LCDet: Low-Complexity Fully-Convolutional Neural Networks for Object Detection in Embedded Systems

Subarna Tripathi
UC San Diego [*]
stripathi@ucsd.edu

Gokce Dane
Qualcomm Inc.
gokced@qti.qualcomm.com

Byeongkeun Kang
UC San Diego
bkkang@ucsd.edu

Vasudev Bhaskaran
Qualcomm Inc.
vasudevb@qti.qualcomm.com

Truong Nguyen
UC San Diego
tqn001@eng.ucsd.edu

Abstract

Deep Convolutional Neural Networks (CNN) are the state-of-the-art performers for the object detection task. It is well known that object detection requires more computation and memory than image classification. In this work, we propose LCDet*, a fully-convolutional neural network for generic object detection that aims to work in embedded systems. We design and develop an end-to-end TensorFlow(TF)-based model. The detection works by a single forward pass through the network. Additionally, we employ 8-bit quantization on the learned weights. As a use case, we choose face detection and train the proposed model on images containing a varying number of faces of different sizes. We evaluate the face detection performance on publicly available dataset FDDB and Widerface. Our experimental results show that the proposed method achieves comparative accuracy comparing with state-of-the-art CNN-based face detection methods while reducing the model size by* $3\times$ *and memory-BW by* $3-4\times$ *comparing with one of the best real-time CNN-based object detector* YOLO *[23]. Our* 8*-bit fixed-point TF-model provides additional* $4\times$ *memory reduction while keeping the accuracy nearly as good as the floating point model and achieves* $20\times$ *performance gain compared to the floating point model. Thus the proposed model is amenable for embedded implementations and is generic to be extended to any number of categories of objects.*

1. Introduction

Deep Convolutional Neural Network (CNN) based models are the current state-of-the-art for the task of object detection. The best methods for object detection aim to increase the accuracy on standard datasets. They run on powerful GPUs that dissipate a huge amount of power. On the other hand, embedded processors and DSPs are great low-power solutions where the instruction sets benefit from fixed-point operations. For practical deployment of object detector on mobile devices, we need low-complexity CNN models that can run on embedded processors. The algorithms need to leverage the fixed point operations without compromising the accuracy.

In this paper, we propose *LCDet*, a low-complexity object detector to address the above issues. We design and develop an end-to-end TensorFlow-based fully-convolutional deep neural network for object detection inspired by YOLO [23]. The differences of the proposed network from YOLO are described in section 3.1.

We choose face detection as a use-case due to its many practical applications in mobile phones, although the algorithm is generic enough for any number of classes. The detection pipeline of our TensorFlow-Slim based network requires a single forward pass through the network. Evaluation results for the face detection performance on publicly available datasets such as FDDB [12] and Widerface [40] show that the proposed method achieves comparative accuracy with respect to state-of-the-art CNN-based face detection methods while reducing the model size by $3\times$ and memory-BW by $3-4\times$ comparing with YOLO [23], one of the fastest DCN-based object detector. Additionally, we quantize the model by $8-$bit precision, which leads to additional $4\times$ memory reduction with almost no loss in detection accuracy. The $8-$bit quantization is one of the most important steps for a deployment in fixed-point architectures such as DSPs or dedicated convolution accelerators.

We believe we report one of the first studies of $8-$bit quantization on TensorFlow models for object detection task that heavily uses *regression*. It is understood that 8-bit quantization of floating point models that were trained

[*] Work done in part during an internship at Qualcomm.

for regression task are more prone to accuracy drop comparing with the models that were trained for classification task. Experimental results show that the highest detection accuracy with quantized model drops by less than $1-2\%$ comparing with the floating point model and achieves $20\times$ performance gain in terms of frame rate compared to the floating-point model.

The rest of the paper is organized as follows. We discuss the related work in section 2. We present the method, including the architecture, training, and model quantization in section 3. In section 4, we report our results on face detection task and discuss relative complexity and accuracy. Finally, we conclude in section 5.

2. Related Work

2.1. CNN-based Object Detection

Several papers propose ways of using deep convolutional networks for detecting objects [7, 8, 26, 30, 30, 27, 38, 6, 1]. Some approaches classify the proposal regions [7, 8] into object categories and some other recent methods [26, 23, 16] unify the localization and classification stages. Detailed prior-art on object detectors and their speed-accuracy tradeoff can be found in [10].

Precisely, the single stage detection pipeline of YOLO [23] is extremely fast. YOLO is the first reported real-time CNN-based object detector model that runs with high-end GPUs. Its performance accuracy on PASCAL VOC [5] dataset is comparable with state-of-the-art methods.

Unlike YOLO, our model is fully-convolutional. Thus it is highly memory-efficient, computationally more effective and not restricted by input image resolution.

2.2. CNN Object Detection for Embedded Systems

The most accurate and best performing CNN-based models require high-end GPUs. There is a growing interest for developing specific hardware design [18, 3] including FPGAs, DSPs, custom vision chips and embedded GPUs for energy efficient CNN-based object-detection. Detailed algorithmic advancements and case-studies for CNN-algorithms for embedded systems can be found in [21, 17].

The best-performing CNN-based object detection methods which run on real-time (on high-end GPUs), falls significantly short on embedded GPUs. For example, a supplier in surveillance camera market found out that even after replacing the back-end of YOLO from GoogleNet to a simpler CNN such as AlexNet, it's embedded implementation runs at most 5 frames per second on embedded GPUs. This motivates us to investigate on fully-convolutional low complexity object detector that can run real-time on embedded platforms.

