Object Detection in TensorFlow

Nissim Cantor, Avi Radinsky, Jacob Silbiger

GitHub: https://github.com/ndcantor/tensorflow-street-classifier

Our Mentor, Gershom Kutliroff

- Chief Science Officer, Taranis
- CTO & Founder, ClearVuze
- Principle Engineer, Intel
- CTO & Founder, Omek Interactive
- Chief Scientist, IDT Video Technologies

Linkedin

Goal

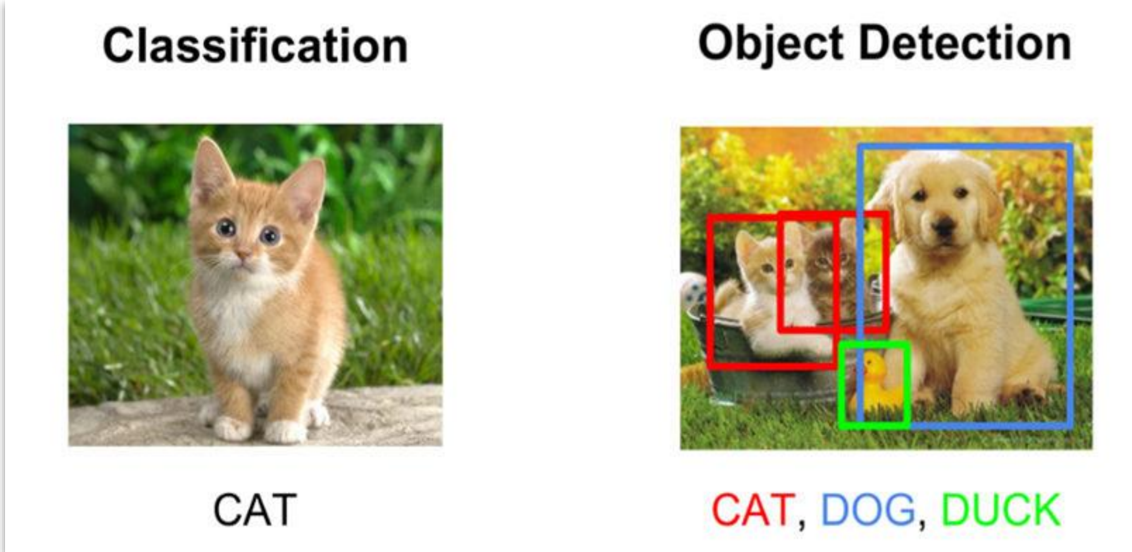
- The goal of our project was to create an image classification and object detection model that simulates how one would be used in a self driving car
- Our model uses a neural network to draw boxes around and label objects in an image
- Brief overview of model architecture:
  - TensorFlow transfer learning image classifier
  - Object detection using:
    - Selective search
    - Non-maximum suppression

Background

Image Classification vs. Object Detection

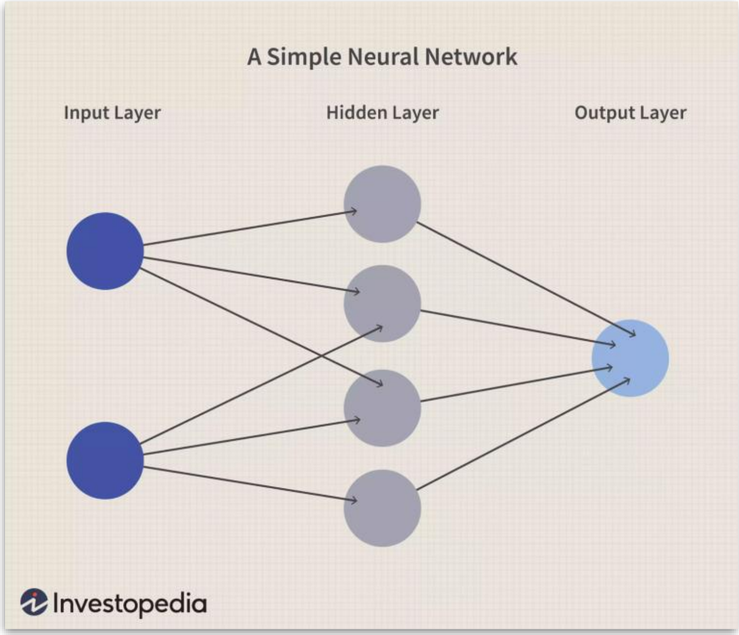
- Image classification - determining the classification of a given image (ex. Is this a photograph of a cat or a dog).

