高效神经网络的量化和训练仅整数算术推理

Benoit Jacob Skirmantas Kligys Bo Chen Menglong Zhu Andrew Howard
马修唐 Hartwig Adam Dmitry Kalenichenko
{benoitjacob,skligys,bochen,menglong,
mttang,howarda,hadam,dkalenichenko}@google.com谷歌公司

抽象的

智能移动设备的日益普及和基于深度学习的模型的高昂计算成本需要高效准确的设备推理方案。我们提出了一种量化方案,允许使用纯整数算法进行推理,这比在常用的纯整数硬件上进行浮点推理更有效。我们还共同设计了一个训练程序,以保持量化后端到端模型的准确性。因此,所提出的量化方案改善了准确性和设备延迟之间的权衡。即使在以运行时效率着称的模型系列 MobileNets 上,这些改进也很重要,并在流行 CPU 上的 ImageNet 分类和 COCO 检测中得到了证明。

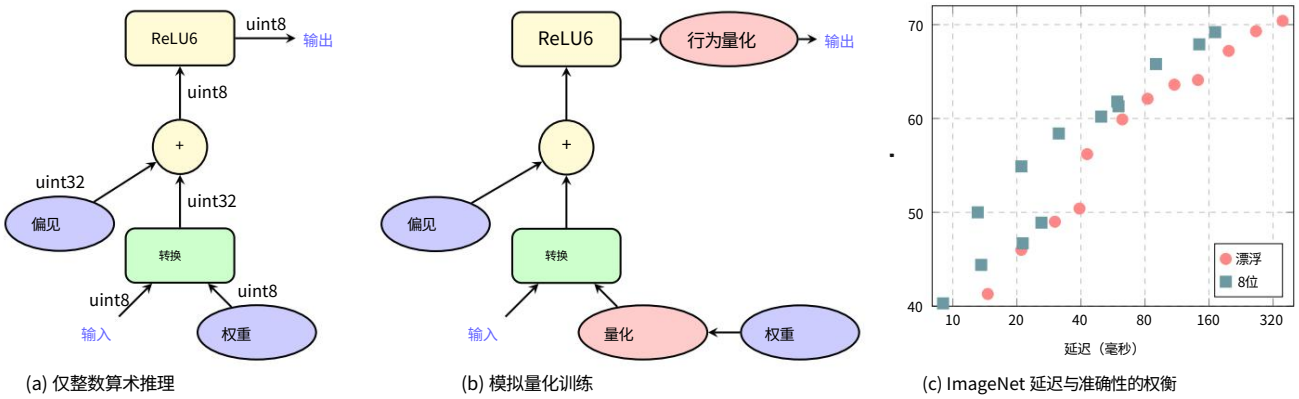
一、简介

当前最先进的卷积神经网络 (CNN) 不太适合在移动设备上使用。自从 AlexNet [20] 出现以来,现代 CNN 主要根据分类/检测精度进行评估。因此,网络架构的发展没有考虑模型的复杂性和计算效率。另一方面,要在智能手机、AR/VR 设备(HoloLens、Daydream)和无人机等移动平台上成功部署 CNN,需要较小的模型尺寸来适应有限的设备内存,并需要低延迟来维持用户参与度。这导致了一个新兴的研究领域,该领域专注于以最小的精度损失减少 CNN 的模型大小和推理时间。

该领域的方法大致分为两类。第一类,例如 MobileNet [10]、SqueezeNet [16]、 ShuffleNet [32]和 DenseNet [11],设计了利用计算/内存高效操作的新颖网络架构。第二类拳将 CNN 的权重和/或激活从 32 位浮点数转换为更低的位深度表示。这种方法被三元权重网络 (TWN [22])、二元神经网络 (BNN [14])、XNOR-net [27]等方法所接受[8、21、26、33、34、35] ,是我们调查的重点。尽管存在大量的舞蹈,但当前的量化方法在延迟与准确性之间的权衡方面存在两个方面的不足。

首先,尚未在合理的基线架构上评估先前的方法。最常见的基线架构,AlexNet [20]、 VGG [28]和 GoogleNet [29],都通过设计进行了过度参数化,以提取边际精度改进。因此,很容易对这些架构进行相当大的压缩,将对这些架构的量化实验最多减少到概念验证。相反,一个更有意义的挑战将是量化已经在延迟与准确性之间有效权衡的模型架构,例如 MobileNets。

其次,许多量化方法无法在真实硬件上提供可验证的效率改进。仅量化权重([2, 4, 8, 33]) 的方法主要关注设备上的存储,较少关注计算效率。值得注意的例外是二元、三元和位移位网络[14、22、27] 。后一种方法采用 0 或 2 的幂的权重,这允许通过移位实现乘法。然而,虽然位移位在定制硬件中可能很有效,但它们对现有的带有乘加指令的硬件几乎没有好处,如果正确使用(即流水线) ,并不比单独的加法更昂贵。此外,只有当操作数很宽时,乘法才会很昂贵,并且一旦权重和激活都被量化,避免乘法的需求就会随着位深度的增加而减少。值得注意的是,这些方法很少提供设备上测量来验证承诺的时序改进。更多运行时友好的方法将权重和激活都量化为 1 位表示



(a) 仅整数算术推理 (b) 模拟量化训练 (c) ImageNet 延迟与准确性的权衡

图 1.1:仅整数算术量化。 a)卷积层的纯算术推断。根据等式 1,输入和输出表示为 8 位整数。卷积涉及 8 位整数操作数和 32 位整数累加器。

偏置加法仅涉及 32 位整数（第2.4 节）。 ReLU6 非线性仅涉及 8 位整数运算。 b)用卷积层的模拟量化进行训练。所有变量和计算均使用 32 位浮点运算进行。

权重量化（“wt quant”）和激活量化（“act quant”）节点被注入到计算图中,以模拟变量量化的效果（第 3 节）。结果图近似于面板 a) 中的仅整数算术计算图，同时可以使用浮点模型的常规优化算法进行训练。 c)我们的量化方案受益于普通 CPU 中的快速整数运算电路,以提供改进的延迟与精度权衡（第4 节）。该图使用 Qualcomm Snapdragon 835 LITTLE 内核将整数量化的 MobileNets [10]与 ImageNet [3]上的浮点基线进行了比较。

[14, 27, 34]。通过这些方法,乘法和加法都可以通过高效的位移位和位计数操作来实现,这在自定义 GPU 内核 (BNN [14]) 中得到了展示。然而,1 位量化为 10 会导致性能大幅下降,并且可能对模型表示过于严格。

在本文中,我们通过改进 MobileNets 在常见移动硬件上的延迟与准确性权衡来解决上述问题。我们的具体贡献是：

- 我们提供了一个量化方案（第2.1 节） ,将权重和激活都量化为 8 位整数,仅将几个参数（偏置向量)量化为 32 位整数。
- 我们提供了一个量化推理框架,可以在诸如 Qualcomm Hexagon（第2.2、2.3节)等纯整数运算硬件上有效地实现,并且我们描述了一个在 ARM NEON 上高效、准确的实现（附录B）。
- 我们提供了一个与我们的量化推理共同设计的量化训练框架（第 3 节） ,以最大限度地减少真实模型量化的准确性损失。
- 我们将我们的框架应用于基于 MobileNets 的高效分类和检测系统，并在流行的 ARM CPU 上提供基准测试结果（第4 节） ,这些结果显示了最先进的 MobileNet 架构在延迟与准确性权衡方面的显着改进，证明在 ImageNet 分类[3]、COCO 对象检测[23]和其他任务中。

