WHITE PAPER

Artificial Intelligence
Transportation
Intel® AI Builders

Object Detection on Drone Videos using Caffe* Framework

Table of Contents

Abstract

The purpose of this article is to showcase the implementation of object detection[1] on drone videos using Intel® Optimization for Caffe*[2] on Intel® processors. The functional problem tackled is the identification of pedestrians, trees and vehicles such as cars, trucks, buses, and boats from the real-world video footage captured by commercially available drones. In this work, we have conducted multiple experiments to derive the optimal batch size, iteration count, and learning rate for the model to converge. The deep learning model developed is able to detect the trained objects in real-world scenarios with high confidence, and the ratio between the detected objects and desired objects is almost equivalent.

Introduction

Modern drones have become very powerful ever since they have been equipped with potent cameras. They have been successful in areas such as aerial photography and surveillance. The integration of smart computer vision with drones has become the need of the moment.

In today's scenario, object detection and segmentation are the classic problems in computer vision. More challenges exist with the drones due to the top-down view angles and the issue to integrate with a deep learning system for compute-intensive operations.

In this project, we implemented the detection component using Single Shot MultiBox Detector topology (SSD)[1]. We implemented our solution on Intel® Xeon® Gold processors and evaluated frame rate, and accuracy on several videos captured by the drone.

Experiment Setup

The following hardware and software environments were used to perform the experiments.

Hardware

Architecture:	x86_64
CPU op-mode(s):	32-bit, 64-bit
Byte Order:	Little Endian
CPU(s):	24
On-line CPU(s) list:	0-23
Thread(s) per core:	2
Core(s) per socket:	6
Socket(s):	2
NUMA node(s):	2
Vendor ID:	GenuineIntel
CPU family:	6
Model:	85
Model name:	Intel® Xeon® Gold Processor 6128 @ 3.40 GHz
Stepping:	4
CPU MHz:	2899.960
BogoMIPS:	6800.00
L1d cache:	32K
L1i cache:	32K
L2 cache:	1024K
L3 cache:	19712K

Software

Caffe* Setup

Caffe	1.0 (optimized on Intel® architecture)
Python*	2.7
GCC version	4.8.5

Model

The experiments detailed in the subsequent sections employ the transfer learning technique to speed up the entire process. For this purpose, we used a pre-trained Visual Geometry Group (VGG) 16 model with SSD topology.

Solution Design

The main component of our system comprised a training component and a detection algorithm running SSD. SSD is compute-intensive, but has been more optimized for Intel® architecture. We adopted Caffe* optimized on Intel architecture as our Deep Learning Frameworks and the hardware is an Intel Xeon Gold processor.

In this work, the entire solution is divided into three stages:

1. Dataset preparation
2. Network topology and model training
3. Inferencing

Dataset Preparation

The dataset used for training the model was collected through unmanned aerial vehicles (UAVs). The images collected vary in resolution, aspect, and orientation, with respect to the object of interest.

The high-level objective of preprocessing was to convert the raw, high-resolution drone images into an annotated file format, and then use for training the deep learning model.

The various processes involved in the preprocessing pipeline are as follows:

1. Data creation
2. Video to image frame conversion
3. Image annotation
4. Conversion to framework-native data format

The individual steps are detailed below.

Step 1. Dataset creation

The dataset chosen for these experiments consists of 30 real-time drone videos in the following 7 classes: boat, bus, car, person, train, tree, and truck.

Step 2. Video to image frame conversion

To train the model, all the video files were converted to image frames. The entire conversion code was built using OpenCV 3[3].

The final dataset prepared for training consists of 1,312 color images.

Step 3. Image annotation

Image annotation task involved manually labeling the objects within your training image set. In our experiment, the Python* tool, LabelImg*[4] was used for annotation. The tool provided the object coordinates in XML format as output for further processing.