- Object detection - detecting the an image of a certain class within a larger picture (ex. Within this picture of many animals, draw a bounding box around all relevant animals and identify them.)



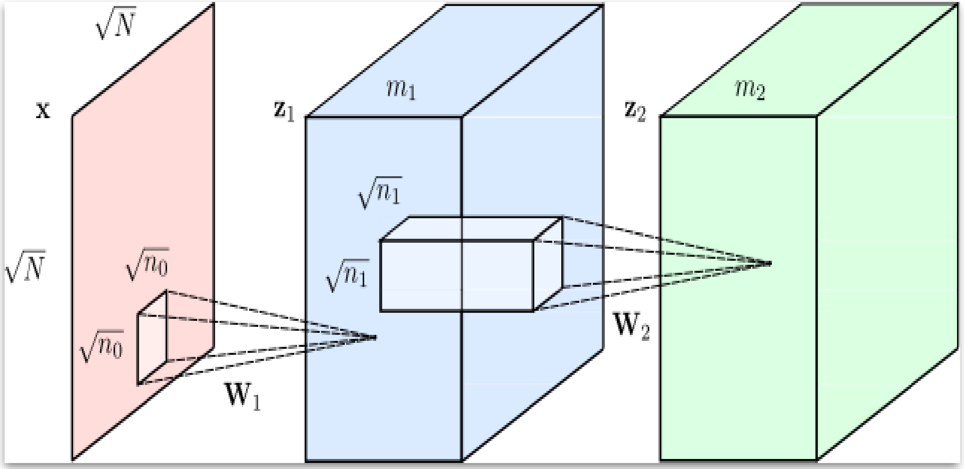
Neural Networks

- “A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. In this sense, neural networks refer to systems of neurons, either organic or artificial in nature.” (investopedia.com)
- It turns out that neural networks often produce generalizable results.
- Its best to view them as a black box.



Convolutional Neural Networks

- A class of Neural Networks commonly used in computer vision applications

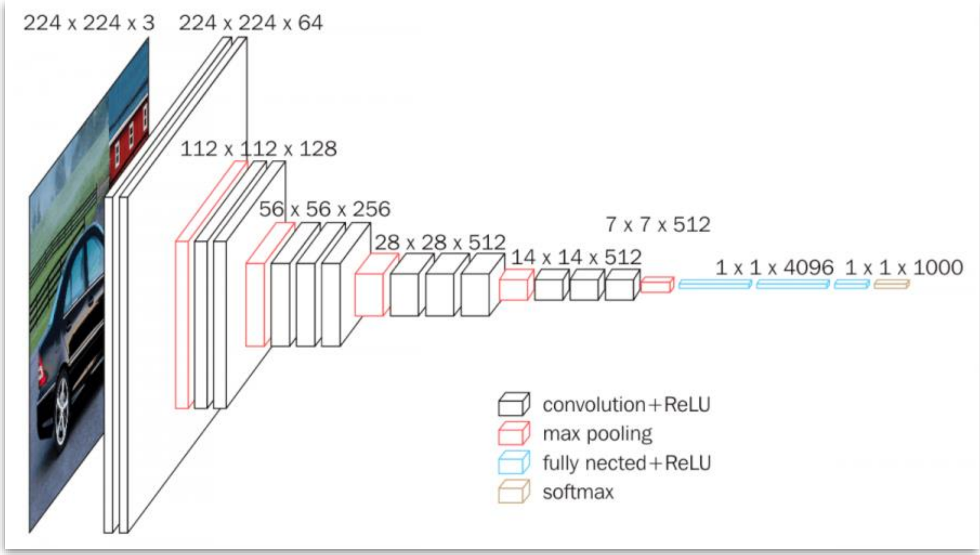


Wikipedia

VGG16 (Simonyan and Zisserman)