TensorFlow is an open source framework and used by many developers to create their own AI-applications. Recently, [28] announced that Snapdragon 835 [22] includes TensorFlow-optimized Hexagon 682 DSP. This DSP architecture and others in this family are designed to process certain features more quickly and at lower power than a CPU or GPU. Our proposed TensorFlow-based model exploits the advantages of similar architecture and is useful for real-time object detection tasks on these platforms.

2.3. CNN-based Face Detection

We choose face detection as an application and evaluate *LCDet*, the proposed object detector, for this task. As per the FDDB evaluation server [12], the state-of-the-art face detection methods are based on convolutional neural networks [39, 15, 41, 34, 13, 29, 20]. Yang *et al*. presented a neural network which combines feature responses regarding facial parts [39]. Li *et al*. presented an integrated method of neural networks and 3D face model [15]. Yu *et al*. modified VGG-16 networks, and also proposed intersection over union (IoU) loss layer [41]. Recently presented works are based on faster R-CNN [26, 34, 13, 29]. Our model achieves comparable quality with Faster-RCNN based methods. Additionally, our method meets all the requirements for real-time embedded applications while the above other CNN-based face detection methods can not achieve real-time performance in embedded platforms.

3. Methods

3.1. Network Architecture

Our proposed model is inspired by YOLO [23] which adopts a single-pass detection pipeline combining bounding box localization and classification by a single network. Layers connectivity differences between YOLO and LCDet is outlined in Figure 1. The last two fully-connected layers of YOLO are replaced by fully-convolutional layers in the proposed model.

Other differences are described below. Unlike the LeakyReLU non-linearity in YOLO, we apply ReLU activations in all but last layer. Additionally, for the final layer of output, we apply different activations on classification (softmax) output, confidence (sigmoid) score, and localization (no activation) outputs. YOLO doesn't apply any nonlinearity in the final layer. From the layers connectivity perspective, the back-end of the CNN architecture is almost similar to YOLO; however, *LCDet* can work on any input image resolution by virtue of being fully-convolutional.

Let's suppose, the convolutional layer right before the first fully-connected layer in YOLO is called the final feature map of spatial size $W_f \times H_f$. Here, W_f and H_f denote the number of grid centers along the horizontal and vertical axes.

From the same feature layer, the proposed model connects to the final convolutional layer that outputs in a spa-

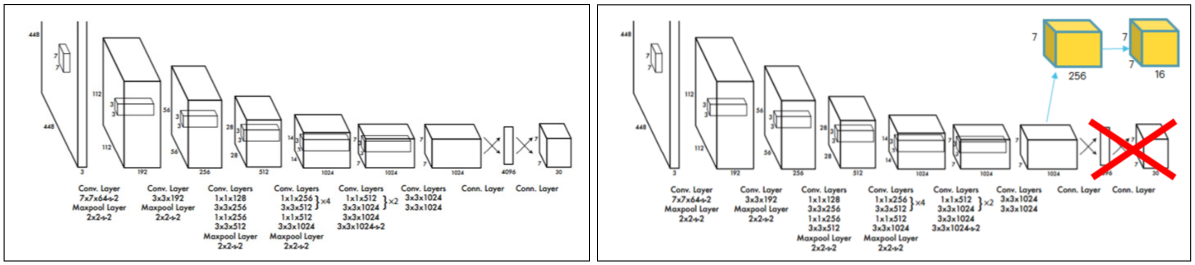

Figure 1. YOLO vs proposed network. Output channel's value 16 corresponds to 1 class-conditional and $1 + 4$ confidence and coordinates for each of the $B = 3$ boxes. All Leaky ReLU activations are replaced by ReLU in the final model of LCDet.

tial grid-like pattern ($W_f \times H_f \times Channels$) as shown in Figure 1. Each grid center is associated with C class probabilities, 1 confidence score, and 4 scalar values of coordinates for each of the B ($= 3$) possible bounding boxes. Similar to YOLO, the confidence score is the predictor for Intersection-over-Union with the ground truth bounding box. Finally, we employ Non-Maximum suppression (NMS) for keeping top bounding boxes. During the inference, the detection pipeline consists of a single forward pass through the network.

3.2. Training Methodology

Unlike Faster R-CNN [26], which deploys a 4-step alternating training strategy to train Region Proposal Network (RPN) and detector network, our detection network can be trained end-to-end, similarly to YOLO [23]. We apply a multi-part object detection loss as described in (equation 1) similar to YOLO.

