# IFQ-Net: Integrated Fixed-point Quantization Networks for Embedded Vision

Hongxing Gao, Wei Tao, Dongchao Wen
Canon Information Technology (Beijing) Co., LTD
{gaohongxing,taowei,wendongchao}@canon-ib.com.cn

Tse-Wei Chen, Kinya Osa, Masami Kato
Device Technology Development Headquarters, Canon Inc.
twchen@ieee.org

## Abstract

*Deploying deep models on embedded devices has been a challenging problem since the great success of deep learning based networks. Fixed-point networks, which represent their data with low bits fixed-point and thus give remarkable savings on memory usage, are generally preferred. Even though current fixed-point networks employ relative low bits (*e.g. *8-bits), the memory saving is far from enough for the embedded devices. On the other hand, quantization deep networks, for example XNOR-Net and HWGQ-Net, quantize the data into 1 or 2 bits resulting in more significant memory savings but still contain lots of floating-point data. In this paper, we propose a fixed-point network for embedded vision tasks through converting the floating-point data in a quantization network into fixed-point. Furthermore, to overcome the data loss caused by the conversion, we propose to compose floating-point data operations across multiple layers (*e.g. *convolution, batch normalization and quantization layers) and convert them into fixed-point. We name the fixed-point network obtained through such integrated conversion as Integrated Fixed-point Quantization Networks (IFQ-Net). We demonstrate that our IFQ-Net gives 2.16× and 18× more savings on model size and runtime feature map memory respectively with similar accuracy on ImageNet. Furthermore, based on YOLOv2, we design IFQ-Tinier-YOLO face detector which is a fixed-point network with 256× reduction in model size (246k Bytes) than Tiny-YOLO. We illustrate the promising performance of our face detector in terms of detection rate on Face Detection Data Set and Bencmark (FDDB) and qualitative results of detecting small faces of Wider Face dataset.*

## 1. Introduction

During the past decade, deep learning models have achieved great success on various machine learning tasks such as image classification, object detection, semantic segmentation, etc. However, applying them on embedded devices remains as a challenging problem due to the enormous resource requirement in terms of memory and computation power. On the other hand, fixed-point data inference yields promising reductions on such requirement for embedded devices [6]. Thus, fixed-point networks are primarily preferred when deploying deep models for the embedded devices.

In general, designing a fixed-point CNN network can be fulfilled by two types of approaches: 1)pre-train a floating-point deep network and then convert it into a fixed-point network; 2) train a deep CNN model whose data (*e.g.* weights, feature maps, etc.) is natively fixed-point. In [9], a method is introduced to find the optimal bit-width for each layer to convert its floating-point weights and feature maps into their fixed-point counterparts. Given the hardware acceleration for 8-bit integer based computations, [12] provides optimal thresholds which minimize the data loss during the 32-bits float to 8-bits integer conversion. These works have shown that it is feasible to significantly save memory usage through relatively low bit (*e.g.* 8-bits) representation yet achieve similar performance. However, such memory saving is far from enough especially for embedded devices. The second approach is to train a network all of whose data is natively fixed-point. Nevertheless, as discussed in [8], its training process may suffer from severe unstable weight updating because of the inaccurate gradients. Strategies such as stochastic rounding somehow result in improvement [3, 4, 10] but a trade-off between low bit data representation and precise gradients still has to be made.

Alternatively, BinaryNet [2] employs binarized weights for forward pass but full precision weights and gradients for stable convergence. Meanwhile, its feature maps are also binarized to $\{-1, +1\}$ so that its data can be represented as 1-bit fixed-point for less memory usage during inference time. However, a notable performance drop of

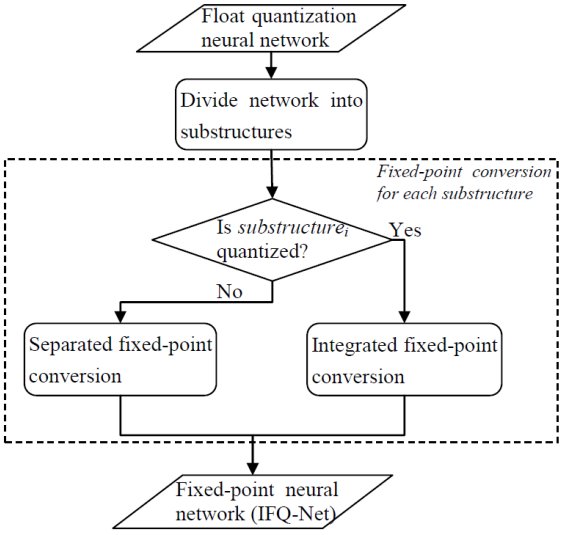



Figure 1. The flowchart of converting a floating-point quantization network into IFQ-Net.

30% (Top-1 accuracy) is observed on ImageNet classification. Subsequently, XNOR-Net [13] employs extra scaling factors on both weights and feature maps so that their "binary" elements are generalized to $\{-\alpha, +\alpha\}$ and $\{-\beta, +\beta\}$ respectively. These extra factors enrich the data information thus gains 16% accuracy back on ImageNet. Furthermore, HWGQ-Net [1] uses a more flexible k-bits quantization on feature maps whose elements can be generalized to $\{0, \beta, 2\beta, 3\beta\}$ in the situation of 2-bits uniform quantization. Such $k$-bits feature maps ($k \geq 2$) give a further 8% improvement making HWGQ-Net to be the state-of-the-art quantization network on ImageNet classification.