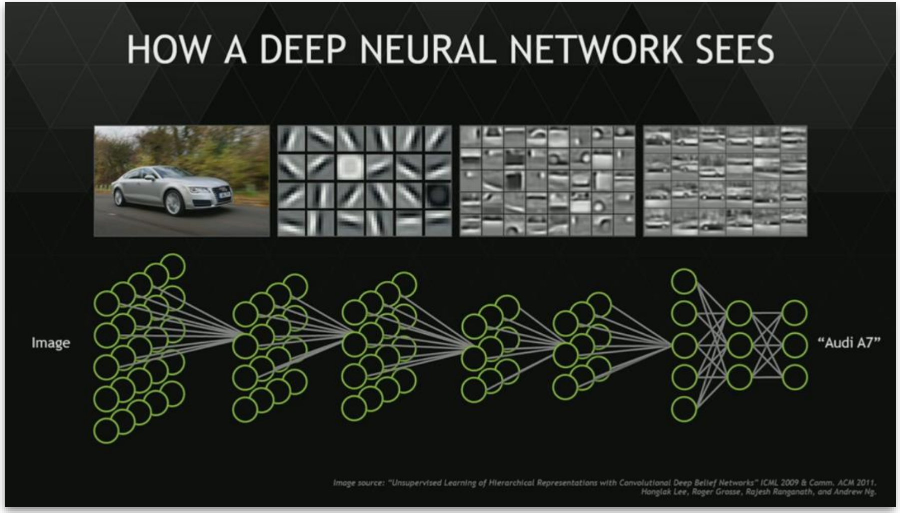
- A CNN architecture developed by Oxford University researchers that showed high rates of success in image recognition.
- It showed that 'deep' Neural Networks might be more effective than 'large' neural networks.

Wiki



But how does a Neural Network actually work?

- Currently, the prevalent theory as to how a neural network works is that each layer is detecting a specific characteristic of the image that it sees, and as you travel through the layers of the network the layers begin detecting increasingly complex characteristics.

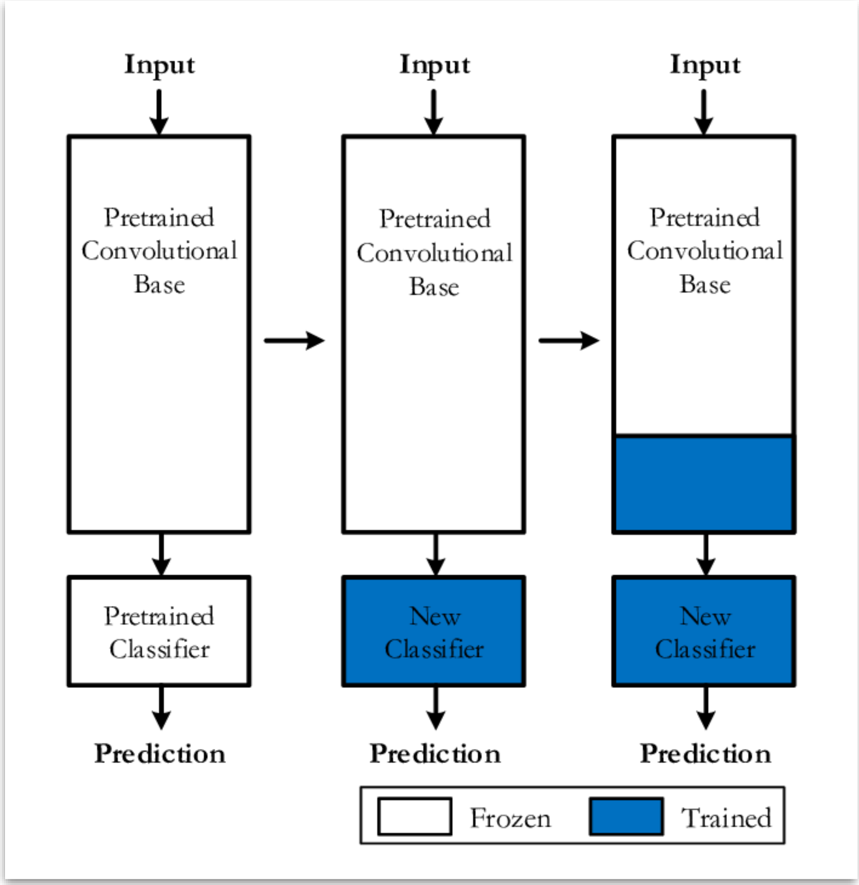


HOW A DEEP NEURAL NETWORK SEES

Image source: “Unsupervised Learning of Hierarchical Representations with Convolutional Deep Belief Networks” ICML 2009 & Comm. ACM 2011. Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Ng.

