Computer Vision Writeup

Maxwell Spivakovsky

November 2022

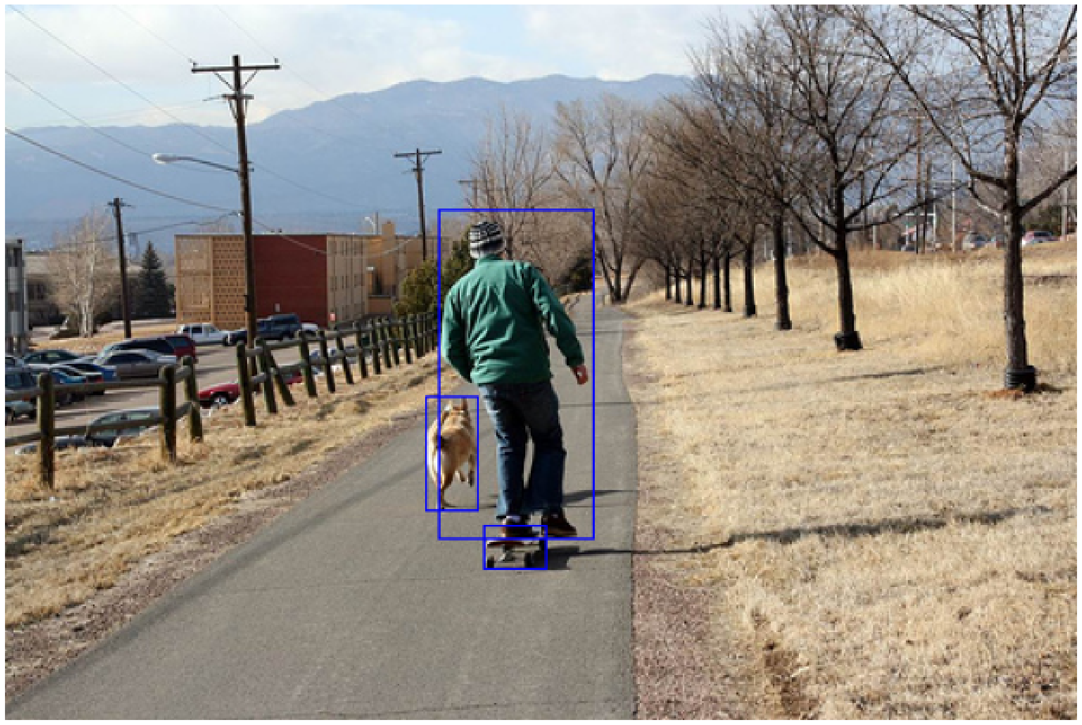
1 Introduction

For my final exploration, I decided to learn about using deep learning in computer vision and use a modern computer vision model for object detection. Through the use of the COCO (Common Objects in Context) data set (which is prominent in the computer vision ML community), I was able to create and train a machine learning model for object detection in images. The COCO data set contains over 330 thousand images, with over 200 thousand labelled images. Images are annotated with bounding boxes (rectangular regions that contain particular objects), segmentation masks (polygons within bounding boxes that define contours of objects), key points (for images of humans), and captions, all of which allow for another level of image recognition (1). The API has tools for visualizing annotations (through the annotation object) and tools for measuring the performance of your model. Users must create their own models, but can run it through images in the COCO data set (eg. validation images), get model predictions, package these predictions in a specific format that is accepted by COCO API, and then use functions from the COCO API that will calculate performance metrics by comparing predictions and the real (known as "ground truth") objects (this will be covered more in the results section.). But COCO does not have any models.

