Object Detection and Its Implementation on Android Devices

Zhongjie Li
Stanford University
450 Serra Mall, Stanford, CA 94305
jay2015@stanford.edu

Rao Zhang
Stanford University
450 Serra Mall, Stanford, CA 94305
zhangrao@stanford.edu

Abstract

Object detection is a very important task for different applications including autonomous driving, face detection, video surveillance, etc. CNN based algorithm could be a great solution for object detection with high accuracy. Besides, most current deep learning applications are running on servers or desktop computers. Considering there are a lot of mobile computing devices available, we implemented the CNN based object detection algorithm on Android devices. The model architecture is based on SqueezeNet to get image feature maps and a convolutional layer to find bounding boxes for recognized objects. The total model size is around 8 MB while most other object detection model takes more than 100 MB's storage. The model architecture makes the calculation more efficient, which enables its implementation on mobile devices.

1. Introduction

Deep learning based object detection has been very successful in recent years. Especially the CNN (convolutional neural network) model has significantly improved the recognition accuracy on large data-sets. For the ImageNet benchmark data set, the CNN based model has been dominating the leader-board since it's introduced by Krizhevsky in 2012 for the first time.

While CNN based model can achieve higher accuracy, they have following disadvantages:

- **High computation cost.** The CNN based model are usually very deep with tens or hundreds of layers and each layer takes a lot of computation.
- **Large memory demand.** The CNN based model has a lot of parameters that usually take hundreds of Megabytes of memory space.
- **Low efficiency.** Most CNN based model are designed without efficiency improvement.

As mobile computing devices are very popular and comparatively powerful, people want to embrace the benefits of CNN with their mobile devices. However, to enable their mobile application, new CNN architectures need to be developed to overcome the above issues.

Also, most deep learning frameworks have provided interface for mobile platforms, including iOS and Android. In this paper, we developed a CNN based model and then implemented it with Tensorflow and Android.

Our model is trained with KITTI benchmark. The KITTI data-set has over 10 Gigabytes of well-labeled data for object detection purpose. After training, our model is able to detect objects in view of camera on the Android device.

The input to the model is a 1242-pixel width, 375-pixel height image from KITTI data-set containing labeled cars, pedestrians, cyclists as targets to be detected and other objects that we don't care. We use a SqueezeDet layer and then a ConvDet layer to generate tens of thousands of bounding box coordinates (for localization), confidence score (for detection) and class scores (for classification). All these information are sent into a Non-Maximum Suppression (NMS) filter to predict the final detection results.

Similarly, the input to our app is a camera stream, then we use inference interface to help us call the model pretrained and installed on our android device to produce the same type information (bounding box coordinates, confidence score and class scores). As above, a NMS filter implemented in the app facilities the final prediction.

The rest of this paper is organized in the following order. Section 2 lists the related work of CNN architectures as well as CNN for object detection, discusses the state-of-the-art progress in CNN model compression. In Section 3, our model based on SqueezeDet is represented and elaborated. Section 4 introduces details of the KITTI data-set and the features to be used for our model. In Section 5, we conduct experiments with our proposed model and analyze the results from the experiments. Section 6 concludes our work and our future work is stated in Section 7.

2. Related Work

In this section, we talk about CNN related work in object detection and the trend towards smaller CNN models.

2.1. CNN Architectures

Convolutional Neural Network (CNN) usually stands for the neural network which contains one or more convolutional neural layers. Each neural layer can be regarded as a combination of several spatial filters. These filters are used for extracting features from pictures. Some well-known filters are Histogram of Oriented Gradients (HOG) and color histograms, etc. A typical input for an convolutional layer is a 3-dimensional grid. They are height (H), width (W) and channels (C). Here each channel represents a filter in the convolutional layer. The input of first layer usually has a shape of (H, W, 3), where 3 stands for the RGB channels for the raw pictures.

CNN became popular in visual recognition field when it is introduced by LeCun *et al*. for handwritten zip code recognition [11] in the late 90s. In their work, they used (5, 5, C)-size filters. Later work proved that smaller filters have multiple advantages, such as less parameters and reducing the size of network activations. In a VGG network [16] proposed by Karen Simonyan *et al*., (3, 3, C)-size filters are extensively used, while the networks such as Network-in-Networ [13] and GoogLeNet [18] widely adopt (1, 1, C)-size filters, the possibly smallest filters and used for compressing volume of the networks.