Transfer Learning

- The characteristic detecting nature of neural networks allows us to leverage pre-trained networks on different problems with minimal changes.
- The changes can be as minimal as retraining the last layer in the network but sometime can involve training several or more of the last layers in a network.



Fine Tuning

Through tweaking the parameters of a neural network, one can try and maximize its accuracy. However, each tweak of the parameters has its positives and negatives.

- Running through the data numerous times (multiple epochs)
    - It can make up for small data sizes (+)
    - It can cause overtraining in the model (-)
- Increasing the size of the neural network
    - Accuracy can increase (+)
    - It becomes easier to overtrain the model (-)

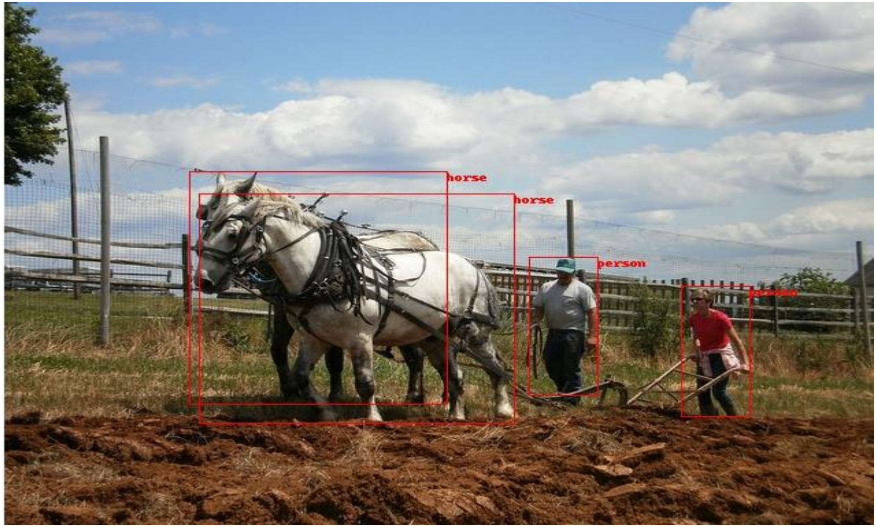
TensorFlow Street Classifier

Data

- Since we wanted to simulate a model that might be used in a self driving car, we decided to train our network to recognize everyday objects found in city streets
- Classes:
  - Bicycle
  - Car
  - Motorcycle
  - Person
  - Train

Datasets

- We used the COCO (Common Objects in Context) Dataset as it contains high-quality images of these objects in the real-life scenes.
- We also used Open Images V6 Dataset to supplement some more images for the small sized classes.
- Both datasets contained labels and bounding boxes which we used as ground truth to train our model.

