arXiv:2304.08925v1 [cs.LG] 18 Apr 2023

# 深入理解数据预处理，助力深度神经网络高效端到端训练

PingGong[1], YuxinMa[1],Cheng Li[1,4],Xiaosong Ma[2],SamH. Noh[3]1中国科学技术大学 2卡塔尔计算研究院 3弗吉尼亚理工大学4安徽省高性能计算重点实验室

**摘要**

本文主要关注公有云中深度神经网络训练的数据预处理流水线。首先，我们通过实验测试采用原始数据或记录文件的两种主要数据预处理方法对性能的影响。初步结果表明，即使启用了 NVIDIA DALI 这一高度优化的数据预处理库所支持的最高效软硬件配置，数据预处理仍然是明显的瓶颈。其次，我们找出潜在原因，尝试多种优化方法，并阐述其优缺点。我们希望这项工作能为“数据存储、加载流水线”与“训练框架”的新型协同设计，以及二者之间灵活的资源配置提供参考，从而充分利用资源并最大限度地提升性能。

## 1 引言

深度学习（DL）取得巨大成功的一个重要因素是存在多样且规模庞大的数据集，另一个因素则是算力充足，这些算力大多来自图形处理器等加速器。MXNet [19], TensorFlow [25], 和 PyTorch [21], 等广受欢迎的深度学习框架，将这两大驱动力结合起来，用于训练深度神经网络（DNN）模型：它们从图像、视频和文本等原始数据中迭代取出批次样本，为图形处理器提供数据。在由大量 GPU 核心高效处理之前，原始数据必须先加载到内存中，并依次经过解码、裁剪、缩放和归一化等变换算子处理。

为了有效利用当今可用的大规模 GPU 并行性，数据预处理也必须足够高效，持续输送数据以满足 GPU 的需求。然而，现有研究大多只关注训练部分，往往忽视了数据预处理流水线 [7, 15, 18]。不过，值得庆幸的是，近期已有少数研究开始关注这一问题。例如，Uber 和 Apple 推出了各自的数据平台，提供用于存储和管理机器学习任务数据的数据模型和方法 [1, 23]。Yang 和 Gong 提出了一种分布式缓存，用于加速分布式 DNN 训练中数据准备的输入/输出步骤 [27], ；而 OneAccess 则采用了类似思路，以避免并发超参数调优实验在共享数据集时重复加载数据 [13]。然而，现有研究均未全面概述深度学习数据预处理流程、其性能表现或其中涉及的主要系统设计问题。

本研究考察了 TensorFlow、MXNet 和 PyTorch 这三种主流框架中的数据预处理流水线，以及它们对整体训练效率的影响。我们的主要发现如下：

- 广泛使用的深度学习框架长期以来都采用较长且高度相似的数据预处理流水线，并普遍选择相同的数据格式和数据结构。
- 与此同时，不同的深度神经网络模型在数据预处理和实际训练中的相对速度也各不相同。大多数研究都聚焦于训练性能，因此多数模型训练工作流的瓶颈出现在数据加载和预处理阶段。
- 尽管上述三种主流框架都在中央处理器上进行数据预处理，并在图形处理器上进行训练，但英伟达的新框架 DALI [11] 通过支持中央处理器与图形处理器协同处理训练数据样本，确实提升了整体性能。不过，它也存在一些限制，例如可能发生内存溢出（OOM）；目前需要手动寻找合适的批量大小，以避免内存溢出，同时保持较高的吞吐量。
- 然而，单个耗时步骤（例如目前由 DALI 在中央处理器端执行的图像解码）可能主导整个流水线的执行时间，导致图形处理器等待数据。要消除这一瓶颈，就需要增加CPU资源。
- 我们的初步结果表明，更灵活的节点配置有助于以更低成本运行各种深度学习流水线：为 AlexNet 和 ResNet50 分配更多 CPU 资源后，训练速度分别提升了 7.86% 和 3.03%；而在性能基本相当的情况下，ResNet50 的 CPU 资源分配量可减少 75%。从基于 SSD 的存储选项加载数据样本，几乎能让 ResNet18 达到最佳训练速度；而使用动态随机存取存储器可使 AlexNet 的速度提升 1.84× 倍。