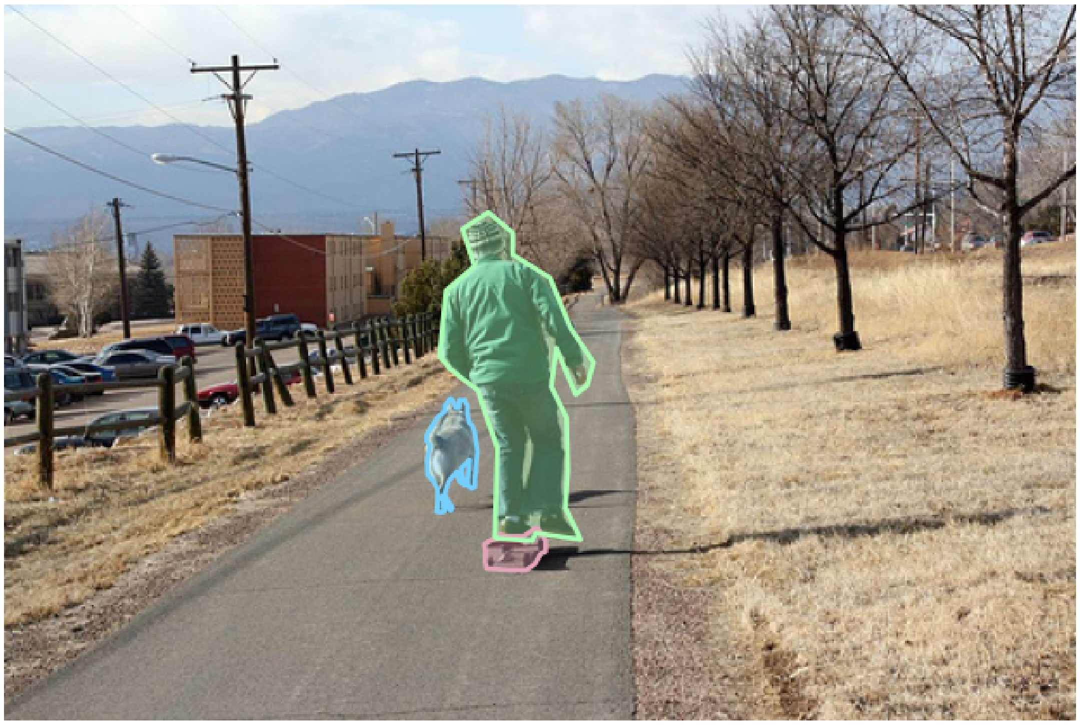
2 COCO

I started by playing around with the COCO API and data set. Objects in images are classified into multiple categories (91 categories are defined but only 80 are present in images) and super-categories. For example, apple and banana (two different categories) are both contained in the food super-category. The COCO API has functions that allow you to display categories and sort by categories. This makes training on specific object potentially much easier. Annotations of images (which contain things like captions, masks, and key points), are what the model is actually trained on. Images are traditionally loaded from the COCO website, but I decided to download and implement my own data set/loader to increase training speed. Additionally, to use PyTorch machine learning libraries on these images, I had to figure out how to efficiently convert the images (originally arrays) into tensors on GPU (the model is relatively slow even on GPU because it is so big, but on CPU it would be impossibly slow). I tested COCO API and TorchVision image processing functionality on one of the images in the validation data set.