Given a HWGQ-Net, each filter of its quantized convolution layer can be expressed as a multiplication of a floating-point $\alpha$ and a binary fixed-point matrix whose elements are limited to $\{-1,+1\}$. Similar representations can also be applied to its feature maps (see Equation 1). Therefore, to obtain its fixed-point counterpart, it would be only necessary to convert the floating-point $\alpha$ and $\beta$ while other parts of the layer are natively fixed-point. Besides, Batch Normalization (BN) layer, which is usually employed on top of each convolution layer, also contains floating-point parameters and thus requires fixed-point conversion (see Equation 2). One way to do this is to separately convert each of the floating-point data but it usually results in data loss that would be accumulated over the network and cause a notable performance drop.

In this paper, we propose a novel fixed-point network, IFQ-Net which is obtained through converting a floating-point quantization network into its fixed-point counterparts. As illustrated in Figure 1, we first divide the quantization network into several substructures, where each substructure is defined as a group of consecutive layers that starts with a convolution layer and ends with a quantization layer. An example of the substructures of AlexNet is listed in Table 1. Then we convert the floating-point data in each substructure into fixed-point data. Especially for the "quantized substructure", which starts with a quantized convolution layer and ends with a quantization layer, we propose to compose its floating-point data into the thresholds of the quantization layer and then convert the composition result into fixed-point. As will be presented in section 3.2, our integrated conversion method does not cause any performance drop. At the end, we separately convert each floating-point data in the remaining non-quantized substructures (if any) to fixed-point resulting in a fixed-point network, IFQ-Net.

In this paper, the major contributions we made are:

- proposing IFQ-Net network, obtained through converting a floating-point quantization network into fixed-point. Due to the relatively low bits of the quantization network, IFQ-Net gives much more savings on model size and runtime feature map memory.

- proposing an integrated conversion method to convert the floating-point data in the quantized substructures without performance drop. Since its BN operation (if available) is already integrated into the thresholds of the corresponding quantization layer, our IFQ-Net does not require actual BN implementation on target hardware.

- designing IFQ-Tinier-YOLO face detector, a fixed-point model with 256× tinier model size (246k Bytes) than Tiny-YOLO.

- demonstrating the feasibility of quantizing all convolution layers in IFQ-Tinier-YOLO model, which differs from the original HWGQ-Net whose first and last layers are full precision.

## 2. Quantized convolutional neural network

A CNN network usually consists of a series of layers where the convolution layer monopolizes the inference time of the whole network. However, the weights and features maps were found redundant for most tasks. Consequently, enormous efforts have been done on quantizing the weights and/or the input feature maps into low-bit data for less memory usage and fast computation.

### 2.1. Quantization network inference

Embedded devices are usually employed for network inference only because of their limited computation resources. Hence, in this paper, we mainly focus on the inference process of a network. In the following, we take a

typical quantized substructure from HWGQ-Net as an example to illustrate the computation details of its inference.

For the $l$-th convolution layer of HWGQ-Net, we use $\mathbf{W} \in \mathbb{R}^{c\times h'\times w'}$ and $\mathbf{X}^l \in \mathbb{R}^{c\times h\times w}$ to represent one of the filters and its input feature maps respectively, where $c$, $h'$, $w'$, $h$, $w$ are the number of channels, the height and width of its filter, and the height and width of the input feature maps respectively. In the case of a 2-bit quantized convolution layer from HWGQ-Net, its filter is binarized into $\mathbf{W} \in \{-\alpha, +\alpha\}^{c\times h'\times w'}$ and $\mathbf{X}^l \in \{0, \beta, 2\beta, 3\beta\}^{c\times h\times w}$. Then the computation of a convolution layer can be represented as

$$\mathbf{Y}^{conv} = \mathbf{W} \otimes \mathbf{X}^l + b = \alpha\beta \cdot \mathbf{W}_b \otimes \mathbf{X}^l_q + b \qquad (1)$$

where $\otimes$ represents the convolution operation; $\mathbf{W}_b$ and $\mathbf{X}^l_q$ are integer part of the quantized filter and feature maps so that $\mathbf{W}_b \in \{-1, +1\}^{c\times h'\times w'}$ and $\mathbf{X}^l_q \in \{0, 1, 2, 3\}^{c\times h\times w}$, $b$ is its learned bias.$\mathbf{Y}^{conv}$ is its output feature map.

Typically, a BN layer is applied on top of a convolution layer. It is computed in an element-wise manner as follows,

$$\mathbf{Y}^{BN}_{i,j} = \frac{\mathbf{Y}^{conv}_{i,j} - \theta}{\sigma} \qquad (2)$$

where $0 \le i < w$, $0 \le j < h$, $\theta$ and $\sigma$ are the learned mean and variance of the feature map.

At the end, a quantization layer maps its input feature map $\mathbf{Y}^{BN}$ into discrete numbers. Taking a 2-bits uniform quantization for instance, its computation can be expressed as

$$\mathbf{X}^{l+1}_{i,j} = \beta' * \begin{cases} 0 & \mathbf{Y}^{BN}_{i,j} \le thr_1 \\ 1 & thr_1 < \mathbf{Y}^{BN}_{i,j} \le thr_2 \\ 2 & thr_2 < \mathbf{Y}^{BN}_{i,j} \le thr_3 \\ 3 & \mathbf{Y}^{BN}_{i,j} > thr_3 \end{cases} \qquad (3)$$

where $thr_1$, $thr_2$ and $thr_3$ are the thresholds used for quantizing its input $\mathbf{Y}^{BN}$, and $\beta'$ is the scale factor for its output feature map. The resulting $\mathbf{X}^{l+1}$ is then employed as the input of the $(l+1)$-th convolution layer (if available).

