Main object Detection Algorithms:

R-CNN – Region Convolutional Neural network. Most modern version is known as Faster R-CNN. This algorithm uses a search function to find potential candidate objects to run object recognition on. Slower, but accurate. Unfortunately, since this algorithm depends on only running the neural network on likely object candidates, it will most likely perform poorly at detection defects which are more similar to corners than objects.

SSD- Single Shot (multibox) Detector. This algorithm works by creating multiple overlapping bounding boxes over the entirety of an image. A neural network scores the bounding boxes and provides a prediction for smaller regions likely to have high class scores, which are evaluated at deeper layers of the network. Faster than R-CNN, but slightly less accurate. An implementation is not yet available for python.

YOLO – You Only Look Once. The input image is initially divided into a grid. Each cell in the grid is used to predict five bounding boxes. A classifier is run for each predicted bounding box, and outcomes with high certainty scores are returned as objects. This is the fastest of the three algorithms, although slightly less accurate in general. However, since the features we are looking at aren’t typical objects this method is likely to work better than R-CNN. A python implementation called darkflow is available. The most advanced YOLO version is YOLOv3, however darkflow has yet to be updated and still runs on YOLOv2.

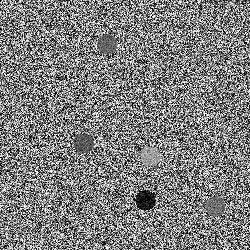
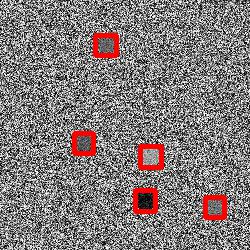
Due to the availability of a python implementation and the non reliance on searching for traditional objects, I have elected to use darkflow for our detection model.

Annotation:

Image annotations must be in PASCAL VOC format. Currently BBox-Label-Tool is being used, which provides a simple gui for drawing a bounding box and outputs labels in the correct format.

Training the Model:

Darkflow has been setup and tested on a set of fake data generated by a python script

Validation has not yet been coded, however visual inspection shows the model is able to successfully label most, but not all, circles present in a grainy image.

The script trainFlow.py will train the network using the image and annotation paths specified in the options section.

Training is very slow on a cpu. Leaving the trainer running from noon Friday until noon Monday yielded ~2300 steps. The generated model should be viable; however the same number of steps can be completed in about 30 minutes when run on a modern GPU.

Running the Model:

runFlow.py will run the model using the image director and weights specified in the options. The output will be an out folder in the image director containing the labeled folders and a text file with a stringified python dictionary containing the bounding box of each detection (format needs improving).

Running the model is viable on a CPU. Labelling 200 circle images takes approximately a minute on a CPU.

Still needs doing:

Model Validation

Image annotations for real data

Streamline setup (creating requirements.txt and GPU setup guide)

Create a system for sharing weights (too big for github)

Potentially expand to allow for multiple classes

Modify runner to load from checkpoint files

Automate cleanup for checkpoints

Cross-platform checks (linux)