we saw is perhaps a limitation of YOLO itself, and that is overlapping objects. YOLO version 3, helped address this somewhat by moving from one scale feature detection to three scales, however, we can see that the problem still persists. Testing out different hyperparameters could help as well, specifically the batch size, and the number and location of the anchor points.

The primary object was the practicality of creating a Pokédex, however, a stated secondary objective was testing out the Template Matching as a theory for human recognition and computers. The machine learning models covered here seem to be using that approach, but the flaws become pretty apparent. For starters, we have to store a ton of different templates, even for a single object. For computers, space is not necessarily an issue, but the more templates we add, the slower the system takes to process; they are essentially brute forcing a solution. Perhaps the biggest strike against this theory for human recognition is that human's have the ability to recognize an object after only seeing it once or twice, adequately dubbed one-shot learning. There have been attempts at creating computer models that can achieve this, such as seen by a team at Google DeepMind (Vinyals et. al., 2017) on handwriting, Baidu's facial recognition (Ng, 2017), and for offline signature verification (Dey et. al., 2017). While still nascent in their development, one shot models seem to be promising at imitating human recognition abilities. However, human level object detection by computers is still very far off. While we are extremely intimate with our own selves, our own minds remain one of the biggest mysteries.

Acknowledgements

Thank you to my peers and professors in DSC680. This research was made possible with the resources provided by Bellevue University. Also, a special thank you to my son, whose love of Pokémon inspired this project.

References

1. Bursuc, A., Krzakala, F., & Lelarge, M. (2017). Neural Networks, Convolutions, Architectures. Lecture.

2. Dalal, N., & Triggs, B. (2005). Histograms of Oriented Gradients for Human Detection. 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05). doi:10.1109/cvpr.2005.177

3. Dana H. Ballard; Christopher M. Brown (1982). Computer Vision. Prentice Hall. ISBN 978-0-13-165316-0.

4. Dey, S., Dutta, A., Toledo, J. I., Ghosh, S. K., Llados, J., & Pal, U. (2017). SigNet: Convolutional Siamese Network for Writer Independent Offline Signature Verification.arXiv: 1707.02131

5. Dwivedi, H. (2018, August 02). Pokemon Generation One. Retrieved May 06, 2020, from https://www.kaggle.com/thedagger/pokemon-generation-one

6. Felzenszwalb, P., Mcallester, D., & Ramanan, D. (2008). A discriminatively trained, multiscale, deformable part model. 2008 IEEE Conference on Computer Vision and Pattern Recognition. doi:10.1109/cvpr.2008.4587597

7. Geitgey, A. (2018, November 07). Machine Learning is Fun! Part 3: Deep Learning and Convolutional Neural Networks. Retrieved May 29, 2020, from https://medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networks-f40359318721

8. Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. 2014 IEEE Conference on Computer Vision and Pattern Recognition. doi:10.1109/cvpr.2014.81

9. Girshick, R. (2015). Fast R-CNN. 2015 IEEE International Conference on Computer Vision (ICCV). doi:10.1109/iccv.2015.169

10. J. R. Uijlings, K. E. Sande, T. Gevers, and A. W. Smeulders. (2013). Selective Search for Object Recognition. Int. J. Comput. Vision 104, 2 (September 2013), 154–171. doi:https://doi.org/10.1007/s11263-013-0620-5

11. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. Communications of the ACM, 60(6), 84-90. doi:10.1145/3065386

12. Muehlemann, Anton. “AntonMu/TrainYourOwnYOLO.” *GitHub*, 14 May 2020, github.com/AntonMu/TrainYourOwnYOLO

13. Ng, A. (Director). (2017, January 24). Face Recognition demo - Baidu's face-enabled entrance [Video file]. Retrieved from https://www.youtube.com/watch?v=wr4rx0Spihs

14. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. doi: 10.1109/cvpr.2016.91

15. Redmon, J., & Farhadi, A. (2017). YOLO9000: Better, Faster, Stronger. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). doi: 10.1109/cvpr.2017.690

16. Redmon, J. Farhardi, A. (2018) YOLOv3: An Incremental Improvement.

17. Remanan, S. (2020, February 08). Beginner's Guide to Object Detection Algorithms. Retrieved May 15, 2020, from https://medium.com/analytics-vidhya/beginners-guide-to-object-detection-algorithms-6620fb31c375

18. Ren, S., He, K., Girshick, R., & Sun, J. (2017). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. IEEE Transactions on Pattern Analysis and Machine Intelligence, 39(6), 1137-1149. doi:10.1109/tpami.2016.2577031

19. Sharma, P. (2020, May 24). Introduction to Object Detection Algorithms. Retrieved May 17, 2020, from https://www.analyticsvidhya.com/blog/2018/10/a-step-by-step-introduction-to-the-basic-object-detection-algorithms-part-1/

20. Shugen, W. (2002). Framework of pattern recognition model based on the cognitive psychology. Geo-spatial Information Science, 5(2), 74-78. Doi: 10.1007/BF02833890

21. Vinyals, O., Blundell, C., Lillicrap, T., Kavukcuoglu, K., & Wierstra, D. (2017). Matching Networks for One Shot Learning. arXiv: 1606.04080

22. Zou, Z., Shi, Z., Guo, Y., & Ye, J. (2019). Object Detection in 20 Years: A Survey. arXiv: 1905.05055v2