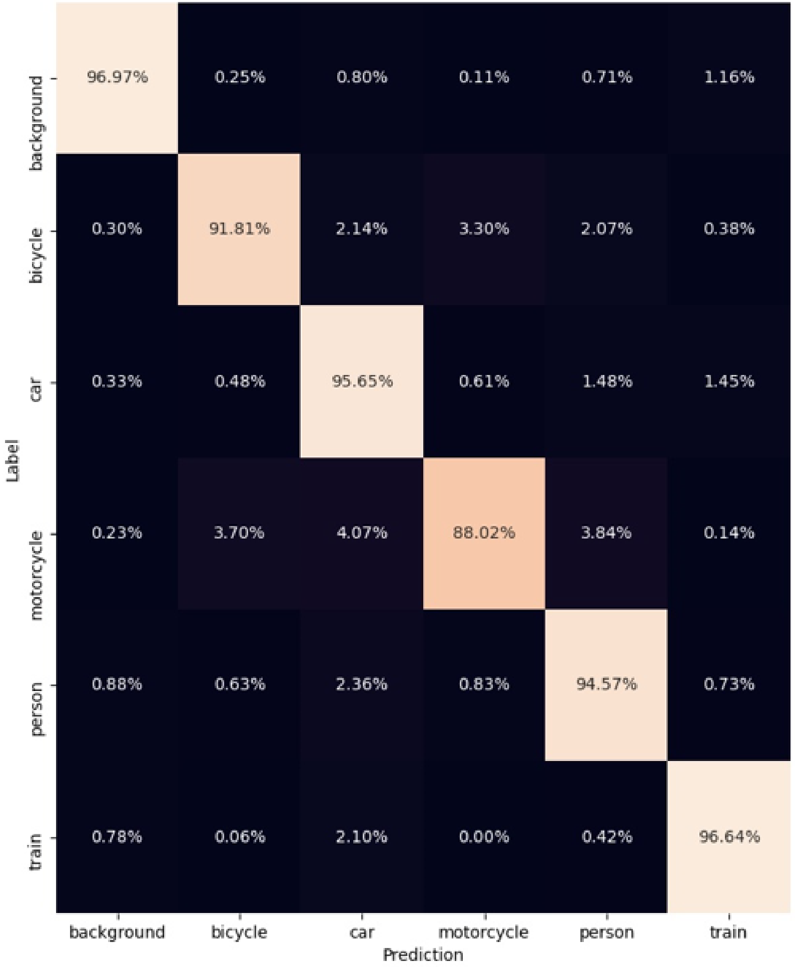


Generating Train/Test Images

- All bounding boxes from every image in the datasets were cropped and saved inside of its class folder (crops of cars were stored in the ‘car’ folder, etc)
- These cropped images were then split into train and test folders
- Shift augmentations were performed for crops in the train folder in order to help the model learn more about the features of each class.
- These crops were then used to train the model
- Any image from the datasets which had no bounding boxes of the classes we were training the machine to recognize were sliced into quadrants, with each quarter being placed into a ‘background’ class folder
- The purpose of the background class is to allow the model to classify unknown objects as ‘background’ instead of mistakenly classifying it as one of the five other classes.
- In total the model was trained on over 848,000 cropped images

Classification Model

- Our model uses transfer learning using the MobileNetV2 architecture
- The model ran for 20 epochs, the first 10 with a learning rate of 0.0001 and the last 10 with a learning rate of 0.00001
- After training, our model obtained a score of over 95% test accuracy
- The figure to the right is a confusion matrix, showing how accurate the model was for each of the 6 classes



	background	bicycle	car	motorcycle	person	train
background	96.97%	0.25%	0.80%	0.11%	0.71%	1.16%
bicycle	0.30%	91.81%	2.14%	3.30%	2.07%	0.38%
car	0.33%	0.48%	95.65%	0.61%	1.48%	1.45%
motorcycle	0.23%	3.70%	4.07%	88.02%	3.84%	0.14%
person	0.88%	0.63%	2.36%	0.83%	94.57%	0.73%
train	0.78%	0.06%	2.10%	0.00%	0.42%	96.64%

Object Detection

- After classifying each image, the next step is object detection
- Object detection works by sending image crops into the classification model and drawing boxes around all crops that were classified as one of the non-background classes
- We tried two different methods of generating crops to send to into the classifier:
    1. Sliding windows
    2. Selective search