When max pooling layer appears in the substructure, as discussed in [13], it is better to place it between convolution and BN layers for richer data information. In other words,

$$\mathbf{Y}^{pooling}_{i,j} = \max_{(m,n)\in A_{i,j}} \{\mathbf{Y}^{conv}_{m,n}\} \qquad (4)$$

where $A_{i,j}$ denotes the local zone employed for pooling operation at location $(i, j)$ of $\mathbf{Y}^{conv}$. Then the input of the BN layer in Equation 2 is accordingly changed to be $\mathbf{Y}^{pooling}_{i,j}$ .

## 2.2. Separated fixed-point conversion

As illustrated in subsection 2.1, the dominating part of the convolution computation $\mathbf{W} \otimes \mathbf{X}^l$ can be implemented

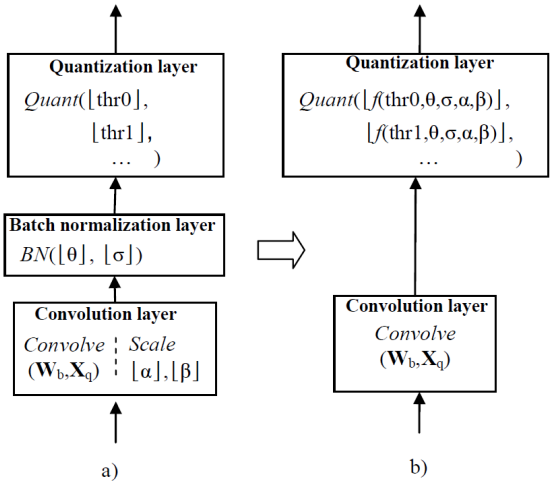



Figure 2. Fixed-point conversion for a substructure: a) separated conversion which separately transforms each floating-point data into fixed-point data through the floor function $\lfloor\cdot\rfloor$ ; b) integrated conversion which employs a composition operation $f$ to compose all the floating-point calculations to quantization layer and then apply the fixed-point conversion for the composed results.

with native fixed-point data only. However, the network still contains lots of floating-point data these being the scaling factor $\alpha$ and $\beta$ in the convolution layer, $\theta$ and $\sigma$ in the BN layer and also $thr_i$ in the quantization layer. Consequently, it is necessary to convert them into fixed-point when designing fixed-point networks for embedded devices.

A traditional way for the aforementioned conversion is to process them separately. As shown in Figure 2a, each floating-point data of the substructure is converted into its fixed-point counterpart. Since directly applying a simple conversion causes significant data loss especially when $\alpha$ is small (*e.g.* 0.001), we use a relatively large $Q_m$ to scale-up the floating-point data[1]. For example, $\alpha$ can be transformed by $\lfloor\alpha Q_m\rfloor$ where $\lfloor\cdot\rfloor$ denotes the flooring operation. At the end, $Q_m$ has to be divided back to achieve "equivalent" outputs. Then, fixed-point conversion of a quantized convolution layer can be expressed as

$$\mathbf{Y}^{conv} = \frac{\lfloor\alpha Q_m\rfloor\lfloor\beta Q_m\rfloor \cdot \mathbf{W}_b \otimes \mathbf{X}_q + \lfloor bQ^2_m\rfloor}{Q^2_m} \qquad (5)$$

To obtain a substructure with fixed-point data only, the same conversion $\lfloor\cdot\rfloor$ is also applied to $\theta$, $\sigma$, $thr_i$ separately.

# 3. IFQ-Net methodology

To obtain a fixed-point network for embedded devices, we propose to first train a quantization network and then

[1]For fast calculation, $Q_m$ is usually set to $2^m$ so that the multiplication can be implemented by simple $m$-bit left shift

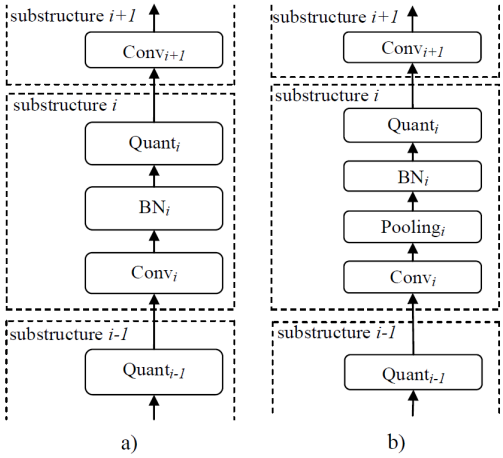



Figure 3. Substructure division for a quantized network: a)substructure without max pooling layer; b)substructure with max pooling layer.

convert its floating-point data, which has been quantized into extremely low bits (*e.g.* 1 or 2 bits), into fixed-point data. As demonstrated in Figure 1, our methodology consists of two steps: first we divide a trained floating-point quantization network into substructures and then we convert each substructure into its fixed-point counterpart. We employ HWGQ-Net algorithm to train a floating-point quantization network.