我们的工作从 [7] 和[31]中汲取灵感,前者利用低精度定点算法来加速 CNN 的训练速度,后者利用8 位定点算法来加速 x86 CPU 上的推理。我们的量化方案侧重于提高移动 CPU 上的推理速度与精度之间的权衡。

2.量化推理

2.1.量化方案

在本节中,我们描述了我们的通用量化方案 12,即值的位表示（在下面表示为q ,表示“量化值”）与它们作为数学实数的解释（在下面表示为r ,表示“实值”）之间的对应关系”）。我们的量化方案是在推理期间使用纯整数算法和在训练期间使用浮点算法来实现的,两种实现方式彼此保持高度对应。我们通过首先为我们的量化方案提供数学上严格的定义，并将该方案分别用于整数算术推理和浮点训练来实现这一点。

1此处描述的量化方案是 Tensor Flow Lite [5]中采用的方案,我们将参考其代码的特定部分来说明下面讨论的方面。

2我们之前在gemmlowp [18] 的文档中描述了这种量化方案。该页面作为本节中开发的某些主题的替代处理方式及其自包含的示例代码可能仍然有用。

我们的量化方案的一个基本要求是,它允许仅使用对量化值的整数算术运算来有效地实现所有算术（我们避免需要查找表的实现,因为与 SIMD 硬件上的纯算术相比,这些实现往往表现不佳）。这相当于要求量化方案是整数q到实数r 的仿射映射,即具有以下形式

$$r = S(q - Z) \tag{1}$$

对于一些常数S和Z。等式(1)是我们的量化方案,常数S和Z是我们的量化参数。我们的量化方案对每个激活数组和每个权重数组中的所有值使用一组量化参数;单独的数组使用单独的量化参数。

对于8位量化， q被量化为8位整数（对于B位量化， q被量化为B位整数）。
一些数组,通常是偏置向量,被量化为 32 位整数,请参见第2.4 节。

常数S （表示“比例”）是任意正实数。它通常在软件中表示为浮点数,如实数值r。 2.2节描述了避免在推理工作负载中表示此类浮点量的方法。

常量Z （表示“零点”）与量化值q 属于同一类型,实际上是对应于实数值 0 的量化值q 。这使我们自动满足实数值r的要求= 0可以用量化值精确表示。此要求的动机是神经网络运算符的有效实现通常需要在边界周围对数组进行零填充。

到目前为止,我们的讨论总结在以下量化缓冲区数据结构 3 中,神经网络中的每个激活数组和权重数组都存在此类缓冲区实例,其中[illegible]

```
template<typename QType> // 例如 QType=uint8 struct QuantizedBuffer
{ vector<QType> q;浮动 S； Q型Z； };
                                    // 量化值 // 比例 // 零点
```

2.2.仅整数算术矩阵乘法

我们现在转向如何仅使用整数运算来执行推理的问题,即如何使用等式（1）将实数计算转换为量化值

3TensorFlow Lite [5] Converter中实际的数据结构是这个头文件中的QuantizationParams和Array。正如我们在下一小节中讨论的那样,这个仍然包含浮点数的数据结构不会出现在实际量化的设备推理代码中。

计算,以及后者如何设计为仅涉及整数运算,即使比例值S不是整数。

考虑两个N × N实数平方矩阵r1和r2的乘法,它们的乘积表示为r3 = r1r2。我们将这些(i,j)矩阵rα (α = 1、2或3)中的每一个的条目表示为r for 1 i, j N,并将它们量化 (i,j) 的量化参数表示为(Sα， Zα）。我们用q表示量化条目等式（1）然后变为：

α

α。

$$\alpha^{(i,j)} = S\alpha(q_{\alpha}(i,j) - Z\alpha)。 \tag{2}$$

根据矩阵乘法的定义,我们有

$$S3(q_{\text{我知道}} - Z3) = \sum_{j=1}^{\text{否}} S1(q^{(i,j)}_{} - Z1)S2(q_{} \text{-Z2}), \tag{3}$$

可以改写为

$$3^{\text{我知道}} = Z3 + M\, q \sum_{j=1}^{\text{否}} {}^{(i,j)}_{} {}^{(j,k)}_{} (q - Z1)(q_{} - Z2), \tag{4}$$

其中乘数M定义为

$$\text{米} := \frac{S1S2}{S3}. \tag{5}$$

在等式（4)中,唯一的非整数是乘数M。
作为一个仅取决于量化尺度S1、 S2、 S3 的常数,它可以离线计算。我们凭经验发现它总是在区间(0, 1) 中,因此可以用规范化形式表示

$$M = 2^{-n}M0 \tag{6}$$

其中M0在区间[0.5, 1)内， n为非负整数。归一化乘数M0现在很适合表示为定点乘数（例如 int16 或 int32,具体取决于硬件能力）。例如,如果使用int32,表示M0的整数就是最接近2 31M0的int32值。由于M0为 0.5,此值始终为 ,因此始终具有至少 30 位的相对精度。因此,与M0的乘法可以作为定点乘法 4 来实现。同时,可以用一个高效的位来实现multi

最少2 30

乘以2移位,尽管需要具有正确的舍入到最近的行为,这是我们在附录B中返回的问题。

2.3.有效处理零点

为了有效地执行等式（4)的评估而不必执行2N3减法和

4 本节讨论的计算在 Tensor Flow Lite [5]参考代码中实现对于全连接层。

无需将乘法的操作数扩展为 16 位整数,我们首先注意到,通过在等式 (4) 中分配乘法,我们可以将其重写为

(我知道) = Z3 + M q 3 (k) NZ1Z2 − 否 −Z2a − (我) 2 (i,j) (j,k) q Z1a j=1 (7)

在哪里

−(k) := 否 2 (j,k) q , a −(i) := 否 1 (i,j) q . j=1 j=1 (8)

每个a或a(k) 2 1 只需要N次加法来计算,所以它们总共只需要2N2 次加法。 (7)的评估的其余成本几乎全部集中在核心整数矩阵乘法累加

否 q (i,j) (j,k) q 1 j=1 (9)

需要2N3 次算术运算;实际上, (7)中涉及的所有其他内容都是O(N2),O中有一个小常数。

因此, a和a ̄的形式(7)的扩展和因式分解(k) (i)的计算能够实现任意零点的低开销处理,同核心整数矩阵乘法累加(9)。

2.4.典型融合层的实现

我们继续第2.3节的讨论,但现在明确定义所有涉及的量的数据类型,并修改量化矩阵乘法(7)以将偏差加法和激活函数评估直接合并到其中。将整个层融合到单个操作中不仅仅是一种优化。由于我们必须在推理代码中重现训练中使用的相同算法,推理代码中融合运算符的粒度(采用 8 位量化输入并产生 8 位量化输出)必须与"假量化"的位置相匹配"训练图中的运算符(第3 节)。

对于我们在 ARM 和 x86 CPU 架构上的实现,我们使用 gemmlowp 库[18],其 GemmWithOutputPipeline 入口点提供对我们现在描述的融合操作的支持5 .

