YOLO for Digit Recognition

Shani Thapa
Machine Learning
Date: 5-10-19
Emails: (thapas2)@students.rowan.edu

***Abstract*—The purpose of this project was to implement a machine learning algorithm for digit detection. Digit detection is the process of image classification for the ten digits of 0-9. The algorithm should be able to take in images of digits and be able to identify all digits in the picture. The data set for the training and validation was provided by the instructor. The student used a YOLO algorithm provided through an open source GitHub repository. YOLO stands for you only look once, and is a real time object detection algorithm. The python code was modified to be used for this project and ran through the images the data provided. The student was able to train the model using small amounts of project data and able to obtain adequate results. Unfortunately, the algorithm required annotation XML that needed the labels of the images. The student could not determine a method to automatically generate these labels for the data. As a result, not enough images could be used for the training of the model to fully predict the dataset given.**

I. Introduction

Digit detection is a fairly difficult problem for computers to resolve. Digits within videos or images can be very difficult for computers to discern for a multitude of reasons. The digits can appear in different sizes, orientations, and parts can be missing. Not to mention, handwritten digits can be extremely different simply due to the person's handwriting. Another major problem is that many digits such as 1 and 7 have very similar shapes so false positive recognition can also occur frequently. Other issues include low resolutions or illumination of the digits in pictures and such. These are some of the problems all digit detection algorithms must overcome. Digit recognition is a very popular ML research topic with many types of algorithm created to reach a high degree of accuracy. A popular dataset for digit detection is MNIST's database of handwritten digits [1]. The goal of the project was to use modern algorithms to be able to recognize all digits within an image. The student found a public GitHub repository using a version 2 YOLO algorithm for digit detection. The author of the code had managed to train the model to obtain fairly impressive results on the SVHN dataset. This code was modified to be used on the dataset provided for this project. The student had to pre-process all the images so they could be used to train this YOLO algorithm. The images had to be reconfigured and reshaped before being accepted. Then, the model was tested on images not used in training. The outputs obtained were fairly decent for the low amount of images used for training. Ultimately, the student was not able to fully realize the YOLO algorithm capacity for digit detection due to not resolving the labeling issue for the images in the dataset given.

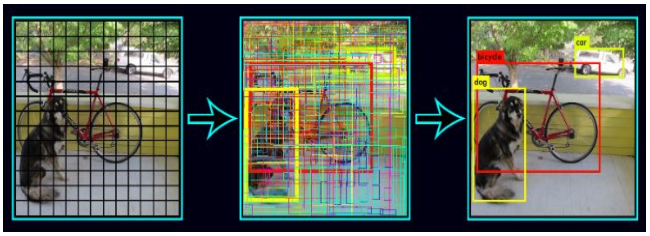

Fig. 1. YOLO Process

II. Methods

A. YOLO

YOLO, you only look once, is an object detection algorithm. Its distinct quality is the speed of its real time processing. There have been three versions of YOLO released, each one increasing performance, adding new classifiers, etc [3]. Unlike other detection algorithm where the model is applied to the image at multiple location, YOLO applies a single neural network to the full image once, hence its acronym [2]. The network divides the image into regions and predicts bounding boxes and probabilities for each region. Another advantage of YOLO is that it looks at the whole image at test time so its predictions are based on the entire context of the image rather than parts. The outputs of YOLO is the image, the object detected in the picture, its confidence in the predictions, and the time required to find the objects. The process for the algorithm can be seen in Figure 1. Thus, YOLO is a state of the art real-time object detection system that is must faster than many of its competitors.

B. YOLO Architecture

The YOLO algorithm found and used was an extremely large model architecture. Figure 2 displayed the total trainable parameters along with the last couple of layers of the model architecture. There were too many layers to practically capture all the model structure in one picture. The model was formed of countless 2-D convolution, maxpooling, batch normalization, and activation layers. The model looks over a 100 layers deep, which may have been excessive for the problem. Nevertheless, the proposed solution would be able to identify multiple digits within an image and potentially all digits in a video feed. The student first ran the algorithm with download pre-trained weights to ensure the code was functioning. Then, the model was retrained using two images provided by the owner of the repository. Then, sample images

```
activation_48 (Activation)      (None, 13, 13, 512)  0          bn5c_branch2b[0][0]
res5c_branch2c (Conv2D)         (None, 13, 13, 2048) 1050624    activation_48[0][0]
bn5c_branch2c (BatchNormalizati (None, 13, 13, 2048) 8192       res5c_branch2c[0][0]
add_16 (Add)                    (None, 13, 13, 2048) 0          bn5c_branch2c[0][0]
                                                                activation_46[0][0]
activation_49 (Activation)      (None, 13, 13, 2048) 0          add_16[0][0]
detection_layer_30 (Conv2D)     (None, 13, 13, 30)   61470      activation_49[0][0]
reshape_1 (Reshape)             (None, 13, 13, 5, 6) 0          detection_layer_30[0]
[0]
==========
Total params: 23,649,182
Trainable params: 23,596,062
Non-trainable params: 53,120
```