The following figure shows the training split for each class:

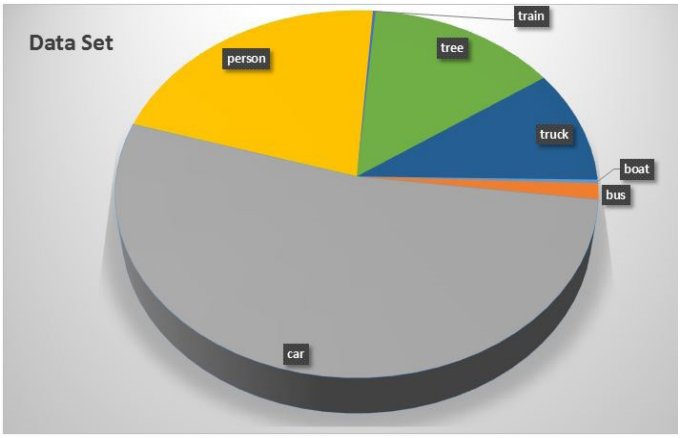

Figure 1. Training data set distribution

Step 4. Conversion to framework-native data format

To enable fast and flexible access to data during training of the network, we used framework-specific file formats.

Data Conversion

Data enters the Caffe model through data layers. Data can be ingested from efficient databases (LevelDB or LMDB), directly from the secondary memory, or from files on disk in HDF5 or common image formats. This experiment used the Lightning Memory-Mapped Database (LMDB) as the data format for ingesting the data to the network.

LMDB is a software library that provides a high-performance embedded transactional database in the form of a key-value store. It stores key/data pairs as byte arrays, which gives a dramatic write performance increase over other similar stores. LMDB always scales up the performance and maintains data integrity inherently by design.

Network Topology and Model Training

SSD

The SSD approach discretizes the output space of bounding boxes into a set of default boxes over different aspect ratios and scales per feature map location. At prediction time, the network generates scores for the presence of each object category in each default box and produces adjustments to the box to better match the object shape. Additionally, the network combines predictions from multiple feature maps with different resolutions to naturally handle objects of various sizes. SSD is simple relative to methods that require object proposals because it completely eliminates proposal generation and subsequent pixel or feature resampling stages, and encapsulates all computation in a single network. This makes SSD easy to train and straightforward to integrate into systems that require a detection component.

Model

In our experiment, we used the VGG 16 as the base network. An auxiliary structure was appended to the network to produce detections with the following key features:

- **Multiscale feature maps for detection:** We added convolutional feature layers to the end of the truncated base network. These layers decreased the size progressively and allowed predictions of detections at multiple scales.
- **Convolutional predictors:** Each added feature layer can produce a fixed set of detection predictions using a set of convolutional filters. For a feature layer of size m×n with p channels, the basic element for predicting parameters of a potential detection is a small 3×3×p kernel that produces either a score for a category, or a shape offset relative to the default box coordinates. At each of the m×n locations where the kernel is applied, it produces an output value. The bounding box offset output values are measured relative to a default box position relative to each feature map location.
- **Default boxes and aspect ratios:** We associated a set of default bounding boxes with each feature map cell for multiple feature maps at the top of the network. The default boxes tile the feature map in a convolutional manner, so that the position of each box relative to its corresponding cell is fixed. At each feature map cell, we predict the offsets relative to the default box shapes in the cell, as well as the per-class scores that indicate the presence of a class instance in each of those boxes. Specifically, for each box out of k at a given location, we computed c class scores and the four offsets relative to the original default box shape. This resulted in a total of (c+4)k filters that were applied around each location in the feature map, yielding (c+4)kmn outputs for an m×n feature map.

To achieve a faster convergence, we trained the network using a Microsoft COCO* pre-trained Caffe model. The model is available to download at the Caffe Model Zoo[5].

Transfer Learning

In our experiments, we applied transfer learning on a pre-trained VGG 16 model (trained on Microsoft COCO dataset). The transfer learning approach initializes the last fully connected layer with random weights (or zeroes), and when the system is trained for the new data, these weights are readjusted. The base concept of transfer learning is that the initial layers in the topology will have learned some of the base features such as edges and curves, and this learning can be reused for the new problem with the new data. However, the final, fully connected layers would be fine-tuned for the very specific labels that they are trained for. Hence, this needs to be retrained on the new data.