$$
\begin{aligned}
\text{loss} = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{K} \mathbb{1}_{ij}^{obj} \left(x_i - \hat{x}_i\right)^2 + \left(y_i - \hat{y}_i\right)^2 + \\
\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{K} \mathbb{1}_{ij}^{obj} \left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2 + \\
\sum_{i=0}^{S^2} \sum_{j=0}^{K} \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 + \qquad (1) \\
\lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{K} \mathbb{1}_{ij}^{noobj} \left(C_i - \hat{C}_i\right)^2 + \\
\sum_{i=0}^{S^2} \mathbb{1}_{i}^{obj} \sum_{c \in classes} \left(p_i(c) - \hat{p}_i(c)\right)^2
\end{aligned}
$$

where $\mathbb{1}_i^{obj}$ denotes if the object appears in cell i and $\mathbb{1}_{ij}^{obj}$ denotes that jth bounding box predictor in cell i is *responsible* for that prediction. The loss function penalizes classification and localization error differently based on presence or absence of an object in that grid cell. x_i, y_i, w_i, h_i corresponds to the ground truth bounding box center coordinates, width and height for objects in grid cell (if it exists) and $\hat{x_i}, \hat{y_i}, \hat{w_i}, \hat{h_i}$ stand for the corresponding predictions. C_i and $\hat{C_i}$ denote confidence score of *objectness* at grid cell i for ground truth and prediction. $p_i(c)$ and $\hat{p_i}(c)$ stand for conditional probability for object class c at cell index i for ground truth and prediction respectively. We use similar settings for YOLO's object detection loss minimization and use values of $\lambda_{coord} = 5$ and $\lambda_{noobj} = 1.$.

We additionally apply sigmoid activation on the prediction of confidence score. Confidence score should be in [0,1] as it is ideally the IOU predictor. We employ the softmax on class prediction. However, for the special case of single-class detection in YOLO-style, we employ sigmoid activation on 1 class prediction output from the network.

The proposed model uses 448 $\times$ 448 frames as input while training and regresses on category types and locations of possible objects at each one of $S \times S$ non-overlapping grid cells. (The model is capable of using any resolution image as an input) For each grid cell, the model outputs class conditional probabilities as well as K bounding boxes and their associated confidence scores. As in YOLO, we consider a *responsible* bounding box for a grid cell to be the one among the K boxes for which the predicted area and the ground truth area shares the maximum Intersection Over Union. During training, we simultaneously optimize classification and localization error (equation 1). For each grid cell, we minimize the localization error for the *responsible* bounding box with respect to the ground truth only when an object appears in that cell.

3.3. Detection-Specific Layers

From the feature layer of size $W_f \times H_f \times Ch_f$, YOLO [23] employs two fully-connected layers. For simplicity, we denote these two layers together as YLDet. We denote the last two convolutional layers of the proposed model by ConvDet.

YOLO works with input feature map size of $7 \times 7 \times 1024$. $F_{fc1} = 4096$, $C = 20$, $W_0 = H_0 = 7$. Thus the number of parameters in *YLDet* is about 269×10^6. The first convolutional layer in the ConvDet has $Ch_{d1} = 256$ parameters. For same feature map size and number of output grid centers, ConvDet only requires 2.3×10^6 parameters, which is

	RP	**cls**	**# Parameters**
RPN	✓	✗	$Ch_f K(5+C)$
YLDet	✓	✓	$F_{fc1}(W_f H_f Ch_f + W_o H_o(C+5K))$
ConvDet	✓	✓	$F_w F_h Ch_{d1}(Ch_f + (C+5K))$

Table 1. Comparison between RPN, ConvDet and YLDet. RP stands for Region Proposals, *cls* denotes classification.

115× less than YOLO.

3.4. Quantized Model

Often times, DSPs or dedicated convolution accelerators operate on fixed point instruction set. There exists literature on fixed point models for embedded systems [37, 9] for classification task. It is well-known that $8-$bit models [32] perform as good as the floating point model for classification [33]. The justification for the high accuracy in low-precision modes comes from the fact that the final activation is a probability *i.e.* in [0,1] intervals, can be represented with an unsigned number without any concern on scaling. However, we are not aware of many published reports on the study of quantization for object detection task that regresses coordinates of all objects.

For each layer of LCDet, we convert the 32-bit floating point parameters to 8-bit fixed point parameters via [35, 31]. The entire object detection model can work in 8-bit fixed point implementation without going back and forth from floating and fixed point after the accumulation in each convolutional layer. Although literature exists for training with low precision mode [4], we perform quantization only for the inference. Since training is performed off-line, it is reasonable and more practical to quantize the trained model for inference.

To quantize the 32-bit floating point to the 8-bit fixed point, we first store minimum and maximum value at a layer. Then, we quantize the relative value to the linearly distributed closest integer in [0,255].

$$w_q = \left[255 \frac{w_f - w_{min}}{w_{max} - w_{min}}\right] \quad (2)$$

where w_q, w_f, w_{min}, and w_{max} represents quantized variable, variable in floating point, minimum, and maximum. $[\cdot]$ represents rounding to the closest integer.

Although we know of no fundamental mathematical reason as to why the low precision mode works well without low-precision training, we see from our experimental results that regression models (such as our LCDet) also works well for low-precision inference. Quantized LCDet is as good as the floating-point model in terms of detection accuracy for the face detection application in general.

4. Experimental Results

In this section, we present the performance of the proposed algorithm with floating point model as well as the associated 8-bit quantization on commonly used face databases such as FDDB benchmark [12] and Widerface [40] validation split. We also provide an analysis of the proposed method's performance in terms of speed, complexity and memory requirements. Although, face detection is studied as a use case, the network does not use any face-specific processing such as facial parts or attributes. LCDet is a general purpose object detector.