Fig. 2. YOLO Architecture Summery

from the project dataset were evaluated by the model. These outputs were placed in the results section.

C. Image Pre-Processing

Training the model required the student to extensively prepare the project data for proper calibration with the borrowed code. The student had to run the chosen images through various pre-processing such as resizing and filtering to be able to train the model. Furthermore, annotating the training images for their proper labels was also required for using them in training. The student initially manually created the XML files that annotated the images to test if the model would train properly. Once the model was shown to be functioning, the student tried to create an algorithm to automatically annotate the images. The locations for each of the digits was fairly easy as each digit were roughly located at the same x and y values. The x values were a little more varied as some digits such as 1 had much smaller widths than others. However, the author was unable to determine how to automatically label the digits of each image without another form of machine learning process. This issue was unable to be resolved.

D. Model Training

The model was trained for 20 epochs with varying batch sizes for all but the last layer of the model at a learning rate of 0.0001. The last layer was trained for 100 epochs or until the loss function did not reduce for 5 epochs. The learning rate of the last layer trained was the same as the rate for the other layers. The model was only trained with 10 images from the dataset. Each of the images chosen were resized to 50x50 due to a keras application error that stated the images must be greater than 32x32 size. The resulting images for digit detection will be shown in the Results section. Ultimately, the inability of the student to get all the proper labels from the dataset caused

Fig. 3. Pre-Trained Model: Training Images

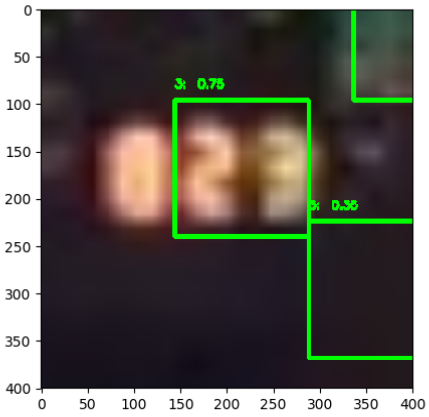

Fig. 4. Pre-Trained Model: Project Image

III. Results

A. Pre-Trained Model

The output of the training images using the pre-trained weights model were shown in Figure 3. The output bounding boxes were very similar to the ones produced by the owner of the repository. The confidence scores were a bit lower though. The project images evaluated from the pre-trained model was shown in Figure 4. Various errors were present in the output of this model. There were two false positive bounding boxes that detected digits in the two corners of the image. Various tested images caused the bounding box at the upper right corner to always appear. There did seem to be a shadow of a number in many of the pictures in the set. The model predicted a correct digit but the region of detection overlapped with another digit it failed to recognize. Other images also had a similar problem along with predicting the wrong digit as well. Various improvements were made after the project data was used to train the model as will be shown.

B. Proper Trained Model

Figures 5 showed the results of the training on the training images. The training was very good as all the digits were