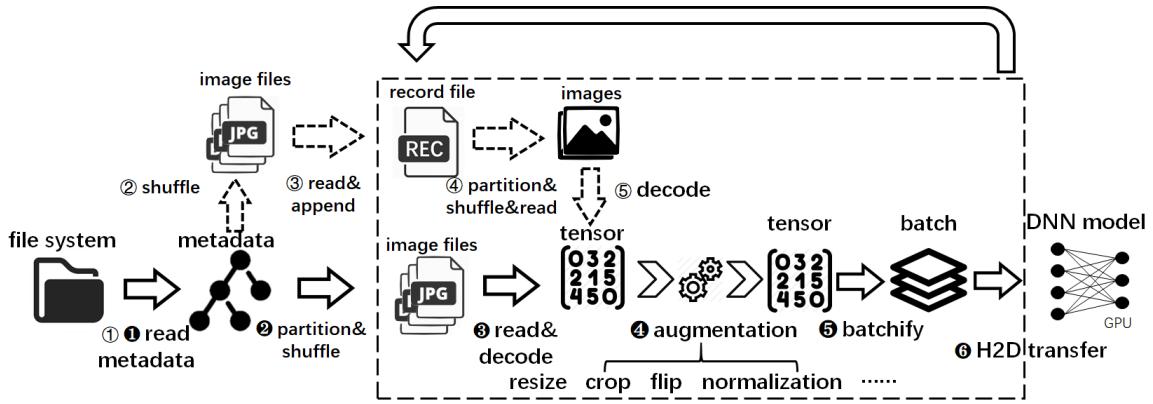



**图 1：端到端训练流水线，其中第一阶段是针对计算机视觉任务的数据预处理。**

我们的研究支持了关于资源解耦 [4, 5, 8, 14],的观点。这是云基础设施设计中的一种新趋势，其特点是采用可组合、解耦的资源部署方式，在保持高度灵活性的同时实现相当的性能 [22, 24]。目前，Google Cloud 和 EC2 等一些云服务提供商已在一定程度上支持这一功能，允许调整图形处理器机器的 vCPU 数量，但仍受特定限制 [6]。我们认为，本文所述的数据预处理与训练资源平衡问题，为即将推出的云资源解聚服务提供了一种普遍且资源密集型的应用场景。具体而言，为避免耗时的手动配置，应提供工具自动完成此类配置和扩缩决策，从而提高用户整体的深度学习模型训练吞吐量（按小时或按美元计算），同时也改善云服务提供商的整体系统利用率。

## 2 理解训练流水线

要获得准确的结果，必须使用大量数据训练深度神经网络模型。然而，现实世界中的数据集通常不适合直接输入神经网络。数据集通常需要经过数据预处理流水线，包括加载、解码、解压缩、数据增强、格式转换和调整大小。随着加速器（包括 GPU）的性能迅速提升，端到端训练中的这一数据加载流水线阶段必然会变得越来越重要。

### 2.1 深度神经网络模型的迭代训练

深度神经网络模型的训练计算密集且耗时。为了缩短训练过程并使深度神经网络模型能够快速部署，目前的常见做法是利用多个图形处理器所提供的大规模并行能力，按照小批量随机梯度下降（SGD）算法进行协同训练 [17]。这一训练阶段在图 1 最右侧以“深度神经网络模型”标示，是大多数提升深度神经网络训练效率的研究重点。该训练过程以迭代方式进行：每一轮都会获取一小批数据样本，并将其加载到 GPU 内存中，作为神经网络模型训练的输入。不过，对于此类训练，通常需要将存储在持久化存储中的应用专用原始数据转换为张量，这是一种供各种训练框架使用的统一数据格式。我们将这一阶段称为数据预处理流水线，如图 1 左侧所示。

### 2.2 数据预处理流水线概述

我们分析了数据预处理流水线的四种不同实现方式。前三种是三种广泛采用的训练框架 TensorFlow、MXNet 和 PyTorch 的内置解决方案，最后一种是由英伟达开发的统一接口 DALI [11]，适用于上述三种框架。尽管它们由不同的社区实现，但其流水线的组成几乎相同。下文将介绍两种主要的数据预处理方法，即从原始数据加载和从记录文件加载。为简便起见，本文以图像处理场景为例进行说明。不过，我们发现，这一通用场景适用于从自然语言处理（NLP）到视频处理的各类任务。