With the networks go deep, the filter size design gradually become a problem that almost all the CNN practitioners have to face. Hence, several schemes for network modularization are proposed. Such modules usually include multiple convolutional layers with different filter sizes and these layers are combined together by stack or concatenation. In a GoogLeNet architecture, such as [18, 19], (1, 1, C)-size, (3, 3, C)-size and (5, 5, C)-size are usually combined together to form an "Inception" module and even with filter size of (1, 3, C) or (3, 1, C).

In addition to modularizing the network, communication and connections across multiple layers also improve the performance of the network. This seems to be a similar idea with Long Short Term Memory (LSTM) or Gated Recurrent Unit (GRU) architecture in Recurrent Neural Network (RNN). Residual Network (ResNet) [8] and Highway Network [17] adopted such ideas to allow connections to skip multiple layers. These "bypass" connections can effectively send back the gradients through multiple layers without any blocking in a backward propagation pass when necessary.

2.2. CNN for Object Detection

With the advancement of accuracy in image classification, the research for object detection also developed in a fast speed. Before 2013, feature extraction techniques such as [1], which proposed an combined application of HoG and SVM can achieve a high accuracy on the PASCAL data-set [3]. In 2013, a fundamental revolution occurred in this field, which was caused by the introduction of Region-based Convolutional Neural Networks (R-CNN), proposed by Girshick and Ross. R-CNN firstly proposes possible regions for residing objects, then makes use of CNN to classify objects in these regions. However, these two independent operations require high computation and make it time-consuming. An modification of R-CNN is made by Girshick and Ross, which is called fast R-CNN [5]. This architecture integrate the two independent tasks into one multi-task loss function, which accelerates the computation of proposals and classification. Later, a more integrated version of R-CNN, namely the faster R-CNN [15] was proposed by Ren *et al*., which achieves more than 10x faster than the original R-CNN. A recent proposal, R-FCN [12] with a fully convolutional layer as the final parameterized layer further shortens the computation time used for region proposals.

R-CNN can be regarded as a cornerstone for the development of CNN for object detection. A large amount of work is based on this architecture and achieves great accuracy. However, a recent work shows that CNN based object detection can be even faster. YOLO (You Only Look Once) [14] is such an architecture integrating region proposition and object classification into one single stage, which significantly contributes to simplification of the pipeline of object detection, as well as reduction of the total computation time.

2.3. Toward Smaller Models

With CNN goes deeper, more parameters need to be stored, which makes the model larger and larger. Deeper CNN and larger modules usually achieve a higher accuracy, but people wonder whether a small model can reach a similar accuracy as a large model. In this sub-section, we talk about several popular model compression techniques aiming to reduce the size of CNN models.

As we know, singular value decomposition (SVD) is widely used to reduce matrix dimensionality. It is also introduced to pre-trained CNN models [2] to reduce model size. Another approach reported is Network Pruning [6], proposed by Han *et al*., which prunes the parameters below a certain threshold to construct a sparse CNN. Recently, Han *et al*. have further improved their approach and proposed a new approach, Deep Compression, together with their hardware design to accelerate the computation of CNN models. A recent research called SqueezeNet [9] even reveals that a complex CNN model as AlexNet [10] accuracy can be compressed to smaller than 0.5 Mbytes.

Here are two examples of model compression. The fa-

mous ImageNet winner VGG-19 model stores more than 500 Mbytes parameters, which achieves a top-5 accuracy of about 87% on ImageNet, while the equally famous ImageNet winner GoogLeNet-v1 only contains about 50 Mbytes parameters, achieving the same accuracy as VGG-19. The well-known AlexNet [10] model with a size of more than 200 Mbytes parameters, achieves about 80% top-5 accuracy on ImageNet image classification challenge, while the SqueezeNet [9] model with a much smaller size, about 4.8 Mbytes parameters, can also achieve that accuracy. We can anticipate that there is much room left for compressing these CNN models, to better fit them to portable devices.