Fig. 5. YOLO Training Images

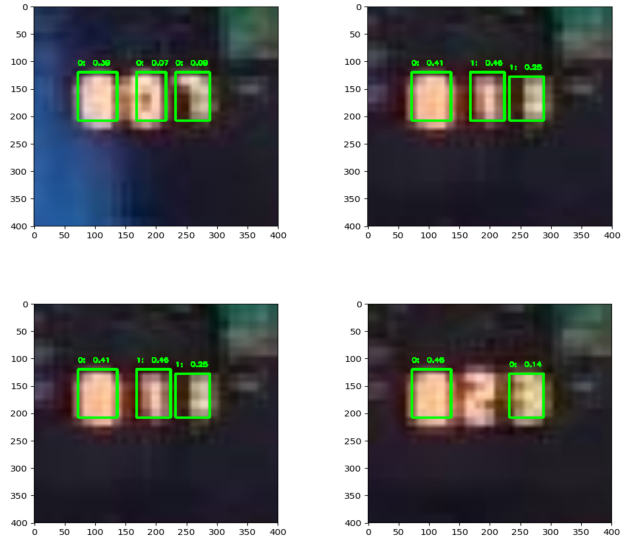

Fig. 6. YOLO Testing Images

recognized and each detected had a very high confidence score. However, only a small amount of images were trained so these results were fairly pointless. Figure 6 displayed some validation images that were evaluated. These outputs were very bare bones as the training had been so insufficient. The model was able to almost always detect all the digits as every testing images had all three digits detected for nearly all of them. This was done by reducing the threshold value for the digit detection. Since the model was initially designed for near 200x200 size images, the original threshold value was reduced from 0.3 to 0.06. This value seemed to give the most correct amount of digits detected, any lower would cause many more false positive digits to be detected. However, the accuracy was very poor and the confidence of each prediction were also very low. Many of the digits detected tended to be predicted to zero since that digit was the one most most encountered by the model during training. This was due to the first digit in every image in the dataset starting with zero. Not much meaningful data was obtained from the model due to the lack of labels preventing the usage of the most of the data in the model training.

IV. Discussion

There were many issues encountered during the project. One of the biggest was just being able to initialize and run the code obtained from the public GitHub repository. Many packages and dependencies were required. Furthermore, some of the packages had to be downgraded to earlier version since some of the new version would results in errors. Computing resources were also a big limitation. Running the code to train the model required an extremely long time on the CPU, thus a method to utilize the GPUs of the computer had to be realized. The student was unable to utilize the Google Cloud VM due to Google not allowing GPU to be given to the VM for some reason. Only when the there were zero GPUs requested was the VM allowed to be used. Thus, the student's personal computer utilized the NVIDIA GPUs through the CUDA and cnDNN libraries. These libraries were also quite complicated as certain each library only worked with certain version of the other. Furthermore, the version of the libraries downloaded also conflicted with many up to date packages. For example, python 3.7 which is the latest version needed to be downgraded to python 3.5 for the NIVDIA libraries to function for the ones the student downloaded, etc.

After the initialization and setup, the largest issue by far was the data-set. There were many problems encountered due to the data. First, none of the data was labeled and caused a massive roadblock. The student tried multiple methods to label all the images but failed to resolve the issue. Second, many of the images were unusable due to the numbers being impossible to discern. This resulted in the student having to manually go through the dataset to ensure the chosen images used were suitable. The small image size and resolution gave the model numerous problems. The images had to be resized larger since certain keras applications requiring the image size to be minimum 32 by 32. This project helped truly cement to the student how much the data preparation was vital to all machine learning processes. No meaningful results were obtainable if the data has not been properly prepared or organized.

V. Conclusions

The purpose of this project was to utilize modern machine learning algorithms for the purposes of digit detection. The topic is a fairly popular and well researched subject in the field of machine learning. There are many problems that algorithms need to be overcome to achieve a high accuracy of digit recognition. Some of these issues are: different sizes and orientation of the digits, missing or cut off parts, similar shapes between numbers, and illumination or poor image quality. Some of these issues popped up in the dataset used for the project. The student found a public GitHub repository that used YOLO for digit detection in natural scenery. YOLO, you only look once, is a modern real-time object detection algorithm with a heavy emphasis on speed. The code was taken and remodeled to be used for the FAA Digital Gauge dataset. Once the dependencies, packages, and environment was setup as instructed the student ran the model on pre-trained weights. After confirming the output on given testing images, the

student ran the project data with the model. The outputs were fairly poor so the model needed to be properly trained. The main task that the student was forced to accomplish was to pre-process the images so they were properly formatted for the algorithm. The author was able to get a model trained with some project data images and obtain somewhat adequate results with the trained model. However, the model required much more training images. XML files that annotated and gave information about the images were required to train the model. Various methods to automatically generate the XML files were tried but a method to generate labels on the data was unsuccessful. The main challenges of the project were the initialization steps to start running the code and the project data-set. The student was able to fully realize the importance of good data and its necessity in machine learning processes.

REFERENCES

[1] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[2] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. *arXiv preprint arXiv:1612.08242*, 2016.

[3] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

The GitHub link to access all the code for the project is: https://github.com/thapashani6/ML-Digit-Detection-