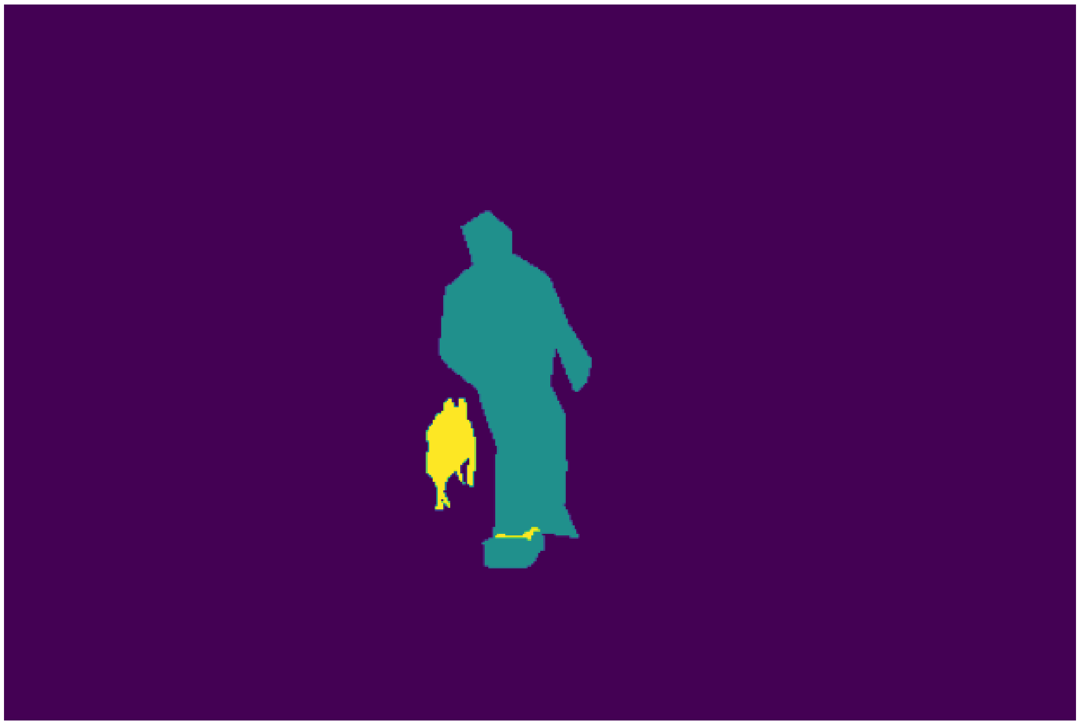
This is one of the many images in the COCO data set. I loaded it with the “io” module and plotted it through the “imshow” function. I then loaded these images with “torchvision”. The benefit of torchvision is that it has a built in function to draw bounding boxes. You can also overlay these bounding boxes over the image to see how accurate they are. The image with bounding boxes is below.



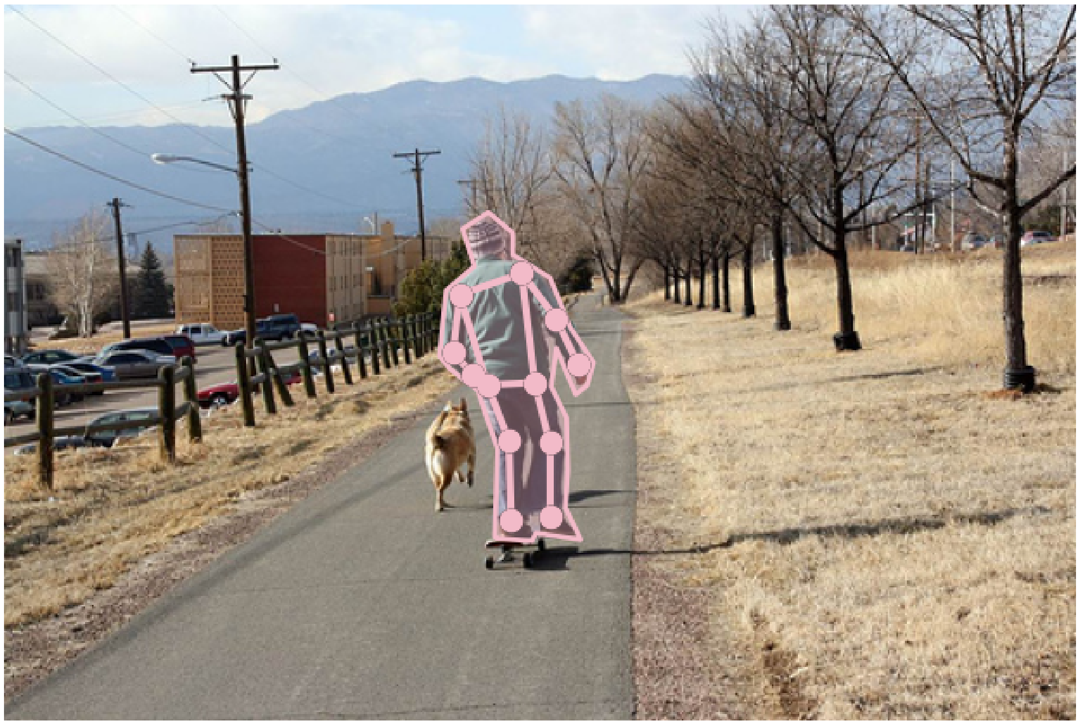
I then loaded the segmentation to visualize how the COCO data set worked and learn more about how masking works.