*2.2.1* 原始数据预处理。图 1 中黑色圆圈里的数字表示原始数据的预处理流程。

❶ 首先，数据预处理器（简称 DPP）读取一个有序的元数据文件，并将其作为字典存储在内存中，以便稍后检索实际数据。元数据文件通常在线下生成，其中的每个元组都包含每张图像的索引（按顺序分配）、标签和路径。然后，❷ DPP 将整个文件标识符列表划分为一组较短的序列并打乱顺序。❸ DPP 接着从每个序列中获取文件 ID，利用这些 ID 在内存字典中查找对应的标签和路径，加载路径所指向的原始图像文件，并将其解码为张量。原始图像文件的张量准备就绪后，❹ DPP 进入“数据增强”步骤，通过随机裁剪、翻转、旋转等一系列操作将张量转换为新的张量。最后，❺ 数据加载器将多个张量组合成批次，❻ 并将这些批次加载到 GPU 上。

*2.2.2* 记录文件预处理。 如图 1 中白色圆圈内的数字所示，除了原始数据预处理之外，还有另一种方法，即使用记录文件。这些记录文件通过离线处理生成：首先从磁盘读取大量存储的原始图像文件，然后将它们追加到少量大型顺序记录文件中。这些文件包含图像及其标签和 ID。随后，在运行时，这些记录文件会被读取到内存中，并划分为较小的数据块。这些数据块会被解码，此后的步骤（❹ - ❻）与原始数据预处理方法相同。

设计这种记录文件加载方法主要有两个原因。首先，与原始数据相比，记录文件将随机访问转换为顺序访问；如果数据存储在慢速存储设备上，或分散在多台机器上，就能因此提升性能。其次，可以将数据预处理流水线中的一些算子（例如裁剪和缩放）转移到记录文件生成阶段，以提高数据加载效率。从本质上说，记录文件加载方法是以额外的处理时间和存储空间换取训练速度的提升。

最后，人们可能会认为，将数据预处理流水线中的所有任务都移至离线执行是可行的，这样就能优化运行时。虽然从性能角度看这确实行得通，但这可能并非深度神经网络的最佳做法。这是因为训练过程需要数据样本输入具备一定的随机性，以提升深度神经网络模型的鲁棒性和通用性；这可以通过在流水线中执行随机旋转和翻转等非确定性算子来实现。要在深度神经网络中取得最佳效果，就需要这种随机性。因此，更可取的做法是仅将图像追加到顺序记录文件中，而不进行进一步的预处理。

*2.2.3 CPU-GPU* 协同处理。我们考虑的四种数据加载流水线实现都支持原始数据和记录文件两种加载方法。TensorFlow、MXNet 和 PyTorch 框架中内置的三种数据加载库与 DALI 的主要区别在于：前三者仅在 CPU 上运行数据加载流水线，而 DALI 则利用 GPU 加速任务，将流水线的部分处理卸载到 GPU 上。有关如何通过设置 device 选项，手动配置 CPU-GPU 协同处理以支持数据预处理算子，请参阅 DALI 代码示例 [3], 。这种方法的优点是能够改进数据加载流水线，但缺点是会占用 GPU 资源。我们在使用 FP32 精度、批量大小为 512 的设置训练 ResNet18 时遇到了 GPU 内存溢出（OOM）错误；将批量大小降至 384 后，错误得以消除。启用此功能可能会迫使我们减小批量大小，以便在 GPU 内存中为 DNN 训练阶段腾出空间。此外，这类调整也需要手动进行，过程可能十分繁琐。

## 3 端到端训练性能分析

本节展示各种深度神经网络模型端到端训练的实验结果。基于这些结果，我们讨论数据加载流水线对整体性能的影响，以及它如何导致训练过程中资源利用不足。

### 3.1 实验设置

**硬件和软件配置：**我们的实验使用 AWS p3.16xlarge 高端 GPU 实例，配备 2.3GHz（基础频率）的 Intel Xeon E5-2686 v4 处理器、488GB 动态随机存取存储器、64 个 vCPU 和 8 个 NVIDIA Tesla V100 GPU（每个 GPU 配备 5,120 个 CUDA 核心、640 个 Tensor 核心和 16GB GPU 内存）。使用的操作系统和其他软件包括 ubuntu18.04、CUDA 10.2、NVIDIA DALI 0.19.0、NVIDIA APEX 0.1 和 PyTorch 1.4.0。除非另有说明，我们使用 ext4 将训练数据存储在 EBS [2], 上，以获得持久、高吞吐量的实例本地块存储。

