Project Submission: Machine Learning

Spencer Buebel **Nick Dyben** **Charlie Kelley**

Abstract

This is a final report on the Machine Learning project focused on Image detection of firearms in still images. In this project we applied a CNN and TensorFlow in order to achieve that goal. These following sections will go over the following stages of our project: The data set we used, the basics of how image classification through bounding box overlays operate, how we classified our data, the algorithm we implemented, how Tensor Flow assisted our implementation, our results, and our conclusions.

1 Acquiring a Data Set

We started with a Kaggle dataset comprised of 333 images with one or more guns. These images are taken from movies and Google images. These images are unlabeled. The specific repository for this data is:

`https://www.kaggle.com/issaisasank/guns-object-detection`

We have drawn bounding boxes on top of a subset of images in the dataset to show that the labels are, in fact, accurate, and we have split the data into training and testing data so that we can test the validity of our model.

1.1 Bounding Box Overlay

The labels provided give the number of guns in each image, and two corners that define the rectangular bounding box for each gun in the image. We start by writing a script to parse these labels and draw rectangles on top of the first 5 images to ensure the labels are accurate. The following figures show some examples of bounding boxes around guns in our labelled dataset.

Figure 1: Image with bounding box

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

Figure 2: Image with bounding box

Although we have proven our dataset's labels are valid, we must conform to the .xml and .csv label formats required by Tensorflow for training. We leveraged a tool called LabelImg to help expedite the process of generating a well-formatted .xml file for each image that includes the image dimensions, the locations of the gun(s) in the image, and the file name. The final step in the data preprocessing was to run an open source Python script to generate .csv files for the training and testing sets of images to concisely summarize the contents of all the .xml files.

1.2 Training and Testing Data

After verifying the validity of our data and labels, we separated a subset of data for testing and one for training. The test data is important so that we can verify the accuracy of our CNN model after training. In our Jupyter notebook, we designated the first 300 images and labels for testing, and the remaining 33 images and labels make up the testing data set. Since the images are already randomized, there is no need for shuffling. The following figures show the size of the test and training data sets are as expected.

Now, separate training and testing data. This is straightforward.

In [20]:
```
training_data = data[:300]
training_labels = labels[:300]

test_data = data[300:]
test_labels = labels[300:]

print(len(training_data), len(training_labels))
print(len(test_data), len(test_labels))
```

```
300 300
33 33
```

Figure 3: Dimensions of training and test data are correct.

2 Single Shot Detector Algorithm

2.1 The Chosen Algorithm

We use the term single shot detector, SSD, to refer broadly to architectures that use a single convolutional network to directly predict classes and anchor offsets without requiring a second stage classification operation. It is significantly faster in speed because it ignores this second classification stage. High detection accuracy in SSD is achieved by using multiple boxes or filters with different sizes, and aspect ratio for object detection. The different sizes allow for easier detection of different sizes of objects without the second pass over the image. It also applies these filters to multiple feature maps from the later stages of a network.

The specific SSD model we chose has a base VGG-16 network followed by multi-box convolution layers. The VGG-16 base network for SSD is standard for most CNN architectures for high quality object detection. Prediction for the bounding boxes and confidence for different objects in the image is done not by one but by multiple feature maps of different sizes that represent multiple scales.

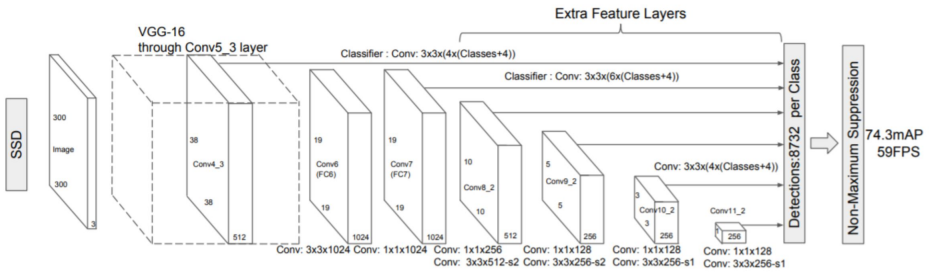


Figure 4: Single Shot Detector Architecture

Utilizing the SSD algorithm instead of the LeNet algorithm was a decision we made due to the fact that the SSD algorithm was faster than the LeNet algorithm while also being able to achieve higher levels of accuracy. It's a significant upgrade from the base LeNet algorithm, which has been surpassed several times in the last decade.

3 Tensor Flow

TensorFlow can help to build neural network models to automatically recognize images. These are typically Convolutional Neural Networks(CNN) but can start from several different base models. The TensorFlow API provides the tools with which to train a SSD, CNN, and R-CNN algorithms. Some of these models are trained out of the box on a given database and with many common labels.

3.1 Training the model

For our project we selected TensorFlow's trained SSD model that used a custom data set called "coco." The coco, or common objects in context, data set is a compilation of images of very common objects - such as dogs, people, or chairs. Unfortunate to our project, the coco data set does not come trained with any notion of a firearm; however, TensorFlow allows the ability to do custom training on top of the training they have provided. With this knowledge, we provided TensorFlow with our data set of firearms and allowed it to train an SSD model on top of the "coco" model. Each layer of training modifies the model's prediction map. Each location in this map stores class confidence and bounding box information as if there is an object of interest at every location. This causes a lot of false alarms so another process selects a list of the most likely prediction. This is, in a sense, how TensorFlow helped us to identify firearms in a frame.