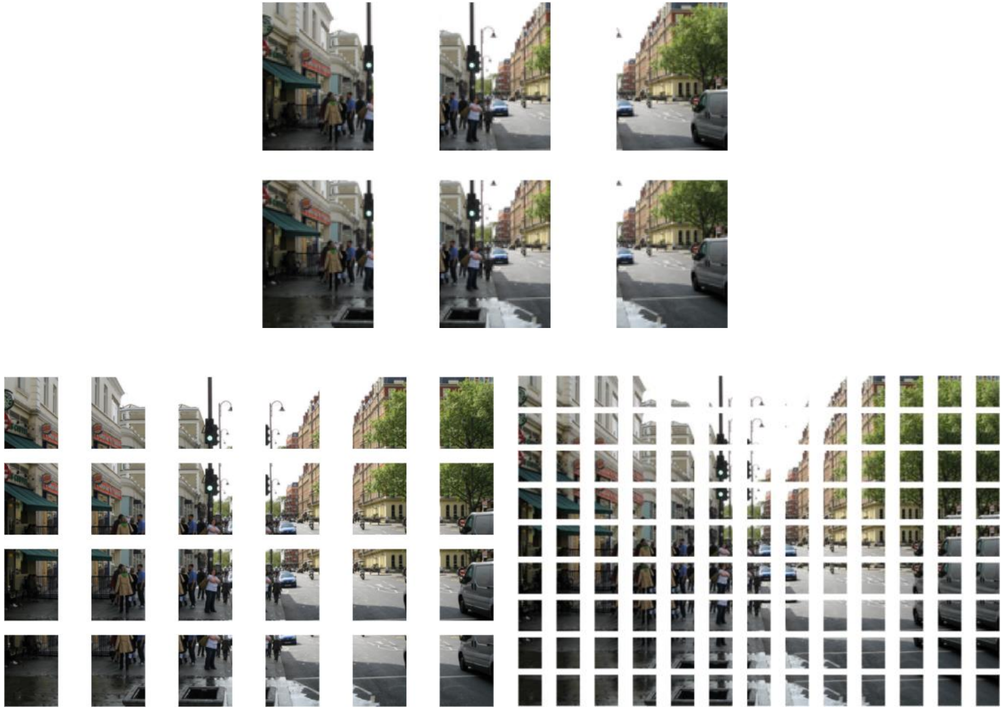
Sliding Windows

- Sliding windows works by passing a fixed-size rectangle across the image, cropping the image and saving the crop’s location on the original image. The image is then enlarged and the process is repeated in order to generate smaller boxes to help find smaller objects.
- The process of repeatedly enlarging the image is called pyramid scaling
- The crops are then sent to the classification model to be classified. The model outputs the original image with bounding boxes drawn around all crops which were classified as non-background objects

Example

Original Image

Image crops with pyramid scaling



Selective Search

The sliding windows method had two main problems:

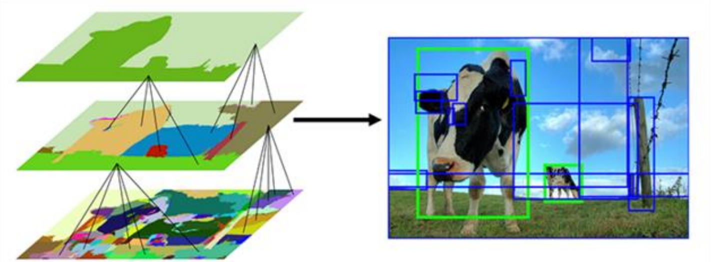
- It is a brute force method, so it took a long time to run
- The boxes all had a fixed size, so it was hard to classify multiple classes of objects that had different shapes

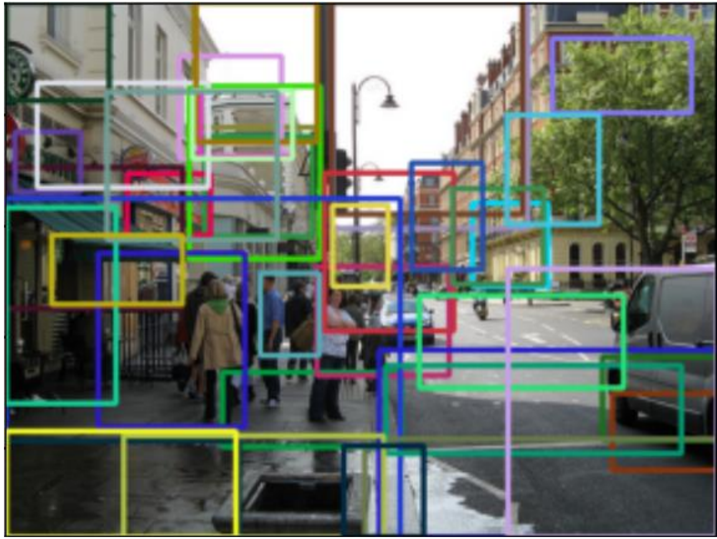
We decided to use an alternative method of generating bounding boxes...