### 3.1. Substructure division

As mentioned in Section 1, a substructure is defined as a group of consecutive layers that starts with a convolution layer and ends with a quantization layer. Given a quantization network, we search for the quantized substructures in the network as demonstrated in Figure 3. Typically, the architecture of a quantized substructure is either {convolution, BN, quantization} or {convolution, pooling, BN, quantization}. The substructures that contain more than one convolution or quantization layer are not considered as quantized substructures. The layers between quantized substructures are defined as non-quantized substructures, which will be treated differently during fixed-point conversion. Generally, BN and/or max pooling layers are placed between convolution layers and quantization layers.

Taking AlexNet-HWGQ network as an example, we divide it into 7 substructures (see Table 1). Because the HWGQ network keeps its first and last convolution layer full precision, so the corresponding substructures ($substructure1$ and $substructure7$) are non-quantized and thus will be converted differently. Please note that we group all the layers on top of the $FC7$ layer as one single substructure.

### 3.2. Integrated fixed-point conversion

A trained quantization network can be divided into substructures that contain lots of floating-point data. To obtain a fixed-point network, it is necessary to convert each of its floating-point substructures into fixed-point. However, converting the floating-point data in a separated manner usually leads to performance drop. Consequently, in the following, we introduce an integrated way to convert a floating-point substructure. Taking 2-bits uniformly quantized substructure from HWGQ-Net as an example, its computations that mentioned in Equation 1, 2 and 3 can be composed as follows

$$\mathbf{Y}^{quant} = \begin{cases} 0 & \frac{\alpha\beta\cdot\mathbf{W}_b\otimes\mathbf{X}_q+b-\theta}{\sigma} \le thr_1 \\ \beta' & thr_1 < \frac{\alpha\beta\cdot\mathbf{W}_b\otimes\mathbf{X}_q+b-\theta}{\sigma} \le thr_2 \\ 2\beta' & thr_2 < \frac{\alpha\beta\cdot\mathbf{W}_b\otimes\mathbf{X}_q+b-\theta}{\sigma} \le thr_3 \\ 3\beta' & \frac{\alpha\beta\cdot\mathbf{W}_b\otimes\mathbf{X}_q+b-\theta}{\sigma} > thr_3 \end{cases} \quad (6)$$

Since $\alpha > 0$, $\beta > 0$ and also $\sigma > 0$, Equation 6 can be transformed to

$$\mathbf{Y}^{quant} = \beta' * \begin{cases} 0 & \mathbf{W}_b \otimes \mathbf{X}_q \le \frac{thr_1*\sigma+\theta-b}{\alpha\beta} \\ 1 & \frac{thr_1*\sigma+\theta-b}{\alpha\beta} < \mathbf{W}_b \otimes \mathbf{X}_q \le \frac{\sigma*thr_2+\theta-b}{\alpha\beta} \\ 2 & \frac{\sigma*thr_2+\theta-b}{\alpha\beta} < \mathbf{W}_b \otimes \mathbf{X}_q \le \frac{\sigma*thr_3+\theta-b}{\alpha\beta} \\ 3 & \mathbf{W}_b \otimes \mathbf{X}_q > \frac{\sigma*thr_3+\theta-b}{\alpha\beta} \end{cases} \quad (7)$$

As illustrated in Equation 7, all the floating-point data of a quantized substructure is composed into the newly formed thresholds (*e.g.* $\frac{thr_1*\sigma+\theta-b}{\alpha\beta}$). Such composition process is performed with floating-point data and thus does not impact the output result.

The next step is to convert the new thresholds into fixed-point data. $\mathbf{W}_b$ and $\mathbf{X}_q$ are both integers thus the resulted $\mathbf{W}_b \otimes \mathbf{X}_q$ are also integers. In Equation 7, when thresholding the integers $\mathbf{W}_b \otimes \mathbf{X}_q$ with newly formed floating-point thresholds, theoretically, their fractional parts do not affect the result. Consequently, we can simply discard the fractional part by applying the floor function $\lfloor\cdot\rfloor$ on the new thresholds. Compared to the separated fixed-point conversion method, our method does not require to scale-up the floating-point data with $Q_m$ yet gives identical quantization results. Besides, the remaining floating-point data $\beta'$ can be processed in the next substructure just like how we deal with the $\beta$ of Equation 1. Consequently, all the computations of a quantized substructure, as represented in Equation 7, can be casted on fixed-point data after $\lfloor\cdot\rfloor$ is applied on each of the new thresholds.

The fixed-point implementation of on-device BN computation is challenging for embedded devices. As an alternative solution which differs from the method that merges it into a convolution layer, the proposed integrated fixed-point conversion method transforms the BN computation into the

new quantization thresholds. Consequently, our IFQ-Net also does not require actual BN implementation on embedded hardware.