Inferencing

The video captured by the drone was broken down into frames using OpenCV with a configurable frames per second. As the frames were generated, they were passed to the detection model, which localized the different objects in the form of four coordinates (xmin, xmax, ymin, and ymax) and provided a classification score to the different possible objects. By applying the NMS (Non-Maximal Suppression) threshold and setting confidence thresholds, the number of predictions can be reduced and kept to the prediction that is the most optimal. OpenCV was used to draw a rectangular box with various colors around the detected objects (see Figure 2).

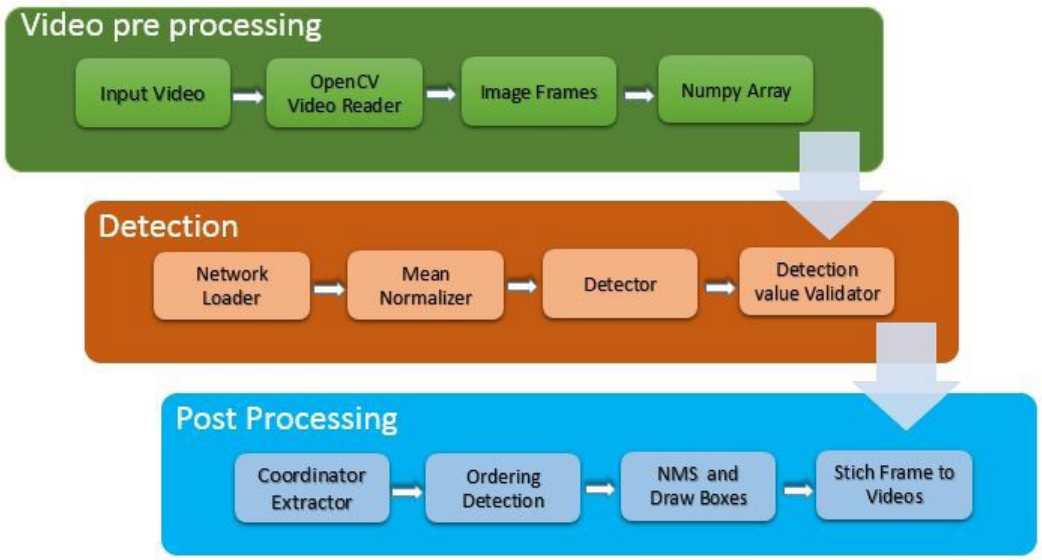

Figure 2. Detection flow diagram

Results

The following detection was obtained when the inference use-case was run on below sample images.

Figure 3. Cars in traffic as input for an inference[6]

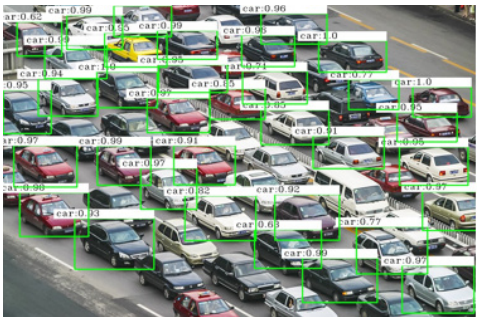

Figure 4. Green bounding boxes display the objects detected with label and confidence

Figure 5. Parked cars as input for an inference[6]

Figure 6. Green bounding boxes display the objects detected with label and confidence

Figure 7. Moving cars as input for an inference[6]

Figure 8. Green bounding boxes display the objects detected with label and confidence

Conclusion and Future Work

The functional use case attempted in this paper involved the detection of vehicles and pedestrians from a drone or aerial vehicle. The training data was more skewed towards cars as opposed to other objects of interest since it was hand crafted from videos. The use case could be further expanded for video surveillances and tracking.

References

The references and links used to create this paper are as follows:

1. Scalable High Quality Object Detection (PDF)
2. Caffe:
   - Training and deploying deep learning networks with Caffe optimized for Intel architecture
   - https://github.com/intel/Caffe
3. https://github.com/opencv/opencv
4. https://github.com/tzutalin/labelImg
5. Caffe Model Zoo:
   - models_VGGNet_coco_SSD_300x300.tar.gz
   - https://github.com/intel/Caffe/wiki/Model-Zoo
6. Test Image Sources:
   - Traffic jam Getty image
   - Picture Dealer image
   - Movement of cars and pedestrians crossing the streets of Buenos Aires