We use transfer learning by first converting the weights for detecting 20-category PASCAL objects [5] from DarkNet [25] library to a TensorFlow checkpoint, called *YOLO PASCAL TF*. For face detection with baseline YOLO [23], we first restore parameters from the *YOLO PASCAL TF* checkpoint for all except the last layer. We then finetune it for the 1-class (only face) object detection task. For LCDet, we restore parameters for all but last two layers from the *YOLO PASCAL TF*. We then initialize the last two layers with random weights and train the proposed LCDet for single-class object detection. Also, we replace all LeakyReLU activations from YOLO architecture to ReLU activation after each convolution except the final output layer. Readers are referred to section 3.2 for the activations in the last layer.

We adopt data augmentation techniques such as scaling and object-centric cropping to minimize overfitting. In our experiments, we use initial learning rate of 10^{-5}, the minibatch size of 32, and $8K$ epochs. We use Adam [14] optimizer for training. We used NVIDIA Tesla K40 GPUs for training and testing experiments. We also trained similar models with batch normalization for the convolutional layers. The training appeared to converge early, but that didn't yield any improvement in the detection accuracy. In order to minimize the number of parameters, we go with models without batch normalization.

4.1. Dataset

FDDB [12] is a benchmark dataset for face detection in unconstrained settings. It contains $2,845$ images with a total of $5,171$ faces. The dataset provides fixed partitioning of 10 folds. WIDER FACE is a larger dataset [40] for face detection. It consists of $32,203$ images with $393,703$ faces. The dataset is organized in 61 event classes. For each event class, 40%, 10%, and 50% of data were selected for training, validation, and testing. The ground truth for test split is not disclosed. In our experiments, we scaled each image to a pre-determined size. For gray-scale images, we duplicate the single channel three times to make them images with three channels.

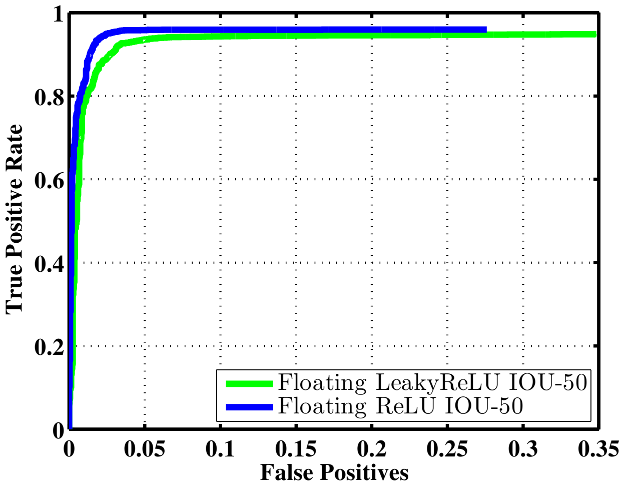

Figure 2. Performance of LCDet on FDDB for LeakyReLU vs ReLU model with discrete core metric.

4.2. Detection Accuracy

We first evaluate LCDet for two different nonlinearities such as Leaky ReLU and ReLU at each convolutional layers. Although the model has been initialized from YOLO PASCAL TF which was trained with Leaky ReLU activations. After finetuning for face detection task with two different models with two different activations, we find ReLU activation performs better than leaky ReLU. The performance of LCDet with LeakyReLU vs ReLU has been shown in Figure 2. Next we evaluate the performances of LCDet and its 8-bit quantized model on FDDB dataset. The models are trained on FDDB images. This allows us to investigate the performance gap of floating and fixed point models independent of the other components in the whole system.

Figure 3 shows the performance of the floating point model and the 8-bit quantized model by the TP-FP curve with discrete score evaluation method per [12]. Solid curve denotes the performance at standard detection crietria *i.e.* 50% Intersection over Union (IoU) with ground truth. Dotted lines denote less strict but practical detection criteria *i.e.* 40% IoU with ground truth boxes. Regressed coordinate locations from the fixed point model suffer from higher deviation from the floating point model prediction. This effect becomes more evident when we increase the IoU criteria for detection. As we see in Figure 4, for more relaxed IoU criteria such as 40% IoU, floating vs fixed point model exhibit similar detection performances. However, for stricter detection criteria such as 60% IoU or higher, the performance of the fixed-point model appears to drop significantly, especially for the lower false positive regions on the TP-FP curves.

Mobile devices use face detection for several face based quality enhancement processing such as auto-exposure or auto-focus. Any false detection should be highly penalized as their consequences are more expensive. On the other hand, if the detected box overlaps with the actual face by little less than 50%, certain end use cases that use face detection bounding box as input can still function with similar performance. In less strict IoU operating region, LCDet fixed point model is regarded as good as the the LCDet floating point model. Figure 4 shows that the detection rate in those operating points (upto 45% IoU) is similar for floating and fixed LCDet models.

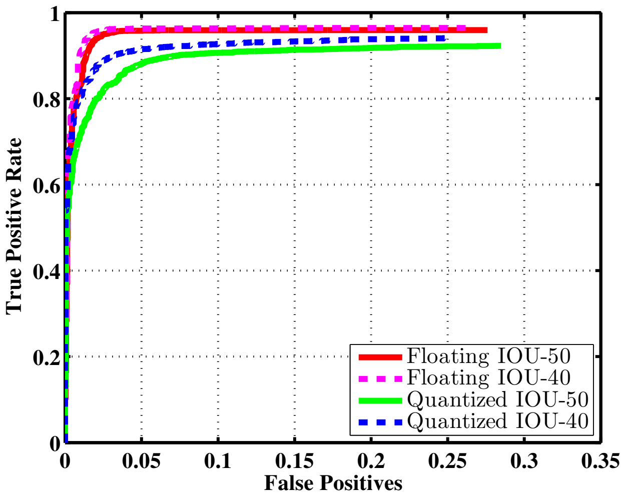

Figure 3. Performance of Fixed vs Floating point models (ReLU) on FDDB with discrete score metric. Effects of quantization on regression is better understood with relaxed detection criteria in terms IOU going down from 50% to 40%. Although the floating point model achieves a little improvement, the fixed point model achieves 5% improvement in true positives. As expected, regression of box coordinates appear to be affected by quantization significantly.