3. Methods

In this section, the CNN model to detect objects and the implementation of android app are elaborated in detail.

3.1. CNN Model

The model has the benefit of small model size, good energy efficiency and good accuracy due to the fact that it's fully convolutional and only contains a single forward pass. The overview of this object detection model is as following in Figure 1.

The CNN model we adopted is called SqueezeDet [21]. The SqueezeDet model is a fully convolutional neural network for object detection. It's based on SqueezeNet architecture that extracts feature maps from image with CNN. Then another convolutional layer is used to find bounding box coordinates, confidence score and class probabilities [20]. Finally, a multi-target loss is applied to compute final loss in training phase and a NMS filter is enforced to reduce the number overlapping bounding boxes and generate final detection in evaluation phase.

3.1.1 SqueezeNet and ConvDet

The core of the SqueezeNet model is "fire" module. It contains two part. First, it squeeze the current state with 1x1 convolutional layer. And Later it expand the results with 1x1 and 3x3 convolutional layers. The main purpose of the "fire" module is to use 1x1 convolutional layer to replace 3x3 convolutional layers as much as we can, as the 3x3 convolutional layer take 9 times more parameters. And if 3x3 has to be used for the sake of activation area, we want to limit the input layer size as much as we can. With 9 layers' fire modules, 2 layers of polling and 1 layer of dropout, the feature map for each image can be obtained. After that, a 1x1 convolutional layer is used to extract bounding box coordinates, class scores and confidence score. For each activation in feature map, it will generate K bounding boxes with 4K bounding box coordinates (x_1, y_1, x_2, y_2). Each

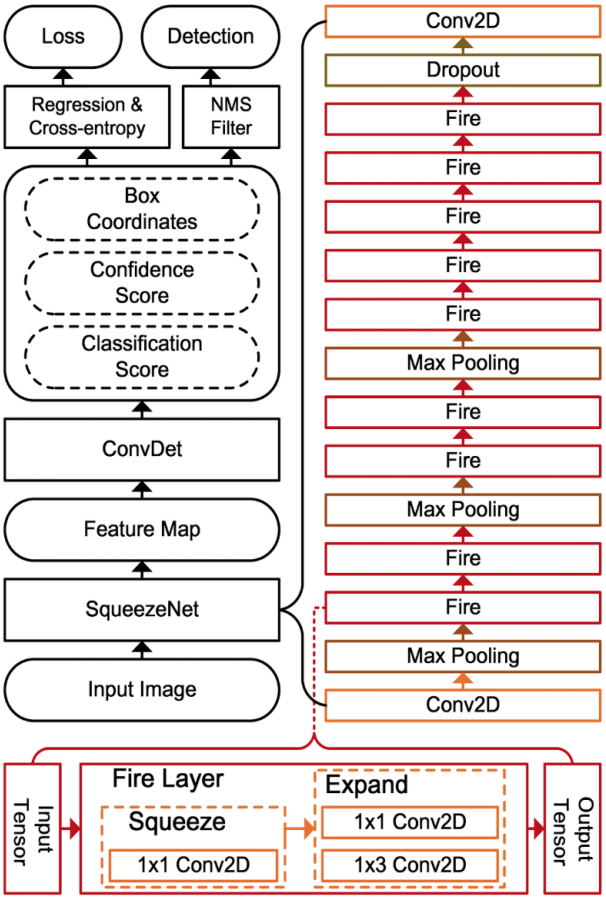

Figure 1. SqueezeDet Architecture

bounding box corresponds to 3 class scores and 1 confidence score. These information will be used for loss calculation in training and for final detection in inference.