我们将q1矩阵作为权重,将q2矩阵作为激活。权重和激活都是 uint8 类型(我们可以等效地选择 int8,并适当修改零点)。累加 uint8 值的乘积需要 32 位累加器,我们为累加器选择有符号类型的原因很快就会清楚。 (9)中的总和因此具有以下形式:

$$\text{int32 += uint8 * uint8。} \tag{10}$$

为了使量化的偏置加法是将 int32 偏置加到这个 int32 累加器中,偏置向量被量化为: 它使用 int32 作为其量化数据类型;它使用 0 作为其量化零点Zbias;其量化尺度Sbias与累加器的量化尺度相同,是权重尺度和输入激活尺度的乘积。在 2.3节的符号中,

$$S_{bias} = S_1 S_2, \quad Z_{bias} = 0。 \tag{11}$$

尽管偏置向量被量化为 32 位值,但它们仅占神经网络参数的一小部分。此外,对偏置向量使用更高的精度满足了实际需要:由于每个偏置向量条目都被添加到许多输出激活中,偏置向量中的任何量化误差都倾向于充当整体偏置(即具有非零的误差项均值),必须避免这种情况,以保持良好的端到端神经网络准确性6 。

有了 int32 累加器的最终值,剩下的主要三件事要做:缩小到 8 位输出激活使用的最终比例,向下转换为 uint8 并应用激活函数产生最终的 8 位输出激活。

缩小对应于乘以等式(7)中的乘数M。如第 2.2 节所述,它通过归一化乘数M0和舍入位移位实现定点乘法。之后,我们对 uint8 执行饱和和转换,饱和到范围[0, 255]。

我们专注于仅仅是钳位的激活函数,例如 ReLU、ReLU6。附录A.1中讨论了数学函数,我们目前没有将它们融合到这样的层中。因此,我们的融合激活函数唯一需要做的就是在存储最终的 uint8 输出激活之前将 uint8 值进一步限制在[0, 255]的某个子区间内。在实践中,量化训练过程(第3 节)倾向于学习利用整个输出 uint8 [0, 255]区间,这样激活函数就不再做任何事情,其效果被包含在[0, 255]隐含在对 uint8 的饱和和转换中。

5 本节中的讨论在 TensorFlow Lite [5]中实现,例如卷积运算符(参考代码是独立的,优化的代码调用 gemmlowp [18])。

6 这里讨论的偏置向量的量化在这里实现在 TensorFlow Lite [5]转换器中。

3. 模拟量化训练

训练量化网络的一种常见方法是在浮点数中训练,然后量化生成的权重（有时需要额外的量化后训练以进行微调）。我们发现这种方法对于具有相当大的表示能力的大型模型非常有效,但对于小型模型会导致准确率显着下降。简单训练后量化的常见失败模式包括:1)不同输出通道的权重范围差异较大（超过100×）（第2节要求同一层的所有通道都量化到相同的分辨率,这导致权重在具有较小范围的通道中具有更高的相对误差)和 2)异常权重值,使所有剩余权重在量化后不太精确。

我们提出了一种在前向训练过程中模拟量化效果的方法。反向传播仍然像往常一样发生,所有的权重和偏差都以浮点形式存储,这样它们就可以很容易地被少量微调。然而,前向传播通道通过在浮点运算中实现我们在第 2 节中介绍的量化方案的舍入行为来模拟推理引擎中发生的量化推理:

- 权重在与输入卷积之前被量化。如果批量归一化（见[17])用于该层,则批量归一化参数在量化前“折叠到”权重中,见3.2 节。
- 激活在推理过程中的点被量化,例如在激活函数应用于卷积或完全连接层的输出之后,或者在旁路连接将多个层的输出添加或连接在一起之后,例如在 ResNets 中。

对于每一层,量化由量化级别的数量和钳位范围参数化,并通过逐点应用定义如下的量化函数q来执行:

$$\text{clamp}(r; a, b) := \min(\max(x, a), b)$$

$$s(a, b, n) := \frac{b - a}{n - 1}$$

$$q(r; a, b, n) := \frac{\text{clamp}(r; a, b) - }{as(a, b, n)} \quad s(a, b, n) + \text{一个},$$

(12)

其中r是要量化的实数值， [a; b]是量化范围， n是量化级数， ·表示四舍五入到最接近的整数。在我们的实验中， n对于所有层都是固定的,例如,对于 8 位量化， $n = 28 = 256$ 。

3.1.学习量化范围

权重的量化范围不同
量化与激活量化:

- 对于权重,基本思想是简单地设置a := min w, b := max w。我们对此应用了一个小调整,以便权重一旦量化为 int8 值,仅在[－127, 127]范围内并且永远不会取值－128,因为这提供了一个实质性的优化机会（有关更多详细信息,请参见附录B)。
- 对于激活,范围取决于网络的输入。为了估计范围,我们收集[a; b]在训练期间在激活上看到的范围,然后通过指数移动平均线 (EMA) 将它们聚合,平滑参数接近 1,以便观察到的范围在数千个训练步骤中平滑。

鉴于当范围快速变化时 EMA 更新激活范围的显着延迟,我们发现在训练开始时完全禁用激活量化是有用的（例如,5 万到 200 万步）。这允许网络进入更稳定的状态,其中激活量化范围不会排除很大一部分值。

在这两种情况下,边界[a; b]被微调,以便值0.0在量化后可以精确表示为整数z(a, b, n) 。因此,学习的量化参数映射到等式1中的尺度S和零点Z： $S = s(a, b, n), Z = z(a, b, n)$

(13)

下面我们描述了模拟量化,假设神经网络的计算被捕获为 TensorFlow 图[1]。算法1 中描述了一个典型的工作流程。通过融合优化推理图

算法 1量化图训练和推理

1:创建浮点模型的训练图。
2:根据等式12,在推理过程中张量将被向下转换为更少位的位置插入伪量化 TensorFlow 操作。
3:在模拟量化模式下训练直到收敛。
4:创建和优化在低位推理引擎中运行的推理图。
5:使用量化推理图运行推理。

删除操作超出了本文的范围。图修改的源代码（插入伪量化操作,创建和优化推理图)和低位推理引擎已经在 [19] 中开源了 TensorFlow 贡献。

图1.1a和 b 说明了简单卷积层量化前后的 TensorFlow 图。在图 C.4 中可以找到图C.3中带有旁路连接的更复杂卷积的图示。

请注意,偏差未被量化,因为它们在推理过程中表示为 32 位整数,与 8 位权重和激活相比具有更高的范围和精度。此外,用于偏差的量化参数是从权重和激活的量化参数中推断出来的。参见第2.4 节。

说明[19]使用的典型 TensorFlow 代码如下:

```
来自tf.contrib.quantize \
        将quantize_graph 导入为 qg

g = tf.Graph()和
g.as_default():
    输出 =              ...
    total_loss = 优化器 =    ...
    train_tensor =          ...
                                ...
如果is_training:
    quantized_graph = \
            qg.create_training_graph(g)
别的:
    量化图 = \ qg.create_eval_graph(g)

# 训练或评估 quantized_graph。
```