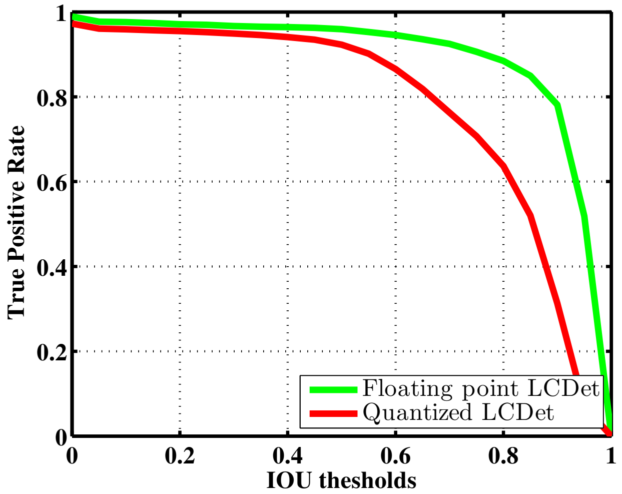

Figure 4. Performance comparison between floating and fixed point models (ReLU) at different IoU thresholds on FDDB.

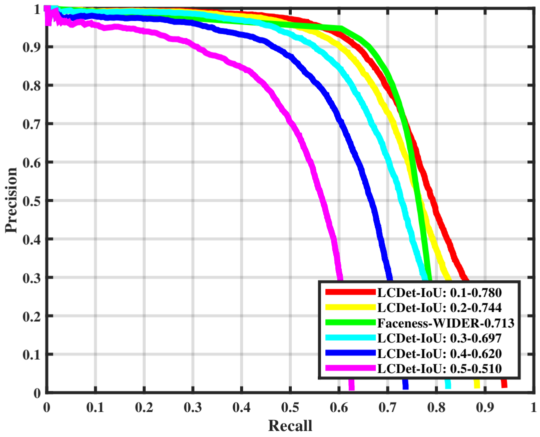

Figure 5. Precision-Recall performance on Widerface Validation set with relaxed IoU criteria.

Next, we train LCDet on the Widerface training split. The Widerface has about 20× more faces than FDDB. some of the faces are extremely small. We evaluate the performance of LCDet on the Widerface validation split using the provided evaluation toolbox. One of the current limitation of the YOLO-style training is that it assumes at most one ground truth object at each grid location, although it can predict up to k objects per grid. For a training image of size 448×448 means 7×7 grids, thus can exploit only 49 ground truth objects. On the other hand, the Widerface has more than 100 ground truth faces in at least 200 training images. YOLO-style training could not use all the ground truth available. In such cases, we used the ground truth with highest area per grid location. On the contrary, faster-RCNN type network can use all possible ground truth objects. As shown in Figure 5, the model needs improvement for the localization accuracies especially for small objects those are present in Widerface. As the IoU criteria is getting relaxed, the model approaches comparable or even better accuracy than other state-of-the-methods such as Faceness [39].

4.3. Complexity and Memory-BandWidth analysis

We first convert the darknet [25] YOLO implementation to TensorFLow-Slim based implementation. We leveraged the weights from the darknet [25] for all of the 24 convolutional layers and first fully connected layer. The number of output nodes in the final detection layer is $W_f \times H_f \times (C + K \times 5)$. For face detection task, $C = 1$, and we use $K = 3$ for all our experiments. Then using the mentioned training methodology, we fine-tune all layers for the face detection task. Table 2 demonstrates the performance and accuracy of these models along with some of the other recent models on powerful GPUs.

The detection-specific module of LCDet uses two convolutional layers. The first one has 4096 kernels of size 3×3 and the second one has 16 kernels of 1×1 size each. Irrespective the backend feature-extractor network, LCDet has 115× fewer parameters for only the detection part comparing with YOLO as described in sec 3.3.