3.1.2 Multi-Target Loss and NMS Filter

In training phase, the loss as proposed in [21] is calculated as a weighted sum of localization loss L_{loc}, detection loss L_{det} and classification loss L_{cls} as shown in Equation 1. All the 3 losses are normalized by their number of terms.

$$L = \lambda_1 L_{loc} + \lambda_2 L_{det} + \lambda_3 L_{cls} \quad (1)$$

The localization loss is defined as regression loss and calculated using the squared difference of bounding box coordinates. In Equation 2, I_{ijk} is the indicator. It equals 1 if the ground truth bounding box is assigned to the predicted one which has the highest Intersection Over Union (IOU) with the ground truth, otherwise it equals 0. In this way, only the "responsible" predicted bounding box will contribute to the final loss. In the equation, N is the number of ground truth

objects in that image. W is the width of the feature map. H is the height of the feature map and K is the factor that 1 activation in the feature map corresponds to K predicted bounding box.

$$L_{loc} = \frac{1}{N}\sum_{i=1}^{W}\sum_{j=1}^{H}\sum_{k=1}^{K} I_{ijk}[(x^1_{ijk} - x^{G1}_{ijk})^2 + (x^2_{ijk} - x^{G2}_{ijk})^2 + (y^1_{ijk} - y^{G1}_{ijk})^2 + (y^2_{ijk} - y^{G2}_{ijk})^2] \quad (2)$$

The detection loss is defined as regression loss and calculated using the squared difference of confidence score. γ_{ijk} is the predicted confidence score. γ^G_{ijk} is the ground truth confidence score, computed as the IOU of "responsible" bounding box. For the bounding boxes which are not "responsible" for ground true, they are penalized by ${(1 - I_{ijk})\gamma_{ijk}}^2$ in Equation 3. Since the confidence score is ranged from 0 to 1, it should be suppressed by a sigmoid function before the calculation of detection loss.

$$L_{det} = \sum_{i=1}^{W}\sum_{j=1}^{H}\sum_{k=1}^{K} \frac{\lambda_{21}}{N} I_{ijk}(\gamma_{ijk} - \gamma^G_{ijk})^2 + \frac{\lambda_{22}}{WHK - N}(1 - I_{ijk}){\gamma_{ijk}}^2 \quad (3)$$

The classification loss is defined as cross-entropy loss. p_c is the predicted probability for class c and it is obtained after a softmax function, which normalize all C classes scores to probabilities, ranged in [0, 1] and summed up to 1. l^G_c is the label indicating the ground truth class. It equals 1 if p_c is the ground true, otherwise it equals 0 in Equation 4

$$L_{cls} = \frac{1}{N}\sum_{i=1}^{W}\sum_{j=1}^{H}\sum_{k=1}^{H}\sum_{c=1}^{C} I_{ijk} l^G_c \log(p_c) \quad (4)$$

In inference phase, a Non-Maximum Suppression (NMS) filter is used to reduce the overlapping bounding boxes and generate the final detections. To simply the explanation , consider only 1 image which generates M bounding boxes. Each bounding box b_m corresponds to 4 coordinates $x^1_m, x^2_m, y^1_m, y^2_m$, 3 classification scores p^1_m (car), p^2_m (pedestrian), p^3_m (cyclist) and 1 confidence score γ_m. To implement NMS algorithm, the IOU threshold is defined as T.

The aim of NMS algorithm is to reduce redundant bounding boxes by selecting the most probable bounding box with the highest confidence score each time and then removing predicted bounding boxes with a high IOU (which implies high overlapping) over the threshold. This algorithm works for each class and is elaborated in Algorithm 1.

Algorithm 1 Non-Maximum Suppression Algorithm

Require: $x^1_m, x^2_m, y^1_m, y^2_m, p^1_m, p^2_m, p^3_m, \gamma_m, 1 \le m \le M$
Ensure: Index set S

1: Initialize $S \leftarrow \emptyset, S_c \leftarrow \emptyset, 1 \le c \le 3$.
2: For each m, assign m to the class set S_c with the highest classification score among p^1_m, p^2_m, p^3_m.
3: In each class S_c, $s_u \leftarrow \arg\max_m \gamma_m$. $S = S \cup \{s_u\}$.
4: In each class S_c, calculate IOU ϕ_v between s_l and $s_v, v \in \{1, 2, ..., M\}$ according to $x^1_v, x^2_v, y^1_v, y^2_v$. For each v satisfying $\phi_v > T$ in class c, $S_c = S_c - \{v\}$.
5: Repeat 2 to 4, until every S_c becomes an empty set.
6: Return S.