3.2.批量归一化折叠

对于使用批量归一化的模型（见[17]），还有额外的复杂性:训练图包含批量归一化作为一个单独的操作块,而推理图将批量归一化参数“折叠”到卷积层或全连接层的权重中和偏见,为了效率。为了准确模拟量化效果,我们需要模拟这种折叠,在经过batch normalization参数缩放后量化权重。我们通过以下方式做到这一点:

$$\text{折叠} := \frac{\gamma w}{\text{均线移动平均线}(\sigma_{\vec{B}}) + \varepsilon}. \qquad (14)$$

这里γ是批量归一化的尺度参数， EMA(跨批次卷积结果的σ,而ε只是数值稳定性的B)是方差的移动平均估计

折叠后,批归一化卷积层简化为图1.1a中描绘的简单卷积层,其中折叠权重为wfold且相应的折叠偏差为 ing。因此,图1.1b中的相同配方适用。batch-normalized卷积层的训练图见附录（图C.5） ,对应的推理图（图C.6） ， batch-norm折叠后的训练图（图C.7)和训练图折叠和量化后的图形（图C.8）。

ResNet深度	50	100	150
浮点精度 76.4% 78.0% 78.8% 整数量化精度 74.9% 76.6% 76.7%			

表 4.1:ImageNet 上的 ResNet:各种网络深度的浮点与量化网络精度。

方案 BWN TWN INQ FGQ 我们的					
重量位	1个	2个	5个	2个	8个
激活位 float32 float32 float32				8个	8个
准确性	68.7% 72.5% 74.8% 70.8% 74.9%				

表 4.2:ImageNet 上的 ResNet:各种量化方案下的精度,包括二进制权重网络（BWN [21、15]）、三元权重网络（TWN [21、22]）、增量网络量化（INQ [33])和细粒度量化（FGQ [26]）

4. 实验

我们进行了两组实验,一组展示了量化训练的有效性（第4.1 节），另一组展示了通用硬件上量化模型的改进延迟与准确性权衡（第 4.2 节）。在被基准测试的神经网络上,推理工作负载中对性能最关键的部分是矩阵乘法 (GEMM)。 8 位和 32 位浮点 GEMM 推理代码使用 gemmlowp 库[18]进行 8 位量化推理,使用 Eigen 库[6]进行 32 位浮点推理。

4.1.大型网络的量化训练

我们将量化训练应用于ImageNet 数据集上的ResNets [9]和 InceptionV3 [30] 。这些流行的网络计算过于密集,无法部署在移动设备上,但出于比较目的包含在内。

培训协议在附录D.1和D.2中讨论。

4.1.1 ResNets

我们在表 4.1 中比较了各种深度的浮点与整数量化 ResNet 。仅整数量化网络的精度在其浮点对应网络的2%以内。

我们还在表 4.2 中列出了不同量化方案下的 ResNet50 精度。正如预期的那样,仅整数量化优于 FGQ [26],后者使用 2 位进行权重量化。INQ [33] （5 位权重浮点激活)实现了与我们类似的精度,但我们提供了额外的运行时改进（参见第4.2 节）。

行为。	类型	准确性	召回 5
		平均标准。开发者均值标准差。	
ReLU6 浮点数		78.4% 0.1%	94.1% 0.1%
8 位		75.4% 0.1%	92.5% 0.1%
7 位		75.0% 0.3%	92.4% 0.2%
ReLU 浮动		78.3% 0.1%	94.2% 0.1%
8 位		74.2% 0.2%	92.2% 0.1%
7 位		73.7% 0.3%	92.0% 0.1%

表 4.3:ImageNet 上的 Inception v3:精度和召回率 5 浮点模型和量化模型的比较。

4.1.2 ImageNet 上的 Inception v3

我们比较了分别量化为 8 位和 7 位的 Inception v3 模型。通过将等式12中的量化级别数设置为n = 27来获得 7 位量化。我们还通过比较具有两个激活非线性 ReLU6 和 ReLU 的网络来探测激活量化的灵敏度。培训协议在附录D.2 中。

表4.3显示 7 位量化训练产生的模型精度接近 8 位量化训练的精度,并且使用 ReLU6 的量化模型精度下降较小。后者可以通过注意到 ReLU6 引入区间[0, 6]作为激活的自然范围来解释,而 ReLU 允许激活从可能更大的区间中获取值,在不同的通道中具有不同的范围。固定范围内的值更容易进行高精度量化。

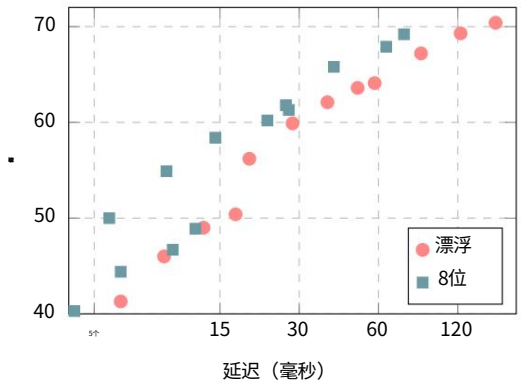


图 4.1:Qualcomm Snapdragon 835 大核上的 ImageNet 分类器:浮点和仅整数 MobileNet 的延迟与精度权衡。

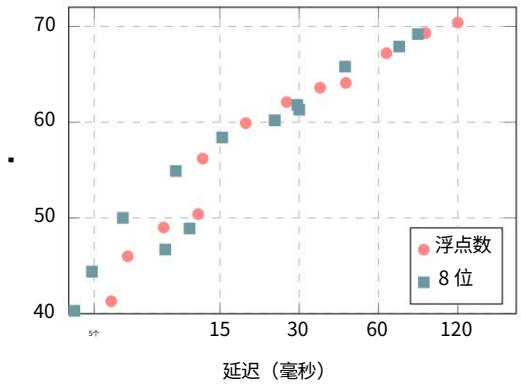


图 4.2:Qualcomm Snapdragon 821 上的 ImageNet 分类器:浮点和整数 MobileNet 的延迟与精度权衡。

4.2. MobileNets 的量化

MobileNet 是一系列架构,可在设备延迟和 ImageNet 分类准确性之间实现最先进的权衡。在本节中,我们将演示纯整数量化如何进一步改进通用硬件的权衡。

4.2.1 图像网

我们在三种类型的 Qualcomm 内核上使用不同的深度倍增器 (DM) 和 ImageNet 分辨率对 MobileNet 架构进行了基准测试,这代表了三种不同的微架构:1) Snapdragon 835 LITTLE 内核,（图 1.1c）,一种强大的Google Pixel 2 中的高效处理器； 2)骁龙835大核（图4.1）,谷歌Pixel 2采用的高性能核心；和 3)

骁龙821大核（图4.2）,谷歌Pixel 1采用的高性能核心。

在相同的运行情况下,仅整数量化的 MobileNets 比浮点 MobileNets 获得更高的精度

时间预算。对于实时 (30 fps) 操作所需的 33 毫秒延迟,Snapdragon 835 LITTLE 内核的精度差距相当大 (10%) 。虽然大多数量化文献都侧重于最小化给定架构的精度损失,但我们提倡更全面的延迟与精度权衡作为更好的衡量标准。