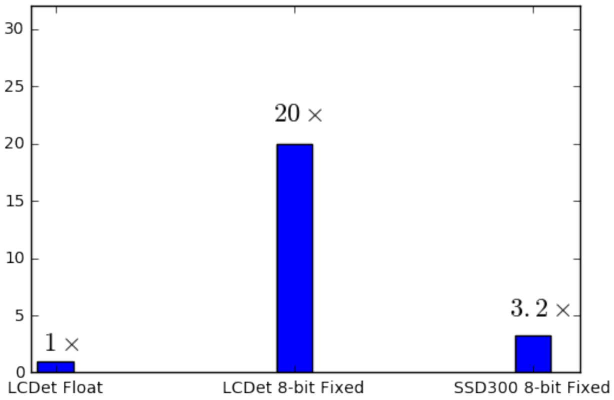

Figure 6. Frame Rate Improvement for Fixed Point Model

Next, we analyze and compare the performance of proposed method with respect to the state-of-the-art deep neural network-based object detector simulated on a commercially available Snapdragon platform such as in [22]. Hexagon DSP includes fixed-point vector extensions which make it an attractive computing unit for computer vision applications and provides performance per power compared to CPU and GPUs on mobile platforms. We quantized our baseline YOLO model and also quantized the proposed LCDet model, and compare their relative model sizes and activation memory footprints for the same input resolution size as shown in Table 3. In Figure 6, we compare the achievable frame rate of the following methods: LCDet-float, LCDet-8bit-fixed, and SSD300-8bit-fixed. In a fixed-point implementation, activations and weights are implemented in 8-bits and mapped to vector extension of DSP, whereas in float implementation vector extension is not utilized. By bringing down the model size and bandwidth (BW) per layer, we achieve close to the 20× increase in frame rate with respect to floating point implementation. The average DDR bandwidth that our quantized model requires is roughly 1 Gbps, whereas the instantaneous BW has a wider range reaching close to 20 Gbps for some layers as shown in Figure 7. Typically, the DDR BW is throttled when multiple applications are run on the embedded systems and frame rate degrades because of stringent BW constraints. This is depicted in Figure 8, where the frame rate drops as DDR BW decreases from 6 Gbps to 1 Gbps.

4.4. Visual Results

We show visual detection results of the face detection performed by the proposed LCDet. Detected faces are marked as *blue* rectangle. Figure 9 to Figure 11 demonstrate face detection results on FDDB dataset. These figures

	Model		Inference	
	Size	FLOPs	Speed	Max
	(MB)	$\times 10^9$	(FPS)	TP
LCDet	250	20	17.55	93.0
YOLO*	1126	20.1	15.44	85.0
Faster-RCNN+VGG16	485	98	4.61 [2]	96.1 [13]
SSD300+VGG16[16, 19]	105	31.6	17.97[19]	NA

Table 2. LCDet vs other methods. Inference speed on NVidia Tesla K40. YOLO* is our TF-implementation for Face Detection. Max TP is the highest true positive rate value achieved in FDDB. SSD and Faster R-CNN running time use TF-SSD implementation [19] and TF Faster RCNN implementation [2] respectively in different implementation platforms.

			Activation
	Model		Memory
	Size	OPs	Footprint
	(MB)	$\times 10^9$	(MB)
Quantized YOLO	281	20.1	333
Quantized LCDet	62	20.0	88

Table 3. Performance analysis of Fixed-point LCDet in terms of OPs, Memory Activation Footprints

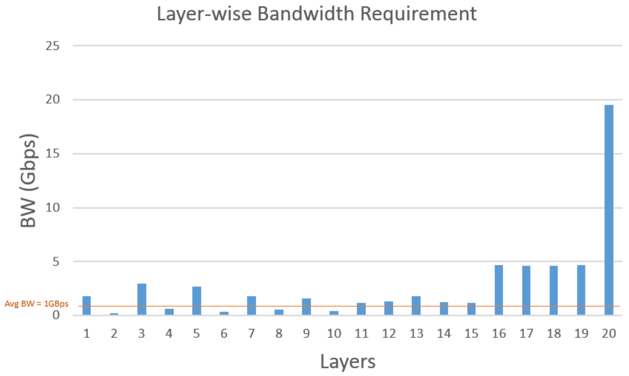

Figure 7. Layer-wise Bandwidth Requirement for LCDet-8bit-fixed Point Implementation

show that LCDet detects multiple faces of different sizes and poses accurately for a variety of illumination and scale changes. Some of the difficult examples from the Wider-face validation dataset are shown in Figure 12. In general, LCDet performs well in detecting faces.

Current limitation of the model is that it struggles in tightly localizing small objects in close proximity. Figure 13 demonstrates some of the examples where localization might have failed as per strict 50% IoU criteria (marked as *yellow* regions), however the detected faces are not false positives for further face-based processing pipeline. The regions marked in *red* shows missed detections.

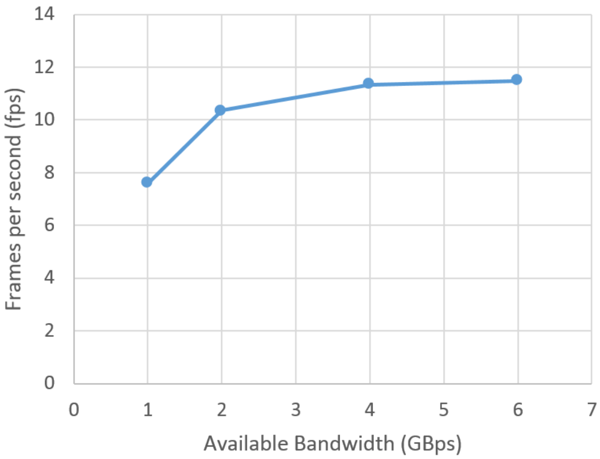

Figure 8. Effect of available DDR BW on Frame Rate

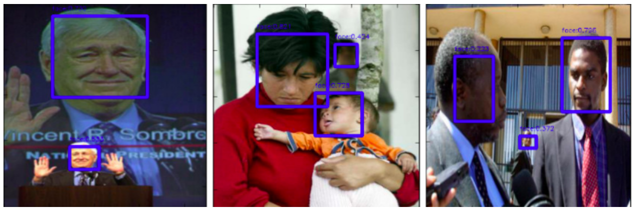

Figure 9. Detected faces of different scales on FDDB.