Training was done locally on an AMD Ryzen 7 3800x meaning training was relatively slow compared to the possibilities that TensorFlow allows with CUDA. We had trouble getting TensorFlow to recognize our NVIDIA GPU, even with CUDA installed, and instead just trained on the CPU. TensorFlow has a great training method where it stores checkpoints after so many iterations of training, which makes training a process which can be interrupted and restarted from a checkpoint. Similarly, if training never converges below a threshold error, checkpoints are still saved, meaning the model obtained can still be used, even if the training algorithm never finishes. Initial training loss was 13.0724, and decreased below 5.0 through about 200 steps of training, with each step taking between 3.75 and 4.25 seconds to complete.

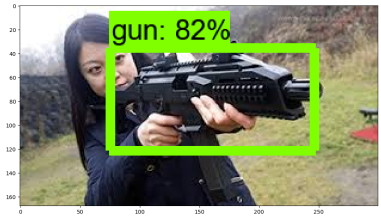
4 Results

After training our model on top of the SSD "coco" model, we ran prediction algorithm on our 33 test images. The following figures summarize our results. Overall, results were good, with many correct classifications, some missed classifications, and fewer false, duplicate classifications. We believe the erroneous results can be explained by a small training dataset (300 images), and poor labelling. Since labelling was a time-consuming, tedious endeavor, it is likely that the labels we generated with LabelImg were less precise than we would have liked, limiting the performance of our trained model.

(a) Great result - two guns

(b) Good result

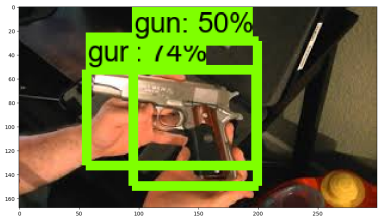
(c) High confidence



(d) Good result

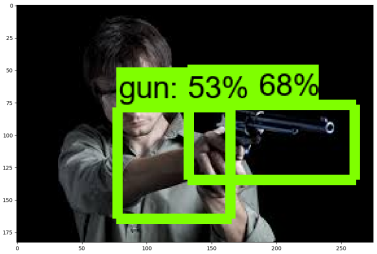
Figure 5: Ideal Results

The results shown above demonstrate cases where detection occurred and was correct. Image (a) shows that the model is able to identify multiple guns within an image, and the remaining images show that multiple types and sizes of guns are able to be detected. The model can sort through different backgrounds and lighting conditions, and still recognize guns. The results also show that our model is able to tell us how confident it is that a gun is present in an image. If this tool was going to be used for law enforcement, this statistic would be very important, as we would want to avoid false alarms, and the threshold confidence for notification could be tuned.



(a) Multiple detection

(b) One detection - two guns



(c) Multiple detections

(d) Missed gun

Figure 6: Sub-Ideal Results

These next results show some less ideal classifications. Images (a) and (c) show examples of false duplicate classifications. In our test dataset, these were the only two examples of such false classifications, and this is not a horrible result, because a gun was still present in each image, and both guns were still detected. Image (b) shows an example where both guns are grouped into one bounding box. Again, this is not the worst possible mistake, since for our application, it would still notify law enforcement appropriately, and manual inspection of the flagged image would clearly tell the authorities more information. Image (d) recognized the gun on the left, but missed the gun on the right. This is likely because of the unusual angle the rightmost gun is being held, where most of the shape is obscured. Perhaps this missed classification could have been rectified with a larger dataset including more guns in different orientations, since our training dataset only held 300 images.

Figure 7: Incorrect Results

Finally, these results show the worst results of our model. In these two cases, the guns in the images were not detected or located by the model, which would definitely be a problem for a safety-critical system. Again, we attribute this type of error to a small dataset and inaccurate labels.

These Results could continue to improved with a larger dataset, more accurate labels, and implementing a higher batch cross validation for testing. If we applied a higher batch cross validation for testing, it would avoid the possible overfitting caused by the small dataset used to train the model.

5 Conclusion

Based on the results, our project was an excellent success. We obtained superior than expected results, most likely due to the optimization of the SSD algorithm, and have a clear grasp about how we would further the project beyond this point. With enough advancements, we could not only improve the accuracy of the model in subpar testing data, but potentially expand to having several classifiers to identify types of firearms. This would have far reaching consequences for law enforcement, being able to identify situations with more informed threat levels.

6 Contributions

- Spencer: Worked on data pre-processing and fitting bounding boxes on images. Split data into training and test data.
- Nick: Researched specific CNN algorithm we will use, and wrote up results.
- Charlie: Researched and found initial Kaggle data set and thought about our next steps.

7 References

[1] H. S. Chatterjee "Various Types of Convolutional Neural Network," Medium, 24-Jul-2019. [Online]. Available: https://towardsdatascience.com/various-types-of-convolutional-neural-network-8b00c9a08a1b. [Accessed: 22-Oct-2020].

[2] C. Li, "Tips for implementing SSD Object Detection (with TensorFlow code)," Lambda Blog, 10-Feb-2019. [Online]. Available: https://lambdalabs.com/blog/how-to-implement-ssd-object-detection-in-tensorflow/. [Accessed: 03-Dec-2020].

[3] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy, "Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

[4] TensorFlow, Object Detection, (2020), GitHub repository, https://github.com/tensorflow/models.git