In the above, we have taken $k = 2$ as an example of converting a $k$-bits quantization network. However, when a larger $k$ is employed, it would be necessary to store $(2^k - 1)$ thresholds. In the uniform quantization scenario, the network's thresholds can be expressed as $thr_i = i * base + offset$. Thus, one may only need to store $base$ and $offset$ because all the thresholds can be restored from them. Similarly, denoting $base' = \frac{\sigma * base}{\alpha\beta}$ and $offset' = \frac{\theta - b + \sigma * offset}{\alpha\beta}$, our newly formed thresholds can also be represented as $thr'_i = i * base' + offset'$. Thus, our new thresholds $thr'$ can also be represented in an efficient way. Then computations in a k-bits uniformly quantized substructure can be expressed as Equation 8 which can be further converted into fixed-point in an integrated manner.

$$\mathbf{Y}^{quant} = \beta' * \begin{cases} 0 & \mathbf{W}_b \otimes \mathbf{X}_q \leq thr'_1 \\ 1 & thr'_1 < \mathbf{W}_b \otimes \mathbf{X}_q \leq thr'_2 \\ 2 & thr'_2 < \mathbf{W}_b \otimes \mathbf{X}_q \leq thr'_3 \\ \vdots & \vdots \\ (2^k - 1) & \mathbf{W}_b \otimes \mathbf{X}_q > thr'_{(2^k-1)} \end{cases} \tag{8}$$

In summary, we presented IFQ-Net obtained by dividing a quantization network (*e.g.* HWGQ-Net) into floating-point substructures and then converting each of them into fixed point. For the quantized substructures, we propose an integrated fixed-point conversion method which gives no performance drop. At the end, for the remaining non-quantized substructure (if any), we employ the separated method to convert them into fixed-point.

It is worth to point out that our IFQ-Net differs from the floating-point data composition method presented in [17] in many aspects: 1)the paper claims that it combines multiple layers but does not explicitly explain how; 2)the paper applies the floating-point data composition for enabling binary convolution computation leaving other parameters as floating-point while our method is proposed for fixed-point conversion and 3)the paper concentrates on implementing a quantized network on FPGA but the performance (*e.g.* detection rate, mAP etc.) is not reported.

# 4. Experimental results

In this section, we demonstrate how we convert each substructure of AlexNet into fixed-point to obtain an IFQ-AlexNet. We first test the performance of the integrated conversion method for the quantized substructures. Then, for the non-quantized substructures, we demonstrate how we experimentally set the scale factor $Q_m$ for the separated fixed-point conversion. We compare the performance of our IFQ-AlexNet with "Lin *et al.* [9]" which is the state-of-the-art AlexNet-based fixed-point network on ImageNet. Furthermore, we also illustrate the performance of IFQ-Tinier-YOLO face detector which is an extremely compact fixed-point network on both FDDB and Wider Face datasets.

## 4.1. IFQ-AlexNet network

To obtain fixed-point networks, we first train floating-point quantization networks AlexNet-HWGQ whose weights and feature maps are quantized into 1-bit and $k$-bits ($k \in \{2, 3, 4\}$) respectively. The AlexNet-HWGQ is trained with 320k iterations on ImageNet while the batch size is set to 256. The initial learning rate is set to 0.1 and decreased by a factor of 0.1 every 35k iterations. We inherit other training settings from [1] and achieve similar performance.

Table 1. Substructures of AlexNet-HWGQ network.

| $substructure1$ | $substructure2$ | ... | $substructure7$ |
|---|---|---|---|
| $\text{Conv}_1$ | $\text{Conv}_2^q$ | | $\text{FC}_7^q$ |
| $\text{Pool}_1$ | $\text{Pool}_2$ | | $\text{BN}_7$ |
| $\text{BN}_1$ | $\text{BN}_2$ | ... | $\text{ReLU}_7$ |
| $\text{Quant}_1$ | $\text{Quant}_2$ | | $\text{FC}_8$ |

As the first step for obtaining the IFQ-AlexNet, we divide a floating-point AlexNet-HWGQ network into 7 substructures ( Table 1). In the table, the superscript $q$ in $\text{Conv}_2^q$ and $\text{FC}_7^q$ means that their weights are binarized (1-bit) and input feature maps are quantized into $k$ bits by their bottom $\text{Quant}_i$ layers. We group the layers $\{\text{FC}_7^q, \text{BN}_7, \text{ReLU}_7, \text{FC}_8\}$ as a single non-quantized substructure.

In the following, we will show how to convert each substructure into fixed-point to obtain an IFQ-AlexNet. In subsection 4.1.1, we show the performance of the proposed integrated conversion method for the quantized substructure while the non-quantized substructures are kept floating-point. We then illustrate the way to set a proper scaling factor $Q_m$ for converting each of the remaining non-quantized substructures (see subsection 4.1.2). At the end, in subsection 4.1.3, we compare our IFQ-AlexNet with "Lin *et al.* [9]" which is the state-of-the-art AlexNet-based fixed-point network.

### 4.1.1 Integrated conversion for the quantized substructures

In this subsection, we focus on converting the quantized substructures ($substructure2$,...,$substructure6$). The ImageNet Top-1 classification accuracy is employed to evaluate the accuracy of the converted networks (see Table 2). In the table, "separated" refers to the networks obtained by converting the quantized substructures of the corresponding AlexNet-HWGQ ($k$ equals to 2 or 3 or 4) in a separated manner (see section 2.2). In contrast, "integrated"

represents the networks obtained by converting their quantized substructures in the proposed integrated way (see section 3.2). Please note that, to compare the performance of different conversion methods on quantized substructures, we keep the non-quantized substructures ($substructure1$ and $substructure7$) floating-point.

Table 2. Performance of different methods on converting the quantized substructures of AlexNet-HWGQ networks on ImageNet top-1 classification accuracy.