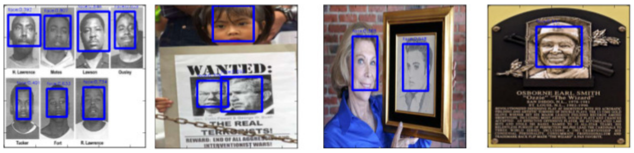

Figure 10. Faces detected in black and white images on FDDB.

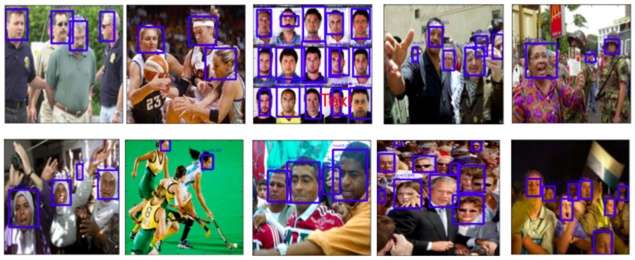

Figure 11. Multiple faces of frontal and side profiles on FDDB.

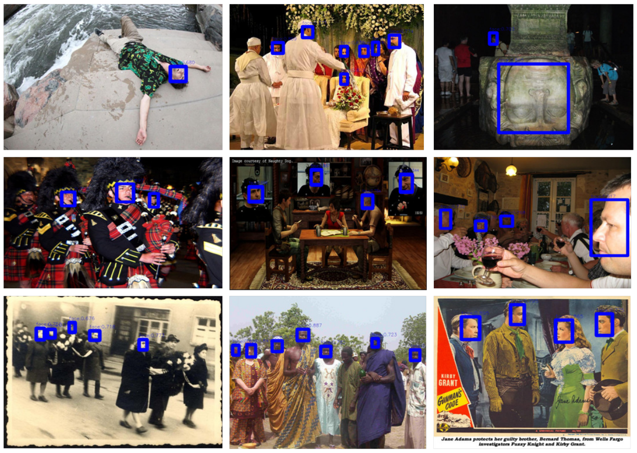

Figure 12. Successful face detection results of LCDet on challenging Widerface Validation images containing difficult examples in pose variation, illumination changes, photograph styles, and different sizes.

5. Conclusions

We propose LCDet, a low-complexity fully-convolutional neural network for object detection amenable for embedded deployment. This is a unified localization and classification model inspired by [23] that bypasses the object proposals bottleneck. LCDet performs comparably with state-of-the-art CNN-based face detection methods on FDDB, while being one of the most computationally effective method. We additionally perform 8-bit quantization on this TF-slim based LCDet model, and report one of the first analysis of quantized model for regression. The quantized LCDet model performs as good as floating point model and reduces the memory footprint by $4\times$. Quantization makes this model apt for implementation in DSPs, or dedicated convolution accelerators. Although, face detection is studied as a use case here, the network is not optimized for face-specific detection only. It is easily expendable for detecting any other categories of objects.

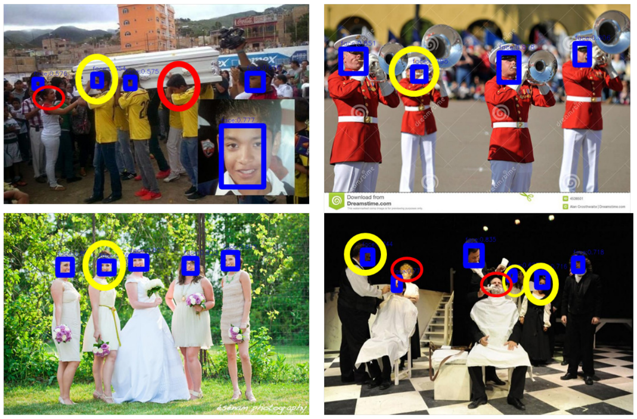

Figure 13. Localization challenges of LCDet on Widerface validation images. Faces marked as the *yellow* regions are considered false positives as per 50% IoU criteria, but true positive for more relaxed IoU criteria. The regions marked in *red* show missed detections.

We are aware of the very recent work YOLO9000 [24] that has become the state-of-the-art on standard object detection datasets. YOLO9000 that is an improved version of YOLO. Empirically, the accuracy of LCDet lies between YOLO [23] and YOLO9000 [24]. On the other hand, another recent work SqueezeDet [36] on low-complexity CNN achieves state-of-the-art performance on KITTI object detector. SqueezeDet appears to be the smallest object detector by virtue of powerful but small backend network of SqueezeNet [11]. At the time of writing, there is no reported performance comparison for all these on the same dataset. As a future work, we will evaluate the relative performance of our proposed model comparing with these methods. It is also interesting to explore SqueezeNet as the backend and study performance of quantization.

Acknowledgments

The authors would like to thank Vikram Gupta and Rakesh Nattoji Rajaram for extensive assistance and insightful comments.