The COCO API contains a convenient functions that allows the conversion of annotations for an image into its mask. Using this, I was able to display the masks of the objects detected in the image.



As you can see, some of the masks end up overlapping. The bounding box can easily be calculated from the mask. Take the left-most and bottom-most pixels of a given mask to get the bottom left corner of the bounding box. The right-most and top-most pixels give the coordinates for the top right corner of the bounding box. From there the bounding box can be drawn. Like I mentioned early, the COCO API also has key-points on images of humans. While I don't use them in what I did, I still learned how to access them from the annotations. The key-points in this image can be seen below.



The last piece of information contained in the annotations is the captions. This provides the context part of COCO. Each image has 5 different captions that seek to describe what is seen in the image. For this image, the given captions are:

1. A man is skate boarding down a path and a dog is running by his side.
2. A man on a skateboard with a dog outside.
3. A person riding a skate board with a dog following beside.
4. This man is riding a skateboard behind a dog.
5. A man walking his dog on a quiet country road.

3 Dataset, Dataloader, and Model Implementation

The "COCO_Dataset" function is somewhat self explanatory. Images and annotations are loaded in the constructor. When attempting to retrieve an image from its index, everything but the mask is stripped from the annotation file, leaving just the image and its mask to be outputted. This can then be used for training and evaluating the model. Unfortunatley its not that simple. The images in the COCO data set come in widely different sizes. The custom “collate_fn” prevents the images from being stacked into one array (which would cause an error), and instead returns a list of samples and a list of targets for the model.

4 Convolutional Neural Networks

Convolutional Neural Networks (CNN’s) have evolved a lot in the past 10 years. The first Region-based Neural Network (R-CNN) was created in 2013.

4.1 R-CNN

The purpose of the R-CNN is to narrow down the total search area for an object to increase efficiency and efficacy of a machine learning model. The first R-CNN was made up of 3 main parts. The first part was Selective Search, which used a non machine learning algorithm to find the aforementioned bounding boxes (this is where the region part of Region-based CNN comes from). These areas are then inputted into the second part of the model, the actual CNN. The CNN converts the outputs into a vector and makes a feature map from this. These feature maps are trained as a classifier, but were originally too weak to correctly classify objects, thus presenting the need for the third part of the model. The last part of the R-CNN model is the Support-vector Machines (SVM) part. It is similar to a logistic regression and is able to classify feature maps into different objects. This achieves the original goal–detect what area has objects, then use a 2 part algorithm (one uses machine learning and the other doesn’t) to determine what object it sees in this area. Roughly 3 years later the slightly improved Fast R-CNN was created (2).

4.2 Fast R-CNN

The Fast R-CNN essentially combined the second and third parts of the R-CNN. Improvements in CNN’s led to SVM’s becoming unnecessary. One issue that the developers ran into while developing a better CNN was that adding more layers to the model which is intuitively beneficial, made accuracy worse. Thus, ResNet (residual neural network) was born. ResNet followed the usual steps of the layers in a ML model (CNN, batch normalization, then ReLU), but featured an additional step that surprisingly was highly effective. After certain ReLU’s, the original value of the object would be added back. Sections of the model that contained this ResNet format were called “bottleneck building blocks”. These building blocks were combined together to create the model through Feature Pyramid Networks (FPN). FPN’s combine bottlenecks of various sizes into one feature map. for example, in ResNet 50 (which has 50 layers), there are 4 bottlenecks which are essentially 4 different matrices that are applied onto the image. FPN essentially normalizes the size of these matrices to prevent any errors from occurring. The biggest jump in technology was roughly one year latter with the Faster R-CNN (3) (4) (5).