**预处理流程配置：**我们的数据预处理流程包含以下算子：图像解码、随机裁剪、缩放至 (224,224)、随机水平翻转和归一化。我们评估了原始数据和记录文件预处理，这两种方式均采用仅由中央处理器执行或由中央处理器与图形处理器混合执行，共形成四种组合：原始 CPU、记录 CPU、原始数据-混合处理和 record-hybrid。结果发现，DALI 的性能优于主流深度学习框架内置的工具 [11 , 20]（我们的测试也证实了这一点），因此我们在所有测试中都采用这种做法。受篇幅限制，我们仅展示 PyTorch 的结果；MXNet 和 TensorFlow 的表现非常相似。

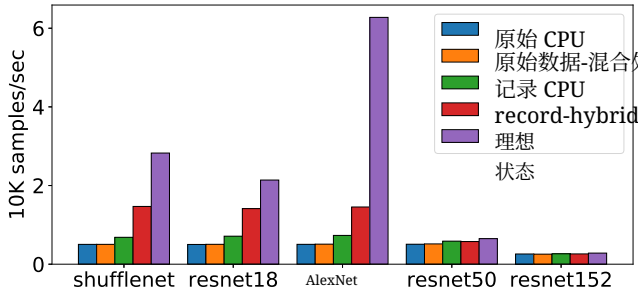



**图2：端到端训练性能**

**模型和数据集：**我们在广泛使用的 ImageNet 数据集 [10] 上训练五种深度神经网络模型，分别是 AlexNet [16], ShuffleNet [26], ResNet18[12], ResNet50 [12],和 ResNet152 [12],。我们启用混合精度训练（开启 FP16 选项），以充分利用图形处理器资源。对于 AlexNet、ShuffleNet 和 ResNet18，我们采用 512 的批量大小，因为更大的批次几乎无法带来性能提升。其余两种模型由于层数更多、规模大得多，因此受内存限制，ResNet50 的批量大小设为 192，ResNet152 的批量大小设为 128。

**指标：**我们以每秒样本数衡量数据预处理和模型训练吞吐量。作为参考，我们还绘制了“理想状态”下的训练性能，即训练重复使用预先加载到 GPU 内存中的同一批次。此外，我们还测量中央处理器和图形处理器的利用率。

## 3.2 总体性能

图2展示了不同预处理方法下的总体训练性能。首先，大多数模型在使用记录文件后性能有所提升，采用混合数据预处理时提升尤为明显。前者得益于更高比例的顺序I/O，即使使用 NVMe 固态硬盘，这一优势依然显著。后者则让高性能图形处理器能够减轻中央处理器上的数据预处理瓶颈，从而实现更均衡的系统利用率。

然而，我们观察到，在五种模型中，有三种模型的最佳端到端训练吞吐量明显低于理想状态。这表明数据预处理流水线的性能会造成影响，并导致图形处理器利用率不足。也就是说，数据预处理流水线无法足够快地向图形处理器供给数据，使其未能得到充分利用；这一点将在下一节中得到验证。具体而言，对于 AlexNet，采用 record-hybrid 时的峰值性能仅为理想状态的 23%。针对这三种数据消费速度较快的模型，混合数据预处理带来的帮助最大（绿色柱与红色柱之间的大幅跃升即为例证）。由于图形处理器原本会因缺少数据而闲置，混合方式可利用它们将数据准备工作从中央处理器卸载出去。与 record CPU 相比，这三种模型的吞吐量提升了 98% 至 114%，