请注意,这种权衡主要取决于硬件中浮点运算与纯整数运算的相对速度。例如,Snapdragon 821 中的浮点计算得到了更好的优化,导致量化模型的延迟减少不太明显。

4.2.2 可可

我们在移动实时对象检测的背景下评估量化,比较 MobileNet SSD [10、25]的量化 8 位和浮点模型在 COCO 数据集[24] 上的性能。我们用可分离卷积替换了 SSD 预测层中的所有常规卷积

DM 类型 mAP LITTLE (ms) big (ms)		
100% 浮点数 22.1 8 位 21.7	778 687	370 272
50% 浮点数 16.7 8 位 16.6	270 146	121 61

表 4.4:浮点和仅整数量化模型的 COCO 数据集的对象检测速度和精度。

延迟（毫秒)是在 Qualcomm Snapdragon 835 大核和小核上测得的。

化（深度后跟1 × 1投影）。这种修改与 MobileNets 的整体设计一致,并使它们的计算效率更高。我们利用开源 TensorFlow 对象检测 API [12]来训练和评估我们的模型。培训协议在附录 D.3 中描述。我们还延迟了50万步的量化（参见第3.1 节）,发现它显着缩短了收敛时间。

表4.4显示了浮点模型和整数量化模型之间的延迟与精度权衡。延迟是使用 Snapdragon 835 内核（大内核和小内核)在单个线程上测量的。量化的训练和推理导致运行时间减少高达50% ,而准确性损失最小（-1.8%相对）。

4.2.3 人脸检测

为了更好地检查较小规模的量化 MobileNet SSD,我们在人脸属性分类数据集（[10] 中使用的基于 Flickr 的数据集)上对人脸检测进行了基准测试。

我们联系了[10]的作者,以按照相同的协议（详见附录D.4)评估我们的量化 MobileNets 在检测和面部属性方面的表现。

如表4.5和4.6所示,量化为 Qualcomm Snap dragon 835 big 或 LITTLE 内核提供了接近2 倍的延迟减少,但代价是平均精度下降约2% 。值得注意的是,量化允许25% 的面部检测器在单个大核上实时运行（1K/28 ≈ 36 fps）,而浮点模型仍然比实时（1K/44 ≈ 23 fps)慢。

我们还检查了多线程对量化模型延迟的影响。表4.6显示了使用4核时1.5到2.2 倍的加速。两个内核的加速比相当,并且对于多线程开销在总计算中所占比例较小的较大模型而言,加速比更高。

4.2.4 人脸属性

图4.3显示了在 Qualcomm Snapdragon 821 上面部属性分类的延迟与准确性的权衡。

DM 型精确召回	
100% 浮点数 68% 8 位 66%	76% 75%
50% 浮点数 65% 8 位 62%	70% 70%
25% 浮点数 56% 8 位 54%	64% 63%

表 4.5:浮点和仅整数量化模型的人脸检测精度。报告的精度/召回率是不同精度/召回率值的平均值,其中groundtruth 和预测窗口之间的IOU x被认为是正确的检测,因为x在{0.5, 0.55, . . ., 0.95}。

DM 型 LITTLE 芯数 2 4	大核心 1 2 4
100% 浮点数 711 – – 337 – – 8 位 372 238 167 154 100 69	
50% 浮点数 233 – – 106 – – 8 位 134 96 74 56 40 30	
25% 浮点数 100 – 8 位 67 52 43 28 22 18	– 44

表 4.6:面部检测:Qualcomm Snapdragon 835 内核上浮点和量化模型的延迟。

由于量化训练几乎不会导致精度下降,因此即使 Qual comm Snapdragon 821 针对浮点运算进行了高度优化（请参见图4.2进行比较）,我们也看到了改进的权衡。

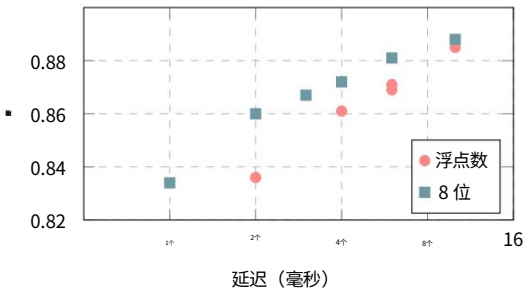


图 4.3:Qualcomm Snap dragon 821 上的人脸属性分类器:浮点数和仅整数 MobileNets 的延迟与精度权衡。

消融研究为了了解性能对量化方案的敏感性,我们进一步评估量化

重量 \ 行为。	8个	7	6个	5个	4个
8	-0.9%	-0.3%	-0.4%	-1.3%	-3.5%
7	-1.3%	-0.5%	-1.2%	-1.0%	-2.6%
6	-1.1%	-1.2%	-1.6%	-1.6%	-3.1%
5	-3.1%	-3.7%	-3.4%	-3.4%	-4.8%
4	-11.4%	-13.6%	-10.8%	-13.1%	-14.0%

表 4.7:人脸属性:整数量化 MobileNets（不同权重和激活位深度）与浮点数相比的相对平均类别精度。

重量 \ 行为。	8个	7	6个	5个	4个
8个	-1.3%	-1.6%	-3.2%	-6.0%	-9.8%
7	-1.8%	-1.2%	-4.6%	-7.0%	-9.9%
6	-2.1%	-4.9%	-2.6%	-7.3%	-9.6%
5	-3.1%	-6.1%	-7.8%	-4.4%	-10.0%
4	-10.6%	-20.8%	-17.9%	-19.0%	-19.5%

表 4.8:面部属性:与浮点相比,量化模型（不同权重和激活位深度)相差 5 年的年龄精度。

使用不同的权重和激活量化位深度进行训练。表4.7和4.8分别显示了二进制属性的平均精度和相对于浮点基线的年龄精度的下降。这些表表明 1) 权重对减少的量化位深度比激活更敏感,2) 8 位和 7 位量化模型的性能与浮点模型相似,以及 3) 当总位深度相等时,它是最好保持权重和激活位深度相同。

5. 讨论

我们提出了一种量化方案,该方案仅依赖于整数算法来近似神经网络中的浮点计算。模拟量化效果的训练有助于将模型精度恢复到与原始模型几乎相同的水平。除了模型大小减少 4 倍之外,推理效率还通过基于 ARM NEON 的实现得到了提高。这一改进促进了普通 ARM CPU 延迟与流行计算机视觉模型准确性之间的最先进权衡。我们的量化方案与高效架构设计之间的协同作用表明,仅整数算术推理可能是推动视觉识别技术进入实时和低端电话市场的关键推动因素。

参考

[1] M. Abadi、A. Agarwal、P. Barham、E. Brevdo、Z. Chen、C. Citro、GS Corrado、A. Davis、J. Dean、M. Devin 等。Tensorflow:异构系统上的大规模机器学习,2015 年。tensorflow 提供的软件。组织, 1, 2015. 5, 11, 12, 13

[2] W. Chen、JT Wilson、S. Tyree、KQ Weinberger 和 Y. Chen。使用散列技巧压缩神经网络。 CoRR, abs/1504.04788, 2015. 1