4.3 Faster R-CNN

This new R-CNN combined the first two models in the original CNN into one. Unlike the previous R-CNN's, the bounding boxes were created through its own convolutional neural network called Region Proposal Network (RPN). RPN replaced the previous bounding box algorithms. It worked by creating anchors–sections of the image that could potentially contain objects of interest. Each pixel in the image could be the center of a 3 by 3 grid of bounding boxes. Each of these 9 boxes were designated as anchors and were made of 2 dimensions. The first dimension was box size, which was either 1, 2, or 4 times the default size. The second dimension was the aspect ratio. The possible aspect ratios were 1:1 (square), 3:1 (horizontal rectangle), and 1:3 (vertical rectangle). This RPN was made of 3 different CNNs which each played a different role. The first one was trained to detect the validity of each anchor (how likely is it that there is actually an object in this box). The second one detects the corners of the box. The third CNN classified the region of interest (ROI). It took a 7 by 7 piece of the bounding box and returned what object was inside it. Because of the 2 different targets (boxes and object classification) that both share the same feature map, training was done iteratively (switching between the 2 goals). This allowed the weights of the model to converge to something that allowed both parts of the model to be accurate (6).

4.4 Mask R-CNN

Mask R-CNN added segmentation to Faster R-CNN but had the same capabilities. It does this through another "branch" in the model that takes every pixel inside the bounding box and decides if its part of the object, meaning it should be included in the polygon that is segmentation (7).

5 Results

Average Precision (AP) and Average Recall (AR) are used to measure how strong object detection models are. The confusion matrix is used to calculate both of these, as well as the mean AP (mAP) and mean AR. It essentially compares the predicted object to the actual object. This matrix is made up of 4 possible results. True positive (model labels correctly), true negative (model correctly guess what label is incorrect), false positive (type 1 error, model predicts the incorrect object), and false negative (type 2 error, model fails to predict the correct label). Precision is the proportion of labels that were actually correct (true positive divided by true positive plus false positive). Recall is what proportion of ground truth labels were correctly predicted (true positive divided by true positive plus false negative). These metrics provide different information on how well the model operates. Next is Intersection over Union (IoU), which compares the detected bounding box with the real bounding box (called the ground truth box). This is done by dividing the overlap area by the total area of the two boxes combined. The model returns a score on every image, which is a prediction of its IoU and its precision (essentially its confidence). Different IoU thresholds can be used, which basically makes it so that the model is more stringent with its labelling and bounding boxes. Both AP and AR are calculated for only one category, so we use the means across all objects. AP is calculated at different IoU thresholds for the mAP, with different IoU thresholds given different weight in the calculation.

```
Average Precision and Average Recall

Average Precision  (AP) @[ IoU=0.50:0.95  | area=    all  | maxDets= 100 ]  = 0.369
Average Precision  (AP) @[ IoU=0.50       | area=    all  | maxDets= 100 ]  = 0.585
Average Precision  (AP) @[ IoU=0.75       | area=    all  | maxDets= 100 ]  = 0.397
Average Precision  (AP) @[ IoU=0.50:0.95  | area=  small  | maxDets= 100 ]  = 0.212
Average Precision  (AP) @[ IoU=0.50:0.95  | area= medium  | maxDets= 100 ]  = 0.403
Average Precision  (AP) @[ IoU=0.50:0.95  | area=  large  | maxDets= 100 ]  = 0.478
Average Recall     (AR) @[ IoU=0.50:0.95  | area=    all  | maxDets=   1 ]  = 0.307
Average Recall     (AR) @[ IoU=0.50:0.95  | area=    all  | maxDets=  10 ]  = 0.484
Average Recall     (AR) @[ IoU=0.50:0.95  | area=    all  | maxDets= 100 ]  = 0.508
Average Recall     (AR) @[ IoU=0.50:0.95  | area=  small  | maxDets= 100 ]  = 0.317
Average Recall     (AR) @[ IoU=0.50:0.95  | area= medium  | maxDets= 100 ]  = 0.544
Average Recall     (AR) @[ IoU=0.50:0.95  | area=  large  | maxDets= 100 ]  = 0.645
```

6 Future Projects

I only ran the Fast R-CNN model, so I plan on running the Mask R-CNN model in the future. Beyond this, I am plan on running object detection through videos to give real time information about surroundings.

References

[1] https://cocodataset.org

[2] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation.

[3] Ross Girshick. Fast R-CNN.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition.

[5] Tsung-Yi Lin, Piotr Dollar, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature Pyramid Networks for Object Detection.

[6] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.

[7] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask R-CNN.