Selective search:

- An algorithm that works by creating crops based on certain criteria in the image:
  - Color, texture, shape, size
- This produces boxes of different shapes and sizes, and doesn’t work by repeatedly running brute-force across the image
- Crops are then sent to the model to be classified
- Our model uses selective search as it is faster and more accurate than sliding windows

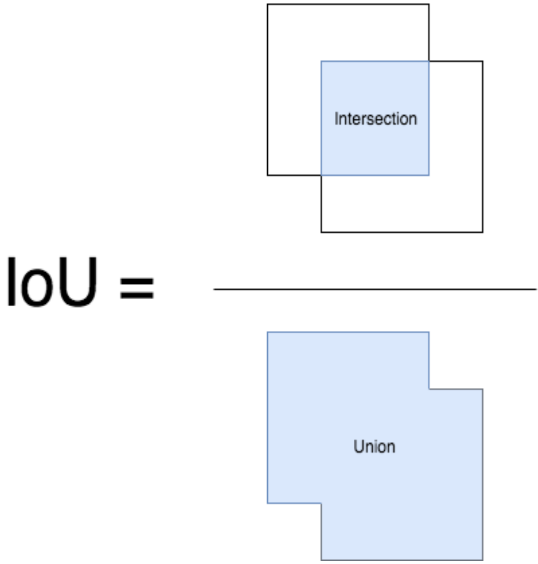
(Diagram from pyimagesearch.com)



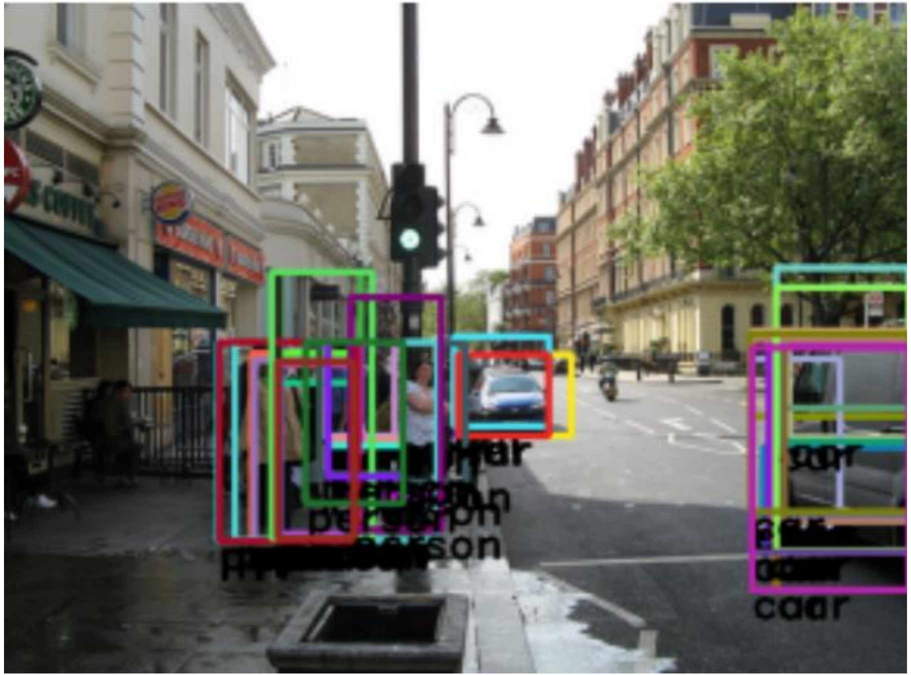


Non-maximum Suppression (NMS)

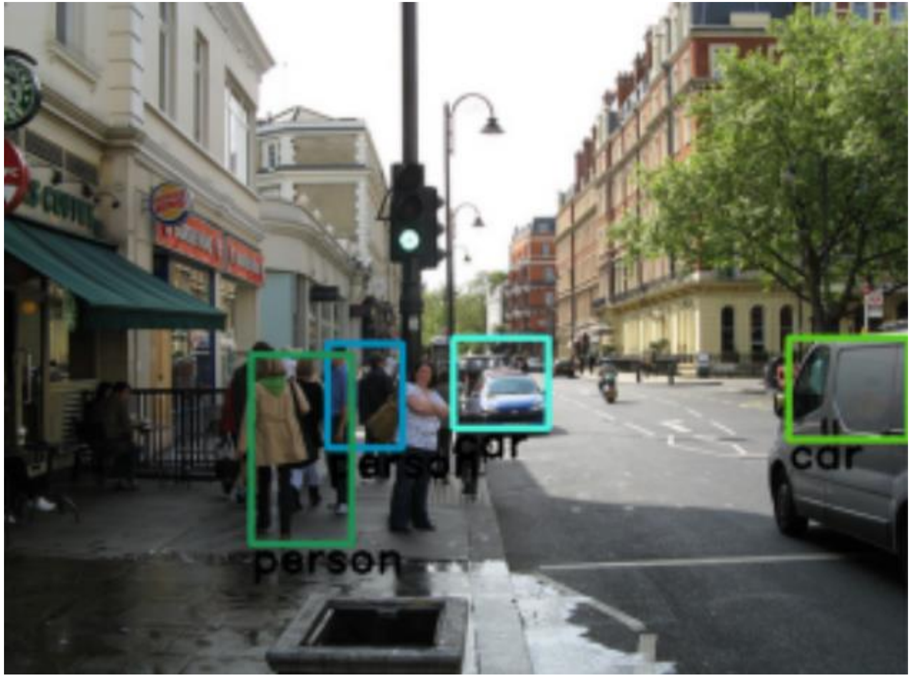
- After the cropped regions are classified, many adjacent regions will be classified as the same object and will overlap with each other. In order to choose only the best crop for each object, we run a non-maximum suppression algorithm to remove all overlapping regions.
- How NMS works:
    - Select the crop whose classification the model is most confident about
    - Remove all bounding boxes whose overlap (based on the intersection over union (IOU) metric) is above a certain threshold
    - Output all remaining bounding boxes (ones which weren’t above the overlap threshold)