[3] J. Deng、W. Dong、R. Socher、L.-J。 Li、K. Li 和 L. Fei Fei。 Imagenet:一个大规模的分层图像数据库。在计算机视觉和模式识别中,2009 年。CVPR 2009。IEEE 会议,第 248-255 页。 IEEE, 2009. 2, 11

[4] Y. Gong、L. Liu、M. Yang 和 L. Bourdev。使用矢量量化压缩深度卷积网络。 arXiv 预印本 arXiv:1412.6115, 2014.1 [5] 谷歌。

张量流精简版。 https://www.tensorflow.org/mobile/tflite。 2, 3, 4, 11 [6] G. Guennebaud、B. Jacob 等。 http://eigen.tuxfamily.org。 6个

本征 v3。

[7] S. Gupta、A. Agrawal、K. Gopalakrishnan 和 P. Narayanan。数值精度有限的深度学习。在第 32 届国际机器学习会议论文集 (ICML-15),第 1737–1746 页,2015 年。2 [8] S. Han、H. Mao 和 WJ Dally。深度压缩:通过修剪、训练量化和霍夫曼编码压缩深度神经网络。 CoRR, abs/1510.00149, 2, 2015. 1

[9] K. He, X. Zhang, S. Ren, and J. Sun.用于图像识别的深度残差学习。在 IEEE 计算机视觉和模式识别会议记录中,第 770-778 页,2016 年。6

[10] AG Howard、M. Zhu、B. Chen、D. Kalenichenko、W. Wang、T. Weyand、M. Andreetto 和 H. Adam。 Mobilenets:用于移动视觉应用的高效卷积神经网络。 CoRR, abs/1704.04861, 2017. 1, 2, 7, 8, 12, 13 [11] G. Huang、Z. Liu、L. van der Maaten 和 KQ Wein berger。密集连接的卷积网络。在 IEEE 计算机视觉和模式识别会议 (CVPR) 中,2017 年 7 月。1 [12] J. Huang、V. Rathod、D. Chow、C. Sun 和 M. Zhu。张量

流式物体检测api, 2017. 8

[13] J. Huang、V. Rathod、C. Sun、M. Zhu、A. Korattikara、A. Fathi、I. Fischer、Z. Wojna、Y. Song、S. Guadarrama 等。现代卷积物体检测器的速度/精度权衡。 arXiv 预印本 arXiv:1611.10012, 2016. 12 [14] I. Hubara、M. Courbariaux、D. Soudry、R. El-Yaniv 和 Y. Bengio。二值化神经网络。 In Advances in neural information processing systems, pages 4107–4115, 2016. 1, 2

[15] I. Hubara、M. Courbariaux、D. Soudry、R. El-Yaniv 和 Y. Bengio。量化神经网络:训练神经网络使用低精度权重和激活。 arXiv 预印本 arXiv:1609.07061, 2016. 6

[16] FN Iandola、MW Moskewicz、K. Ashraf、S. Han、WJ
达利和 K. Keutzer。 Squeezenet：Alexnet 级精度，参数减少 50 倍，模型大小为 1mb。 arXiv 预印本 arXiv:1602.07360, 2016. 1 [17] S. Ioffe 和 C. Szegedy。批量归一化：通过减少内部协变量偏移来加速深度网络训练。

在第 32 届国际机器学习国际会议论文集 - 第 37 卷，ICML 15，第 448-456 页。 JMLR.org, 2015. 5, 6 [18] B. Jacob, P. Warden, et al. gemmlowp：一个小型独立的低精度 gemm 库。 2、 https://github.com/google/gemmlowp。 4, 6, 11 [19] S. Kligys, sorflow

S. Sivakumar，量化 等 阿尔。 十
训练 支持。
https://github.com/tensorflow/tensorflow/tree/master/tensorflow/contrib/quantize。 5, 6

[20] A. Krizhevsky、I. Sutskever 和 GE Hinton。使用深度卷积神经网络的 Imagenet 分类。 In Advances in neural information processing systems, pages 1097–1105, 2012. 1 [21] C. Leng、H. Li、S. Zhu 和 R. Jin。极低位神经网络：用 admm 挤出最后一位。 arXiv 预印本 arXiv:1707.09870, 2017. 1, 6

[22] F. Li、B. Zhang 和 B. Liu。三元权重网络。 arXiv 预印本 arXiv:1605.04711, 2016. 1, 6 [23] T.-Y. Lin、M. Maire、S. Belongie、J. Hays、P. Perona、D. Ra manan、P. Doll´ar 和 CL Zitnick。 Microsoft coco：上下文中的常见对象。在欧洲计算机视觉会议上，第 740-755 页。 Springer, 2014. 2 [24] T.-Y. Lin、M. Maire、S. Belongie、J. Hays、P. Perona、D. Ra manan、P. Doll´ar 和 CL Zitnick。 Microsoft COCO：上下文中的常见对象。在 ECCV，2014. 7 [25] W. Liu、D. Anguelov、D. Erhan、C. Szegedy 和 S. Reed。

Ssd：单发多盒检测器。 arXiv 预印本 arXiv:1512.02325, 2015. 7 [26] N. Mellempudi、A. Kundu、D. Mudigere、D. Das、B. Kaul 和 P. Dubey。具有细粒度量化的三元神经网络。 arXiv 预印本 arXiv:1705.01462, 2017. 1, 6 [27] M. Rastegari、V. Ordonez、J. Redmon 和 A. Farhadi。 Xnor net：使用二进制卷积神经网络的 Imagenet 分类。 arXiv 预印本 arXiv:1603.05279, 2016. 1, 2 [28] K. Simonyan 和 A. Zisserman。用于大规模图像识别的非常深的卷积网络。 arXiv 预印本 arXiv:1409.1556, 2014. 1

[29] C. Szegedy、W. Liu、Y. Jia、P. Sermanet、S. Reed、D. Anguelov、D. Erhan、V. Vanhoucke 和 A. Rabinovich。
通过卷积更深入。在 IEEE 计算机视觉和模式识别会议记录中，第 1-9 页，2015 年。1 [30] C. Szegedy、V. Vanhoucke、S. Ioffe、J. Shlens 和 Z. Wojna。

重新思考计算机视觉的初始架构。
In the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2818–2826, 2016. 6 [31] V. Vanhoucke, A. Senior 和 MZ Mao。提高 CPU 上神经网络的速度。在过程中。深度学习和无监督特征学习 NIPS 研讨会，第 1 卷，第 4 页，2011 年。2

[32] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet：一种用于移动设备的极其高效的卷积神经网络。 CoRR, abs/1707.01083, 2017. 1 [33] A. Zhou、A. Yao、Y. Guo、L. Xu 和 Y. Chen。增量网络量化：迈向具有低精度权重的无损 CNN。 arXiv 预印本 arXiv:1702.03044, 2017. 1, 6

[34] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou.
Dorefa-net：训练具有低位宽梯度的低位宽卷积神经网络。 arXiv 预印本 arXiv:1606.06160, 2016. 1, 2

[35] C. Zhu、S. Han、H. Mao 和 WJ Dally。训练有素的三元量化。 arXiv 预印本 arXiv:1612.01064, 2016. 1

Machine Translated by Google

A. 附录:特定于图层的详细信息

A.1.数学函数

双曲正切、logistic 函数和 softmax 等数学函数经常出现在神经网络中。不需要查找表,因为这些函数是在纯定点算术中实现的,类似于它们在浮点算术中的实现方式7

.