3.2. Android Implementation

For the implementation of CNN model in Android device, we used the interface provided by "Tensorflow Android Camera Demo"[7]. First, the CNN model parameters need to be trained and saved into a protobuffer file. Basically, the way to save the CNN graph is to freeze all variables into constants with well trained values and save them by their names. Then with Android interface tool (called "InferenceInterface"), Android app can load tensor with values, run the graph and read tensor output values. However, current interface only support loading values and reading outputs in the format of 1-D array. So, the input node/output node in the graph should be designed to be 1D array to accommodate that. The app is designed with a streaming video from the camera, and each image frame is passed to the CNN model for object detection. And then the detected results are marked with boxes in real time. To accommodate the 8 fps of the default frame rate in Android device, we need the total processing time to be less than 125 ms. The overall app architecture is as shown in Figure 2.

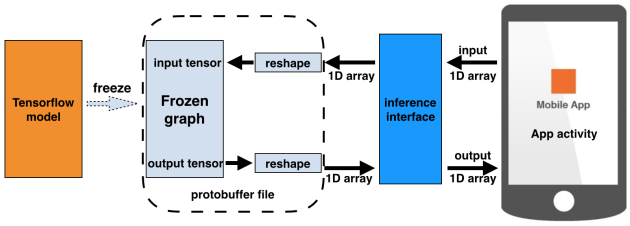

Figure 2. Android App Architecture

4. Data-set and Features

The data-set we use is The KITTI Vision Benchmark Suite [4], which is made for academic use in the area of autonomous driving. For our target, we use the object detection data-set, which contains 7481 training images and 7518 test images. Total 80256 objects are labeled for this data-set and the 3 classes used for evaluation are cars, pedestrians and cyclists. The distribution of object number in the training data-set is shown in Figure 3. 51865 objects are labeled,

including 28742 cars, 4487 pedestrians and 1627 cyclists. On average: 3.8 cars, 0.6 pedestrian and 0.2 cyclist per image. The pictures in this data-set are fully color PNG files. It's clear that cars are more frequently shown in image than pedestrians and cyclists, so the biased data may have some impact on the accuracy on different classes.

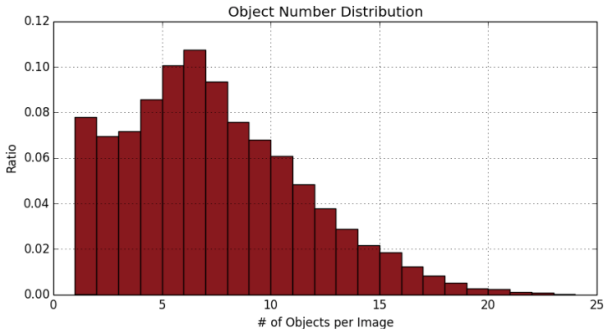

Figure 3. Object Quantity Distribution in KITTI Training Set

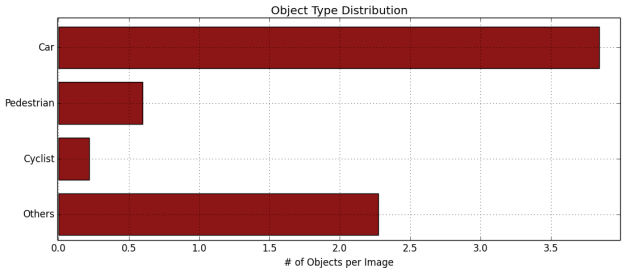

Figure 4. Object Type Distribution in KITTI Training Set

Data augmentation is implemented in the model training including image flipping, random cropping, batch normalization. Figure 5 is a typical scenario image in the dataset.

Figure 5. Example of an Image in Data-set

5. Experiments

The squeezeDet model is trained with KITTI detection dataset. The model is trained in Google Cloud Engine with 8v CPU, 30GB RAM and 1 GPU (NVIDIA TESLA K80). The batch size is 20. It takes around 1.2 s for each batch. After 35k steps of training, the overall recall can get 81%. The detection precisions are as Table 1.