References

[1] K. Ashraf, B. Wu, F. N. Iandola, M. W. Moskewicz, and K. Keutzer. Shallow networks for high-accuracy road object-detection. *CoRR*, abs/1606.01561, 2016. 2

[2] F.-H. Chan. Faster rcnn tensorflow. https://github.com/smallcorgi/Faster-RCNN_TF. 7

[3] Y.-H. Chen, J. Emer, and V. Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *Proceedings of the 43rd International Symposium on Computer Architecture*, ISCA '16, pages 367–379, Piscataway, NJ, USA, 2016. IEEE Press. 2

[4] M. Courbariaux, Y. Bengio, and J. David. Low precision arithmetic for deep learning. *CoRR*, abs/1412.7024, 2014. 4

[5] M. Everingham, L. Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *Int. J. Comput. Vision*, 88(2):303–338, June 2010. 2, 4

[6] S. Gidaris and N. Komodakis. Object detection via a multi-region and semantic segmentation-aware cnn model. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015. 2

[7] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CVPR*, 2014. 2

[8] R. B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015. 2

[9] P. Gysel. Ristretto: Hardware-oriented approximation of convolutional neural networks. *CoRR*, abs/1605.06402, 2016. 4

[10] 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. *CoRR*, abs/1611.10012, 2016. 2

[11] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016. 8

[12] V. Jain and E. Learned-Miller. Fddb: A benchmark for face detection in unconstrained settings. Technical Report UM-CS-2010-009, University of Massachusetts, Amherst, 2010. 1, 2, 4, 5

[13] H. Jiang and E. G. Learned-Miller. Face detection with the faster R-CNN. *CoRR*, abs/1606.03473, 2016. 2, 7

[14] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. 4

[15] Y. Li, B. Sun, T. Wu, and Y. Wang. Face detection with end-to-end integration of a convnet and a 3d model. In *ECCV*, 2016. 2

[16] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, and S. E. Reed. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015. 2, 7

[17] H. Mao, S. Yao, T. Tang, B. Li, J. Yao, and Y. Wang. Towards real-time object detection on embedded systems. *IEEE Transactions on Emerging Topics in Computing*, 2016. 2

[18] Modivious. Miriad vpu. `https://www.movidius.com/?49b390`. 2

[19] B. Paul. Ssd-tensorflow. `https://github.com/balancap/SSD-Tensorflow`. 7

[20] H. Qin, J. Yan, X. Li, and X. Hu. Joint training of cascaded cnn for face detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 2

[21] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, Y. Wang, and H. Yang. Going deeper with embedded fpga platform for convolutional neural network. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '16, pages 26–35, New York, NY, USA, 2016. ACM. 2

[22] Qualcomm. Snapdragon 835 processor, 2017. `https://www.qualcomm.com/products/snapdragon/processors/835`. 2, 6

[23] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 1, 2, 3, 4, 8

[24] J. Redmon and A. Farhadi. YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242, 2016. 8

[25] J. C. Redmon. Darknet. 4, 6

[26] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: towards real-time object detection with region proposal networks. In *NIPS*, pages 91–99, 2015. 2, 3

[27] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *CoRR*, abs/1312.6229, 2013. 2

[28] Snapdragon-blog. Tensorflow machine learning now optimized for the snapdragon 835 and hexagon 682 dsp, 2017. https://www.qualcomm.com/news/snapdragon/2017/01/09/tensorflow-machine-learning-now-optimized-snapdragon-835-and-hexagon-682. 2

[29] X. Sun, P. Wu, and S. C. H. Hoi. Face detection using deep learning: An improved faster RCNN approach. *CoRR*, abs/1701.08289, 2017. 2

[30] C. Szegedy, S. E. Reed, D. Erhan, and D. Anguelov. Scalable, high-quality object detection. *CoRR*, abs/1412.1441, 2014. 2

[31] TensorFlow. Quantization. `https://www.tensorflow.org/performance/quantization`. 4

[32] TensorFlow. Tensorflow for poets. https://petewarden.com/2015/05/23/why-are-eight-bits-enough-for-deep-neural-networks/ . 4

[33] V. Vanhoucke, A. Senior, and M. Z. Mao. Improving the speed of neural networks on cpus. In *Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011*, 2011. 4

[34] S. Wan, Z. Chen, T. Zhang, B. Zhang, and K. Wong. Bootstrapping face detection with hard negative examples. *CoRR*, abs/1608.02236, 2016. 2

[35] P. Warden. Pete warden's blog. https://petewarden.com/2016/05/03/how-to-quantize-neural-networks-with-tensorflow/ . 4

[36] B. Wu, F. N. Iandola, P. H. Jin, and K. Keutzer. Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. *CoRR*, abs/1612.01051, 2016. 8

[37] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng. Quantized convolutional neural networks for mobile devices. *CoRR*, abs/1512.06473, 2015. 4

[38] B. Yang, J. Yan, Z. Lei, and S. Z. Li. Craft objects from images. *CVPR*, 2016. 2

[39] S. Yang, P. Luo, C. C. Loy, and X. Tang. From facial parts responses to face detection: A deep learning approach. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 3676–3684, Dec 2015. 2, 6

[40] S. Yang, P. Luo, C. C. Loy, and X. Tang. Wider face: A face detection benchmark. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 1, 4

[41] J. Yu, Y. Jiang, Z. Wang, Z. Cao, and T. Huang. Unitbox: An advanced object detection network. In *Proceedings of the 2016 ACM on Multimedia Conference*, MM '16, pages 516–520, New York, NY, USA, 2016. ACM. 2