A2。添加

一些神经网络使用普通的加法层类型,它只是将两个激活数组加在一起。与浮点相比,这样的加法层在量化推理方面更昂贵,因为需要重新缩放:一个输入需要使用乘数M = S1/S2的定点乘法重新缩放到另一个的比例,类似于我们所拥有的早先看到(第 2.2 节结尾),在实际加法可以作为简单的整数加法执行之前;最后,必须再次重新缩放结果以适应输出数组的 scale8

.

A.3.级联

对连接层的完全一般支持会带来与附加层相同的重新缩放问题。因为 uint8 值的这种重新缩放将是一个有损操作,并且看起来串联应该是一个无损操作,我们更愿意以不同的方式处理这个问题:我们引入一个要求,而不是实施有损重新缩放,所有输入激活和级联层中的输出激活具有相同的量化参数。

这消除了重新缩放的需要,因此连接是无损的并且没有任何算术9

.

B. 附录:ARM NEON 细节

本节假定您熟悉 ARM NEON 指令集上的汇编编程。下面的指令助记符指的是 64 位 ARM 指令集,但讨论同样适用于 32 位 ARM 指令集。

本文中引用的定点乘法精确映射到 SQRDMULH 指令。使用正确舍入指令 SQRDMULH 而不是 SQDMULH10 非常重要。

2.2节中提到的舍入到最近的右移并不精确映射到任何 ARM NEON 指令。

问题在于“右移舍入”指令,即具有可变负偏移量的 RSHL,通过向上舍入而不是将它们从零舍入来打破平局。对于前

足够了,如果我们使用 RSHL 来实现除法−12/2 3个,
结果将是−1而它应该是−2与“round

到最近”。这是有问题的,因为它会导致整体向上偏差,据观察,这会导致神经网络推理中端到端准确性的显着损失。正确的舍入到最近的右移仍然可以使用 RSHL 来实现,但需要围绕它进行适当的修正算法 11为了高效地实现矩阵乘法的核心累加,我们使用以下技巧。

.

在(10)的乘加运算中,我们首先将操作数的类型从 uint8 更改为 int8(这可以通过从量化值和零点中减去 128 来完成)。

因此核心乘加变成

$$\text{int32} \mathrel{+}= \text{int8} * \text{int8}。 \tag{B.1}$$

如第 3 节所述,通过对量化训练过程进行细微调整,我们可以确保权重一旦量化为 int8 值,就永远不会采用值−128。因此,(B.1)中的乘积永远不会是−128 * −128,因此绝对值总是小于2 。因此,(B.1)可以先在本地 int16 累加器上累加两个乘积,然后再将其累加到真正的 int32 ac 累加器中。这允许使用 8 路 SIMD 乘法(在 int8 操作数上为 S[14] 操作数上,然后是 8 路 SIMD 乘加 [illegible] int8 累加到 int32 累加器中器(SADALP)12 。

C. 附录:图表

D. 实验方案

D.1. ResNet协议

预处理。来自 ImageNet [3]的所有图像都重新调整大小,保持纵横比,使图像的最小边为256。然后裁剪中心224 × 224块,并为每个 RGB 通道减去均值。

优化。我们使用来自 TensorFlow [1]的动量优化器,动量为0.9 ,批量大小为 32。
学习率从10−5开始,每30 个时期以0.1的阶梯方式衰减。由于第 3 节中讨论的原因,激活量化延迟了500、000步。训练使用50 个异步工作人员,并在验证精度稳定后停止,通常在100 个时期后。

---

7 至少 tanh 和逻辑函数的纯算术、SIMD 就绪、无分支、定点实现在 gemmlowp [18] 的定点目录中给出,在指令集中专门针对 NEON 和 SSE。人们可以在 TensorFlow Lite [5]中看到这些是如何调用的。

8 查看 TensorFlow Lite [5]实现。

9这在这部分实现TensorFlow Lite [5] Converter 10的定点数学函数实现在 gemmlowp [18]中使用这样的定点乘法和普通(非饱和)整数加法。我们没有用到一般的饱和算术。

11在这里实现在 gemmlowp [18] 中。

12该技术在优化的 NEON 内核中实现在 gemmlowp [18] 中,这是 TensorFlow Lite 特别使用的(请参阅此行中 L8R8WithLhsNonzeroBitDepthParams 的选择)。

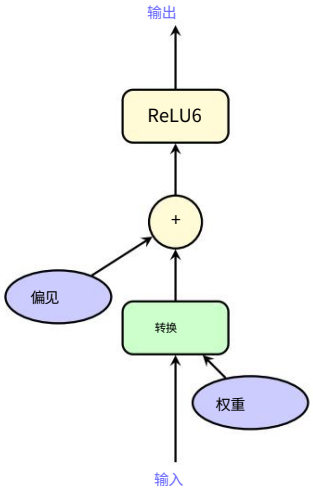


图 C.1:简单图形:原始

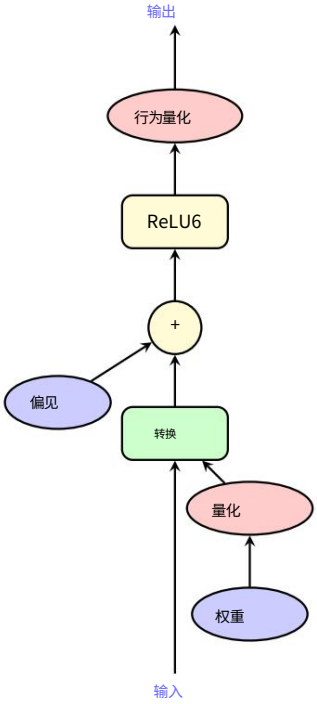


图 C.2:简单图:量化

D.2.初始协议

表4.3中的所有结果都是在训练了大约1000万步之后获得的,每批32个样本,使用50 个分布式 worker,异步。训练数据是带标签的 ImageNet 2012 299 × 299图像。图像增强包括:随机裁剪、随机水平翻转和随机颜色失真。使用的优化器是 RMSProp,学习率从0.045开始,de

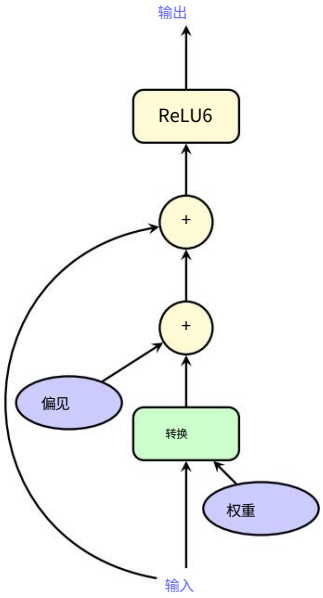


图 C.3:带有旁路连接的图层:原始

每2 个epochs 后,以0.94 的因子呈指数级逐步下降。其他 RMSProp 参数为： 0.9动量、 0.9衰减、 1.0 epsilon 项。训练参数是 EMA 平均衰减0.9999。