Detection accuracy	car	cyclist	pedestrian
easy	90%	86%	80%
medium	85%	80%	74%
hard	75%	77%	67%

Table 1. Detection precision on KITTI object detection dataset

Stochastic Gradient Descent with momentum is used as the optimizer for model training and the learning rate decay is implemented to help the training process converge Figure 6, Figure 7, Figure 8 and Figure 9 .

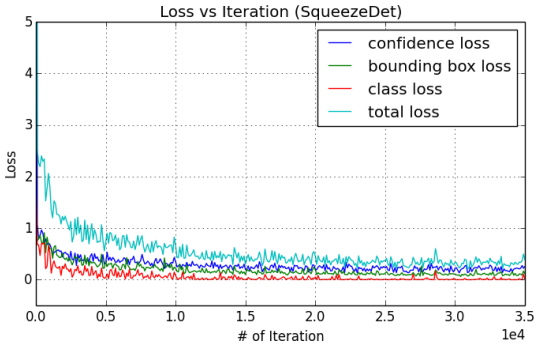

Figure 6. Multi-Target Train Loss

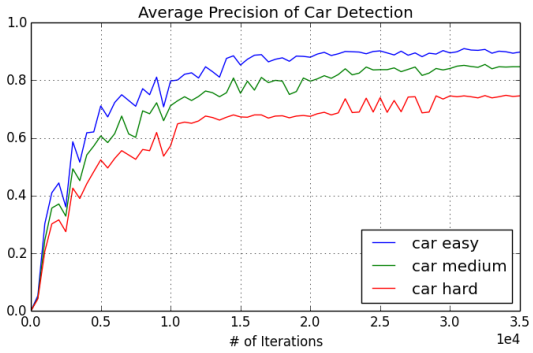

Figure 7. Car Training Accuracy

To better understand why the model failed to recognize some of the images, we went through the samples with wrong classification or missed detections and found that there are three major failure modes, including wrong labels, partially blocked object and confusion between pedestrians and cyclists. It's understandable that there may be some human error during labeling work, so the model shouldn't be expected to reach 100% accuracy. Figure 10 showed some examples of wrong labels. Another difficult task for the

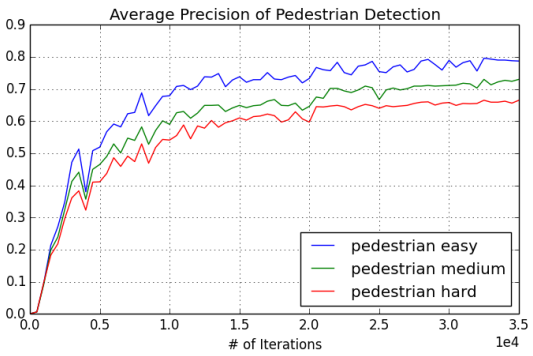

Figure 8. Pedestrian Training Accuracy

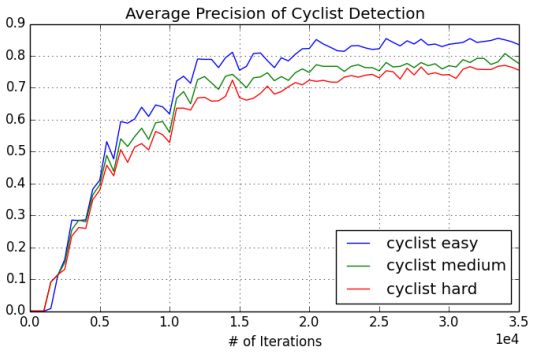

Figure 9. Cyclist Training Accuracy

models is to recognize partially blocked objects, especially for objects with most of their surfaces blocked, as shown in Figure 11. Between the class of pedestrian and cyclist, there are a lot of confusion due to their natural similarity. At some angle where the bicycle is hard to identify, the cyclist is easily to be recognized as pedestrian Figure 12. It may be worth discussing that whether the cyclist class and pedestrian class can be combined for autonomous driving's application.