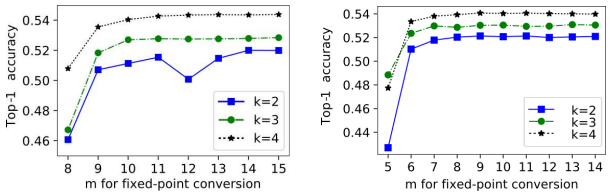
| | $k=2$ | $k=3$ | $k=4$ |
|---|---|---|---|
| AlexNet-HWGQ | 0.5214 | 0.5301 | 0.5471 |
| separated($m=12$) | 0.5206 | 0.5296 | 0.5470 |
| separated($m=10$) | 0.5168 | 0.5292 | 0.5443 |
| separated($m=9$) | 0.5073 | 0.5230 | 0.5385 |
| separated($m=8$) | 0.4585 | 0.4678 | 0.5105 |
| integrated($m=0$) | **0.5214** | **0.5301** | **0.5471** |

As shown in Table 2, the floating-point AlexNet-HWGQ networks achieves competitive classification accuracy. However, "separated" method shows notable performance degradation. The reason is that it separately converts each floating-point data $x$ of a quantized substructures by $\lfloor x * Q_m \rfloor$ which leads to data loss. To reduce such loss, a large $m$ has to be applied ($m=12$) which in turn causes more memory usage. In contrast, for each quantized substructure, our "integrated" method gives identical outputs as its floating-point counterpart in AlexNet-HWGQ while the scaling factor $Q_m$ is not required at all ($m=0$). Even though we employ the uniform quantization as example, our "integrated" method is also effective for the networks quantized by other strategies as long as their floating-point operations can be composed as in Equation 7.

#### 4.1.2 Separated conversion for the non-quantized substructures

In the subsection 4.1.1, we have demonstrated that the proposed integrated method gives lossless fixed-point conversion for quantized substructures. To obtain IFQ-AlexNet all of whose data operations are fixed-point data based, we then convert each of the remaining non-quantized substructures in a "separated" manner. For saving more memory while causing less conversion loss for such substructures, an optimal $Q_m$ is required for each of non-quantized substructure: $substructure1$ and $substructure7$. Since the $substructure7$ directly outputs the inferred results for the task, the preciseness of its computation is more critical. Consequently, we first find the optimal $m$ for its fixed-point conversion while $substructure1$ is kept floating-point.

As demonstrated in Figure 4a), for the networks with different $k$, a larger $m$ for $Q_m$ generally gives better performance. It is because a larger $m$ value gives less data loss during each fixed-point conversion $\lfloor x * 2^m \rfloor$. Nevertheless, when $m \geq 14$, no further performance improvement can be observed for all the three networks indicating $m=14$ would be sufficient for fixed-point conversion for the floating-point data in $substructure7$.




a):$subtructure7$ b):$substructure1$

Figure 4. Top-1 accuracy on ImageNet of networks with various $m$ for $subtructure7$ and $substructure1$ fixed-point conversion.

By fixing $m=14$ for converting $substructure7$ into fixed-point, we then optimize the $m$ for $substructure1$. As shown in Figure 4b), $m=9$ can be considered as the sufficient scaling factor for the fixed-point conversion of $substructure1$.

In summary, to obtain IFQ-AlexNet, we employ the lossless "integrated" conversion method for the quantized substructures and $m=9$ and $m=14$ for the scaling factor $Q_m$ for converting the $substructure1$ and $substructure7$ of AlexNet-HWGQ networks respectively.

#### 4.1.3 Performance comparison

In the following, we compare our IFQ-AlexNet with "Lin *et al.* [9]" which is the state-of-the-art AlexNet-based fixed-point network. Lin *et al.* [9] employ a $\gamma$ ($\gamma \geq 9$) as the number of bits for representing each data of the first layer and then introduce an optimal setting on the number of bits for other layers with respect to $\gamma$ (see Table 3). It is worth to point out that "Lin *et al.* [9]" is converted from an AlexNet-like network which posses $\sim 2\times$ savings on the number of parameters compared to our IFQ-AlexNet (21.5 million vs. 58.3 million[2]).

Table 3 compares the number of bits that are employed to represent every fixed-point data of each layer of "Lin *et al.* [9]" and our IFQ-AlexNet. As shown in the table, for $conv2 \sim conv5$ layers, IFQ-AlexNet employs 1-bit for representing their weights which is remarkably lower than "Lin *et al.* [9]". Most importantly, for $FC6$ and $FC7$ layers which are parameter intensive and thus dominate the model size, we consistently employ 1-bit weights. Thus, our IFQ-Net gives $6\times$ savings (1-bit vs. 6-bits). On the other hand, regarding to the feature maps, our IFQ-AlexNet networks also generally use lower bits than their competitors (the same bits may happen on $conv2$ and $conv4$ layers

[2]To be consistent with the reference paper [9], the parameters in $FC_8$ are not included.

only if $k = 4$ and $\gamma = 9$).