D.3. COCO检测协议

预处理。在训练期间,所有图像被随机裁剪并调整为320 × 320 。在评估期间,所有图像直接调整为320 × 320 。所有输入值都归一化为[−1, 1]。

优化。我们使用了 TensorFlow [1]中的 RMSprop 优化器,批量大小为32。学习率从4 × 10−3开始,每100 个epoch以0.1倍的阶梯方式衰减。由于第 3 节中讨论的原因,激活量化延迟了500、000步。训练使用20个异步工作人员,并在验证精度稳定后停止,通常在大约600万步之后。

指标。评估结果使用 COCO 主要挑战指标报告:IoU=.50:.05:.95 的 AP。我们遵循[13] 中相同的训练/评估拆分。

D.4.人脸检测与人脸属性分类协议

预处理。随机 1:1 裁剪取自[10]中使用的基于 Flickr 的数据集中的图像,并将大小调整为320×320像素用于人脸检测， 128×128像素用于人脸属性分类。生成的作物以50% 的概率水平翻转。每个 RGB 通道的值被重新归一化为在[−1, 1] 范围内。

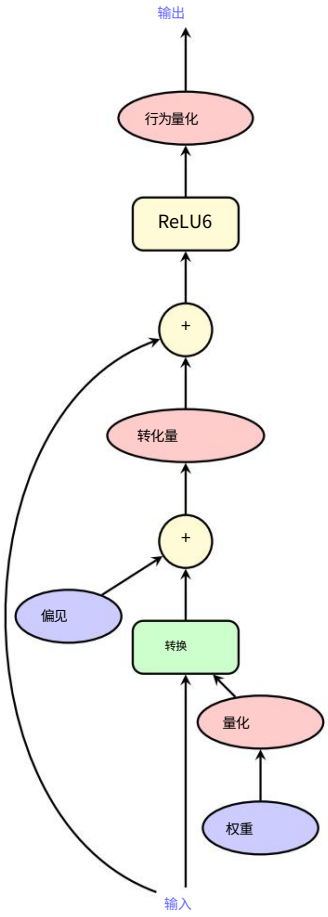


图 C.4:具有旁路连接的层:量化

人脸检测优化。我们使用了 TensorFlow [1]中的 RMSprop 优化器，批量大小为32。学习率从4 × 10－3开始，每100轮以0.1倍的阶梯方式衰减。出于第 3 节中讨论的原因，激活量化延迟了500、000步。训练使用20个worker asynchronously，并在验证精度稳定后停止，通常在大约300万步之后。

人脸属性分类优化。我们遵循了 [10] 中的优化协议。我们使用了 Tensorflow [1] 的 Ada grad 优化器，批量大小为32 ，学习率为0.1。训练异步使用12 个worker，并在2000万步处停止。

延迟测量。我们创建了一个二进制文件，它对随机输入重复运行人脸检测和人脸属性分类模型100秒。我们使用 adb push 命令将此二进制文件推送到 Pixel 和 Pixel 2 手机，并使用指定了适当任务集的 adb shell 命令在 1、2 和 4 个小核以及 1、2 和 4 个大核上执行它。我们报告了面部检测器模型的平均运行时间

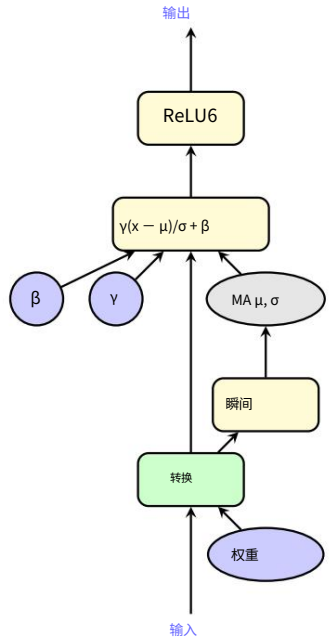


图 C.5:具有批量归一化的卷积层:训练图

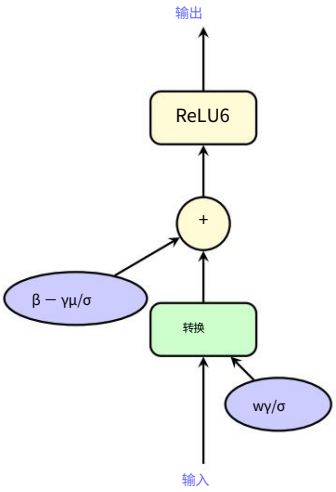


图 C.6:具有批量归一化的卷积层:推理图

320 × 320输入，以及128 × 128输入的人脸属性分类器模型。

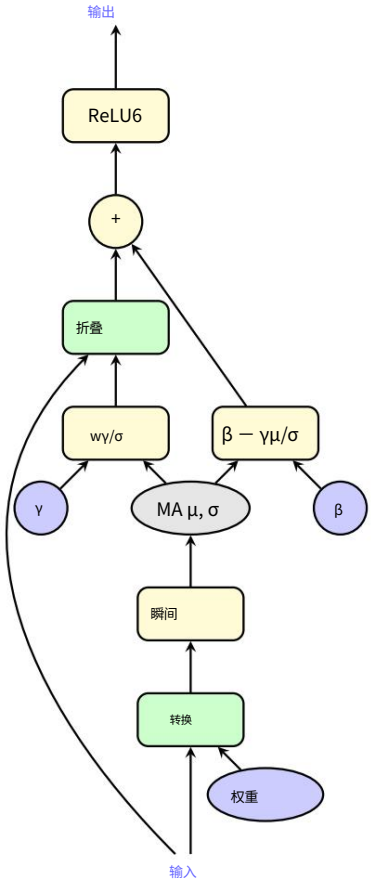


图 C.7:具有批量归一化的卷积层:训练图,折叠

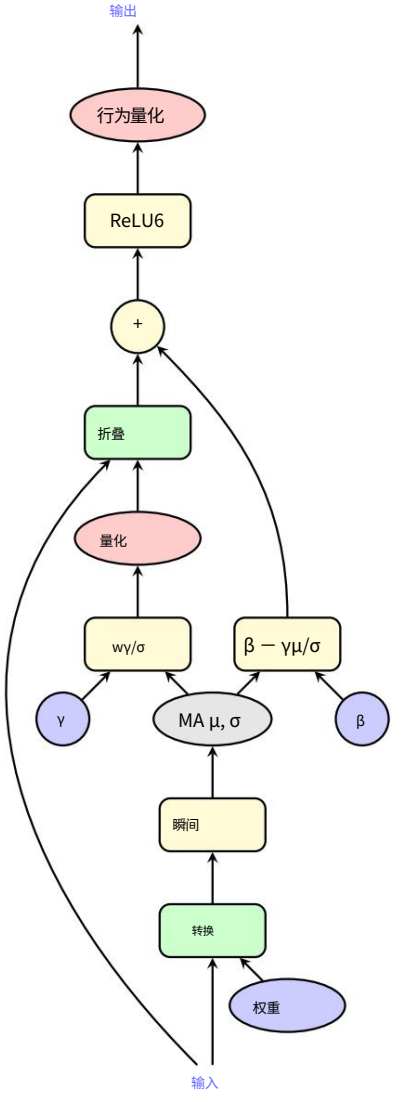


图 C.8:具有批量归一化的卷积层:训练图,折叠和量化