Text Detection

The goal of the project is to detect the location of text in an image. This differs from typical image processing projects which aim to classify what object is in an image. Instead of outputting one node representing the final classification, our neural network outputs four final nodes, each representing the coordinate of a bounding box. The bounding box encompasses the instance of text. The data for this project came from the COCO dataset (Common Objects in Context). The images and annotations can be downloaded from the links listed in the appendix. The image dataset is COCO's 2014 training set, and the annotations are from a separate project from COCO to highlight instances of text in the dataset. To clarify, COCO first released a set of images that were annotated with classifications for around 90 common objects. A few years later, COCO released an annotations file with information specific to instances of text within an image. The annotations are formatted with the image name, the four coordinates of a bounding box, and a label. The bounding box coordinates are xmin, ymin, xmax, and ymax. They represent the top left and bottom right corners. The most difficult part of the project was tuning the architecture of the neural network to maximize the accuracy. The results of our experiments are recorded and described below.

Option 1: 3 convolutional layers with 32, 32, and 64 filters
15 epochs: .570
30 epochs: .580
100 epochs: .537

Option 2: 4 convolutional layers with 32, 64, 128, and 256 filters
15 epochs: .545

Option 3: 4 convolutional layers with 128, 256, 512, and 512 filters (EAST)
15 epochs: .52

Option 4: 4 convolutional layers with 32, 32, 64, and 64 filters
15 epochs: .551

Option 5: Same as option1 but with images resized to 128x128
15 epochs: .558

Option 6: Same as option1 but with images resized to 164x164
15 epochs: .559

Option 7: 3 convolutional layers with 32, 64, and 128 filters and images sized at 164x164
15 epochs: .559

Ultimately, option 1 with 30 epochs had the highest scoring accuracy on the validation set. With 100 epochs to train, there is a strong possibility that the model was overfitting the data. This would cause the model to poorly generalize the unseen validation set. The high number of convolutional layers and filters were also likely to cause overfitting as too little data is being processed too much. The most interesting result was the lesser accuracy of option 3. The architecture in this experiment is the same as that for EAST (Efficient and Accurate Scene Text Detector), the most commonly used program for text detection. The high number of filters may work better for the larger dataset that is used to fit EAST, but it was not optimal for the smaller (16,000 image) dataset we worked with. One limit we had to deal with in this project was the computational expense of image processing. In block 12 of the code, we transform a list of image filenames into an array of 2d arrays representing the grayscale pixel values for this image. We resize each input image to 100 by 100 pixels because the full size images would not load on our current hardware. We also chose grayscale representation to limit expense as color is not the most important variable for detecting text. We experimented with resizing the images by 128 and 164 as they were low enough to still work. However, the accuracy on 128 and 164 size images scored lower accuracy than the images resized to 100 by 100 pixels.

Through the experimentation process, I learned that there is job security for AI engineers because there is no one best architecture for any dataset. Deciding the architecture is also much more complicated than it seems. Adding more layers and filters ended up hurting the accuracy. Also, the task of detecting one object in an image is much more simpler than detecting every instance of an object in an image. Unfortunately, the latter is much more useful. I still aim to build an OCR program from scratch so that I can extract the text from a receipt, but there is a lot more to it than just plugging in a neural network. I found some guides that explained how to use Selective Search and Non-Maxima Suppression to detect multiple objects in one image, but I don't yet comprehend it enough to build from scratch.

Appendix:

1. Download the image dataset from: https://cocodataset.org/#download
    a. (2014 Train Images 13gb)
    b. Or download from github: https://github.com/forresterw/Text-Detection
2. Download the annotations cocotext.v2.json from: https://bgshih.github.io/cocotext/#h2-download
    a. Unzip and rename to cocotext.json
3. Ensure that the images are in a folder called train2014 and that the python script "ForresterWelch_FinalProject.py", cocotext.json, and train2014 are in the same folder
4. In that folder, install the following packages:
    a. (Skip this step if the packages are already installed on your computer)
    b. Tensorflow, pandas, numpy, PIL
5. Run the python script from terminal or IDE of your choice

6. The Accuracy of the model against the validation set should be the final output of the script