Table 3. Comparison on the number of bits employed for each layer of AlexNet-based fixed-point networks.

| | Lin *et al*. [9] ($\gamma \geq 9$) | | IFQ-AlexNet | |
|---|---|---|---|---|
| | Weights | Feature maps | Weights | Feature maps |
| $conv1$ | $\gamma$ | $\gamma$ | 9 | 8 |
| $conv2$ | $\gamma$-5 | $\gamma$-5 | 1 | k |
| $conv3$ | $\gamma$-4 | $\gamma$-4 | 1 | k |
| $conv4$ | $\gamma$-5 | $\gamma$-5 | 1 | k |
| $conv5$ | $\gamma$-4 | $\gamma$-4 | 1 | k |
| $FC6$ | 6 | 6 | 1 | k |
| $FC7$ | 6 | 6 | 1 | k |

For "Lin *et al*. [9]" networks, different $\gamma$ give different preciseness of its fixed-point data. We directly borrow the experimental results from the paper setting $\gamma$ to 9 and 10. Table 4 illustrates the memory usage of the weights and feature maps of the fixed-point networks in terms of millions of bits (Mbits). As shown in the table, regarding to the model size of the compared fixed-point networks, our IFQ-AlexNet networks ($k$ = 2 or 4) give 2.16× savings(58.8 Mbits vs. 127.3 Mbits) over "Lin *et al*. [9] ($\gamma$ = 9)".

Table 4. Model size (Mbits), inference memory for feature maps (Mbits) and performance comparison of fixed-point networks.

| | Lin *et al*. [9] | | IFQ-AlexNet | |
|---|---|---|---|---|
| | $\gamma = 9$ | $\gamma = 10$ | $k = 2$ | $k = 4$ |
| Model size | 127.3 | 128.5 | **58.8** | **58.8** |
| Inference memory (feature maps) | 10.8 | 12.0 | **0.6** | 1.1 |
| Top-5 accuracy | 0.74 | **0.78** | 0.76 | **0.78** |

To evaluate the memory usage of feature maps during inference time, we measure the maximum memory that consumed by one single layer, which is $Conv1$ in the case of AlexNet. Such evaluation makes more sense than evaluating the summation of all layers because the feature maps from other un-connected layers is not required thus can be discarded during inference time. Comparing with "Lin *et al*. [9]", our IFQ-AlexNet networks output 4× smaller feature maps for $Conv1$ layer ($55 \times 55$ vs. $112 \times 112$). Furthermore, our IFQ-AlexNet employs less bits to represent each element of the feature maps of $Conv1$ layer($k = 2\,or\,3\,or\,4$ vs. $\gamma = 9\,or\,10$). Consequently, when comparing IFQ-AlexNet ($k = 2$) with "Lin *et al*. [9]", our method gives 18× savings on inference memory for feature maps.

Furthermore, we follow the reference paper [9] and use Top-5 accuracy to evaluate the performance of the AlexNet-based fixed-point networks. Comparing with "Lin *et al*. [9] ($\gamma = 9$)", IFQ-AlexNet ($k = 2$) gives 2% improvement accuracy with significant savings on model size and feature maps memory as well. Moreover, comparing the "Lin *et al*. [9] ($\gamma = 10$)" and IFQ-AlexNet ($k = 4$ ) networks which have higher precision, our method also achieves 2.18× and 10.9× savings on model size and feature maps respectively without performance drop.

### 4.2. IFQ-Tinier-YOLO face detector

Face detection has various applications in real life and thus emerges many algorithms such as Faster R-CNN [16], SSD [11], Mask R-CNN [5] and YOLOv2 [15]. In this section, we aim to apply our IFQ-Net to face detection task. For the embedded devices, the simple architecture of a deployed network would give great benefit on the hardware design. Consequently, we make use of YOLOv2 detection algorithm as the framework for our face detector.

We initially employ the Tiny-YOLO [15] network due to its compact size. Furthermore, we design a more compact network Tinier-YOLO based on Tiny-YOLO by: 1) only using half the number of filters in each convolution layer; 2) replacing the $3 \times 3$ filter into $1 \times 1$ for the third to last convolution layer; 3)binarizing the weights of all convolution layers by HWGQ. The above three strategies give 4×, 2× and 32× savings respectively and overall 256× savings on model size resulting in a 246k Bytes face detector.

Table 5. Comparison on the model size (MB) of the trained face detectors and their detection rate on FDDB dataset [7].

| | Tiny-YOLO | IFQ-Tiny-YOLO ($k$=2) | Tinier-YOLO | IFQ-Tinier-YOLO ($k$=2) |
|---|---|---|---|---|
| model size (MB) | 63.00 | 1.97 | 7.89 | **0.25** |
| detection rate | **0.92** | 0.89 | 0.90 | 0.84 |

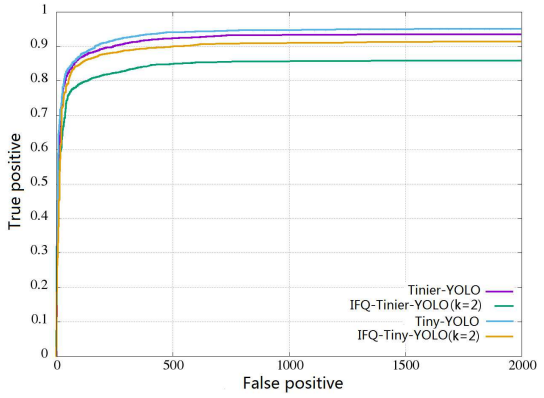



Figure 5. Performance of the face detectors on FDDB dataset [7].

We use the training set of Wider Face [18] and Darknet deep learning framework [14] to train the baseline Tiny-YOLO and our Tinier-YOLO networks. Furthermore, to obtain their quantized fixed-point counterparts IFQ-Tiny-YOLO and IFQ-Tinier-YOLO, we first train the quantiza-

Figure 6. Qualitative performance of the proposed IFQ-Tiner-YOLO ($k = 2$) face detector on Wider Face dataset [18].

tion network with our HWGQ implementation on Darknet (k = 2) and then convert each of their substructure into fixed-point. Each network is trained for 100k iterations with batch size 128. The learning rate is initially set to 0.01 and down scaled by 0.1 at $30k$th, $60k$th and $90k$th iteration. Besides, we also inherit the multi-scale training strategy from YOLOv2.