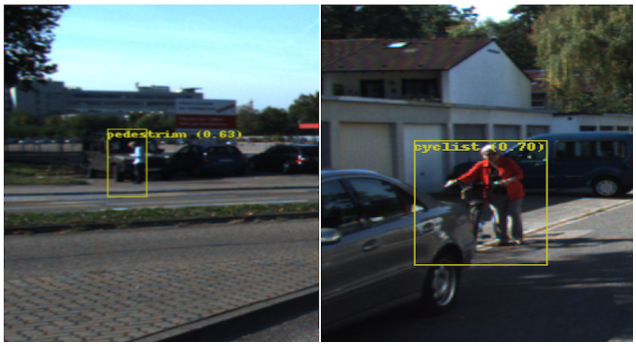

Figure 10. Failure Mode: Wrong Label

The image quality may vary in the real life scenarios, for

Figure 11. Failure Mode: Partially Blocked Object

Figure 12. Failure Mode: Confusion Between Pedestrians and Cyclists

example, the image gets darker in a cloudy day and gets blurry in a rainy or foggy day, etc. Considering that the model is trained with preset conditions, we would like to evaluate how accurate the model is under different image variation types. So, we processed the image with varying conditions including brightness, blurriness, contrast, color degradation and image resolutions. Then we run the object detection model on these processed images and found that the accuracies do degrade quite a lot under conditions like blur and low contrast, as shown in Figure 13.

Then we plot out the model's performances under different image conditions to understand the accuracy's sensitivity to different variations, as shown in Figure 14. It shows that the model's accuracy is very sensitive to image blurs. The average accuracy drops 48% with blurred image, while it drops less than 10% for variations like brightness and color variations.

6. Conclusion

In this project, we trained a CNN object detection model at desktop platform and applied the trained model into a mobile platform. As a baseline, we have a running Android app that runs our CNN model trained by Tensorflow offline. The model size is 8 MegaBytes and the achieved testing

Figure 13. Object Detection Example with Image Variations

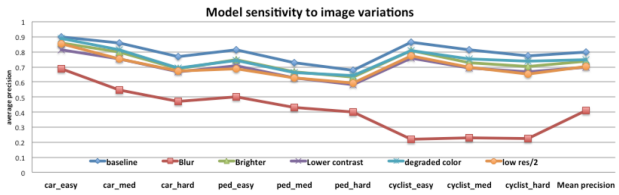

Figure 14. Object Detection Accuracy Sensitivity

accuracy is 76.7%.

The interface between Tensorflow and Android is still not perfect as the latency caused by interface is longer than the actual computation time in the graph. In addition, there is no documentation for the interface. Google announced that they plan to release the "Tensorflow Lite" for mobile platform, so we expect these issues to be significantly improved.

7. Future Work

The Android application can be further improved on its stability and functionality. Also, this app is based on old and less efficient interface, which is called "InferenceInterface". The detection latency and stability can be improved by switching the interface to "Tensorflow Lite", which is yet to be released soon.

As we see in the experiments section, the robustness of the model need to be improved to get good accuracy with image variations of brightness/contrast/blur, etc. So, another thing that can help is to manually add image variations to input image set such that the model is less sensitive to the image variations.

Iandola [9] proposed the idea of model compression with sparsity and 6-bit quantization to reduce the squeezeNet model size from 5MB to 0.47MB with equivalent accuracy. This deep compression method are not explored in this project due to time constraint, but it worth looking into in future development. Smaller model is not only beneficial for storage capacity, it should also be beneficial for computing efficiency.

References

[1] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[2] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*, pages 1269–1277, 2014.

[3] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[4] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3354–3361. IEEE, 2012.

[5] R. Girshick. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1440–1448, 2015.

[6] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015.

[7] A. Harp. Tensorflow android camera demo. `https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/android`, 2016.

[8] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[9] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[11] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[12] Y. Li, K. He, J. Sun, et al. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in Neural Information Processing Systems*, pages 379–387, 2016.

[13] M. Lin, Q. Chen, and S. Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[14] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788, 2016.

[15] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[16] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[17] R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[18] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[19] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[20] B. Wu. Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. `https://github.com/BichenWuUCB/squeezeDet`, 2016.

[21] B. Wu, F. Iandola, P. H. Jin, and K. Keutzer. Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. *arXiv preprint arXiv:1612.01051*, 2016.