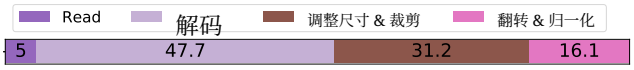



**图3：100% 堆叠条形图展示了在中央处理器上预处理单张图像时各主要步骤的延迟分解，总耗时为 14.26 毫秒（数值为百分比）。**

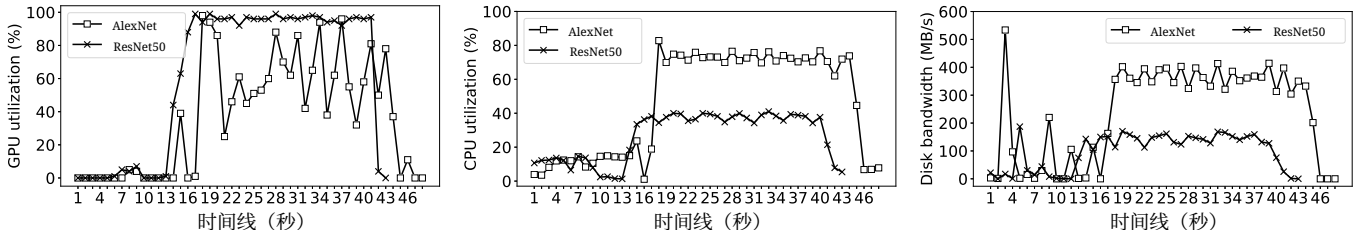
而有趣的是，即便对于这三种模型，使用原始文件时混合处理也没有帮助，因为随机 I/O 主导了加载过程，单靠中央处理器就能达到相同的 I/O 速度。相比之下，ResNet50 和 ResNet152 在训练时的计算密集程度要高得多（其理想训练吞吐量远低于其他模型），因此数据加载的影响较小。

为进一步探究为何将部分预处理任务卸载到图形处理器能够显著提升训练性能，我们在此报告数据加载流水线中各步骤的延迟分解。为此，我们不启用训练，而只运行数据预处理流水线并测试其中各个算子的性能。图3展示了各数据增强算子所耗费的时间。请注意，这里报告的是仅使用中央处理器进行数据预处理时的分解结果。结果表明，所有数据预处理算子的耗时几乎占整个流水线成本的 95%。其中，图像解码最耗时，占总耗时的 47.7%。在我们的实验中使用 record-hybrid 时，数据预处理会将算子拆分，使大部分解码及后续操作由图形处理器执行，从而缩短整体预处理时间。

需要注意的是，record-hybrid 带来的性能提升也有代价。首先，需要额外进行离线处理来生成记录文件。其次，有效的流水线设计要求在图形处理器上同时进行数据预处理和模型训练。这会带来两个后果：一是可能导致内存溢出错误，需要人工干预以避免；二是如图5所示，由于共享图形处理器，训练吞吐量可能会受到负面影响。

## 3.3 资源利用率

图4分别绘制了 AlexNet 和 ResNet50 在采用 record-hybrid 的训练过程中各类资源的利用率，它们分别代表数据消费速度较快和较慢的情况。值得注意的是，每条曲线最初约三分之一的部分对应训练初始化。对于 ResNet50，在整个过程中最后约三分之二的时间里，GPU 几乎始终处于饱和状态，而 CPU 利用率

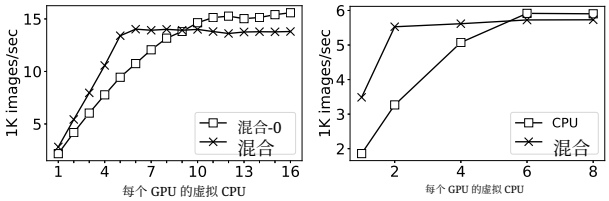



(a) 图形处理器　(b) 中央处理器　(c) 输入/输出

**图4：图2中表现最佳的 record-hybrid 实验对应的 AlexNet 和 ResNet50 的资源统计。**

**表 1：AWS EC2 [9]（上）和 Google Cloud [6]（下）上常用于深度神经网络模型训练的虚拟机实例。所有 GPU 均为 V100。**

| Type | #GPU | #虚拟CPU | I/O | $/h |
|---|---|---|---|---|
| p3.2xlarge | 1 | $\leq 8$ | 可配置 | $< 3.06$ |
| p3.16xlarge | 8 | $\leq 64$ | 可配置 | $< 24.48$ |
| p3dn.24xlarge | 8 | $\leq 96$ | 可配置 | $< 31.21$ |
| V100-1 | 1 | $\leq 12$ | 选项 | $< 3.22$ |
| V100-4 | 4 | $\leq 48$ | 选项 | $< 12.90$ |
| V100-8 | 8 | $\leq 96$ | 选项 | $< 25.80$ |




(a) 调整 AlexNet 的虚拟CPU数量　(b) 调整 ResNet50 的虚拟CPU数量

**图5：为图形处理器调整 CPU 资源后 AlexNet 和 ResNet50 的端到端训练性能**

仅为 38%，远低于其最大容量；输入/输出带宽消耗也仅为 147MB/s，同样远低于最大容量。我们观察到，ResNet50 受图形处理器限制，其训练数据消耗速度足够慢，可以掩盖输入/输出和数据预处理的耗时。因此，我们发现中央处理器和存储设备未得到充分利用。我们还发现，尽管图形处理器资源十分宝贵，DALI 却将这些资源用于数据加载，这并非明智之举。