We compare the trained face detectors on FDDB dataset [7] which contains 5,171 faces in 2,845 testing images. To evaluate the performance of the face detector, we employ detection rate when false positive rate is 0.1 (1 false positive in 10 test images). It corresponds to the true positive rates (y-axis) when the false positive (x-axis) equals to $\lfloor 0.1 \times 2,845 \rfloor = 284$ in Figure 5. Such evaluation is more meaningful in real applications when low false positive rate is desired. As illustrated in Table 5, comparing with Tiny-YOLO, IFQ-Tiny-YOLO achieves 32× savings on model size with 3% drop on detection rate (0.89 vs. 0.92). Furthermore, the proposed IFQ-Tinier-YOLO face detector gives a further 8× savings over IFQ-Tiny-YOLO with 5% performance drop. We think its performance is promising in the sense of its extremely compact model size and quite satisfactory detection rate. More importantly, the proposed IFQ-Tinier-YOLO face detector is a fixed-point network which can be easily implemented on embedded devices. The ROC curves of the compared face detectors are illustrated in Figure 5.

Moreover, the proposed IFQ-Tinier-YOLO is also effective on detecting small faces. We test it on Wider Face validation images and show its qualitative results. As shown in Figure 6, our IFQ-Tinier-YOLO also gives nice detection on small faces in various challenging scenarios such as make-up, out of focus, low-illumination, paintings etc.

## 5. Conclusions

In this paper, we presented a novel fixed-point network, IFQ-Net, for embedded vision. It divides a quantization network into substructures and then converts each substructure into fixed-point in either separated or the proposed integrated manner. Especially for the quantized substructures, which commonly appear in quantization networks, the integrated conversion method removes on-device batch normalization computation, requires no scaling-up effect ($m = 0$) yet most importantly does not cause performance drop. We compared our IFQ-Net with the state-of-the-art fixed-point network indicating that our method gives much more savings on model size and feature map memory with similar (or higher in some case) accuracy on ImageNet.

Furthermore, we also designed a fixed-point face detector IFQ-Tinier-YOLO. Comparing with the Tiny-YOLO detector, our model shows its great benefits on embedded devices in the sense of extremely compact model size (246k Bytes), purely fixed-point data operations and quite satisfactory detection rate.

## References


[1] Z. Cai, X. He, J. Sun, and N. Vasconcelos. Deep learning with low precision by half-wave gaussian quantization. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, pages 5406–5414, 2017. 2, 5

[2] M. Courbariaux, Y. Bengio, and J. David. BinaryConnect: Training deep neural networks with binary weights during propagations. In *Annual Conference on Neural Information Processing Systems, NIPS 2015*, pages 3123–3131, 2015. 1

[3] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan. Deep learning with limited numerical precision. In *IEEE International Conference on Machine Learning, ICML 2015*, pages 1737–1746, 2015. 1

[4] P. Gysel, M. Motamedi, and S. Ghiasi. Hardware-oriented approximation of convolutional neural networks. *CoRR*, abs/1604.03168, 2016. 1

[5] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick. Mask R-CNN. In *IEEE International Conference on Computer Vision, ICCV 2017*, pages 2980–2988, 2017. 7

[6] M. Horowitz. Computing's energy problem (and what we can do about it). In *IEEE International Solid-state Circuits Conference Digest of Techinical Papers*, pages 10–14, 2014. 1

[7] 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. 7, 8

[8] D. D. Lin and S. S. Talathi. Overcoming challenges in fixed point training of deep convolutional networks. *CoRR*, abs/1607.02241, 2016. 1

[9] D. D. Lin, S. S. Talathi, and V. S. Annapureddy. Fixed point quantization of deep convolutional networks. In *IEEE International Conference on Machine Learning*, pages 2849–2858, 2016. 1, 5, 6, 7

[10] Z. Lin, M. Courbariaux, R. Memisevic, and Y. Bengio. Neural networks with few multiplications. *CoRR*, abs/1510.03009, 2015. 1

[11] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg. SSD: single shot multibox detector. In *IEEE European Conference on Computer Vision, ECCV 2016*, pages 21–37, 2016. 7

[12] S. Migacz. 8-bit inference with TensorRT. http://on-demand.gputechconf.com/gtc/2017/presentation/s7310-8-bit-inference-with-tensorrt.pdf, May 2017. 1

[13] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *IEEE European Conference on Computer Vision, ECCV 2016*, pages 525–542, 2016. 2, 3

[14] J. Redmon. Darknet: Open source neural networks in C. http://pjreddie.com/darknet/, 2013–2016. 7

[15] J. Redmon and A. Farhadi. YOLO9000: better, faster, stronger. In *IEEE International Conference on Computer Vision and Pattern Recognition, CVPR 2017*, pages 6517–6525, 2017. 7

[16] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017. 7

[17] F. Shafiq, T. Yamada, A. T. Vilchez, and S. Dasgupta. Automated flow for compressing convolution neural networks for efficient edge-computation with FPGA. *ArXiv e-prints*, 2017. 5

[18] 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*, 2016. 7, 8