Before NMS

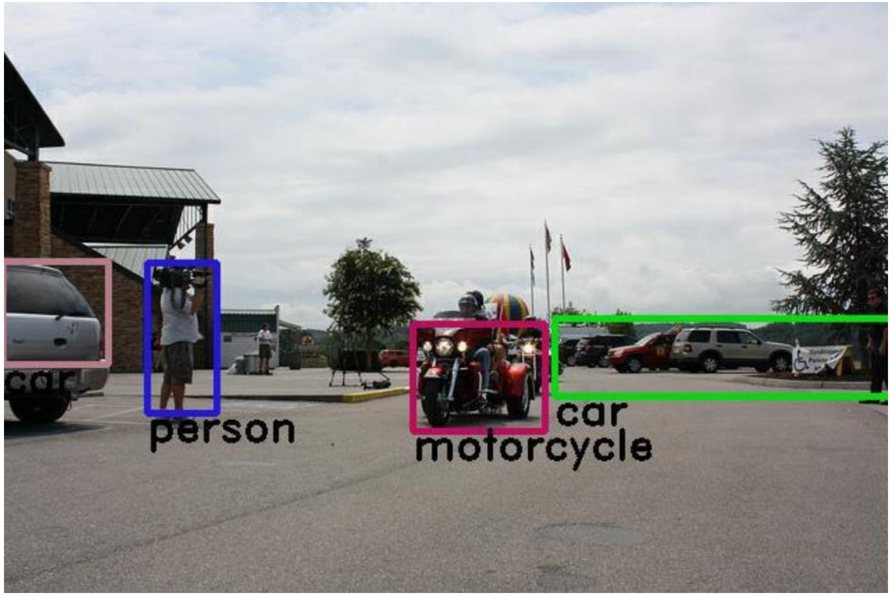


After NMS



Results

- Our model is able to classify images with relatively high accuracy
- We were successful in creating an image detection model that can detect if an object of any of our classes is in the image
- With our GitHub repo, you can download the data and train your own model



Further Exploration

- How we might be able to further improve our model:
  - Uneven training data class sizes led our model to be skewed toward the bigger classes. One way to prevent this would be to find more data to increase the size of the smaller classes
  - Another way to help this problem is to use a focal loss function. This loss function helps the model adjust its weights more evenly by more drastically adjusting weights when it mistakenly classifies an object of a smaller class than an object of a bigger class
  - Another aspect to improve is the accuracy of the bounding boxes. In order to have tighter boxes, we would be able to run a bounding box regression algorithm

Do it Yourself - How to Build and Run Our Model

Pre-requisites:

1. Python 3.7+
2. CUDA 10.0
3. cuDNN 7.5.1

Run:

1. To clone the repo - git clone https://github.com/ndcantor/tensorflow-street-classifier.git
2. Install required Python libraries - pip3 install -r requirements.txt
3. Download data, build, train, and run inference on the model - python3 street_classifier.py
4. Results :
   a. The model's train, validation, and text accuracies will be printed on the screen
   b. A confusion matrix as well as sample test images will be saved to disk.