另一方面，AlexNet 是一个快速的数据消费方，能够迅速处理已加载的数据，随后等待更多数据到来。这反映在图形处理器利用率的波动上。总体而言，图形处理器的利用率低于 50%。相比之下，我们发现其中央处理器利用率远高于 ResNet50（图4b），输入/输出带宽也明显更高（图4c）。我们发现，record-hybrid 在这种情况下很有帮助，因为它能提高图形处理器的利用率。然而，我们也发现，中央处理器不足以满足图形处理器的数据需求，因此需要为数据预处理流水线分配更多中央处理器。

## 4 资源平衡的影响

公有云提供商开始提供更灵活的实例配置。表 1 列出了 Amazon EC2 和 Google Cloud 上可用的 GPU 虚拟机实例。对于给定的实例类型及其固定的 GPU 资源，两种平台都允许用户在预设范围内指定所需的虚拟 CPU 数量。EC2 还允许配置输入/输出吞吐量（与所请求的持久存储容量成正比），而 Google Cloud 仅提供有限的存储选项。如此灵活的配置也带来了更精细的计费策略。例如，在所列出的 Google Cloud 实例中，GPU 和虚拟 CPU 的费用分别为每小时 2.48 美元和 0.033 美元，而内存费用为每 GB 小时 0.0044 美元。

虽然这种灵活的配置为用户提供了更多选择，但为深度神经网络模型选择最佳配置并非易事。我们的最终目标是开发一种工具，在保持高吞吐性能的同时，为模型训练工作流提出针对特定模型的细粒度资源配置。上一节所述的研究发现表明，这种工具将很有价值。

在本节中，我们展示了该工具开发完成后可能取得的一些结果。具体而言，我们研究了调整虚拟 CPU 数量对性能的影响。出于篇幅考虑，我们再次聚焦于前文评估过的两个代表性深度神经网络模型：AlexNet 和 ResNet50。

首先，考虑预处理受限的 AlexNet：将 GPU 数量固定为 4，并调整

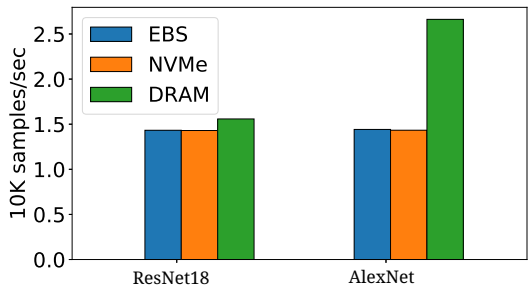



**图6：从不同存储选项加载数据样本时，ResNet18 和 AlexNet 的端到端训练性能**

虚拟 CPU 的数量。当 4 个 GPU 可使用全部 64 个虚拟 CPU 时，我们研究另一种加载方法（除前面评估的混合模式外）：在中央处理器上进行解码，其余预处理则在图形处理器上完成，称为 hybrid-0。此模式会使用更多 CPU 资源，同时将开销更大的 GPU 资源留给训练。回想一下，在混合模式下，图像解码由中央处理器和图形处理器共同完成。图 5a 显示，混合模式在 24 个虚拟 CPU（每个 GPU 6 个）时训练速度达到饱和，而 hybrid-0 将饱和点推迟到 44 个虚拟 CPU（每个 GPU 11 个）。因此，训练速度提高了 7.86%，如图中超过 11 个虚拟 CPU 后更高的曲线所示。

现在考虑训练受限的 ResNet50，此时图形处理器显然是瓶颈。因此，我们在这些实验中使用全部 8 个图形处理器。这里可以探究如何在不降低性能的情况下减少虚拟 CPU 分配，或通过重新配置 CPU-GPU 协同处理任务来提升性能。图5b 证实，在此类实例中，所需虚拟 CPU 数量远低于可用数量。例如，在混合模式下，16 个虚拟 CPU（每个图形处理器 2 个）即可充分供给图形处理器。为进一步将图形处理器留给耗时的训练，我们将 DALI 优化改回仅使用中央处理器进行数据准备，即 CPU 模式，图中标记为 CPU。现在需要 48 个虚拟 CPU（每个图形处理器 6 个）才能使训练流水线达到饱和，这会增加中央处理器成本，而训练速度仅提升 3.03%。我们还观察到（此处未显示），从 ResNet50 转向层数更深的 ResNet152 时，训练流水线对图形处理器的依赖程度更高，虚拟 CPU 需求降至 8 个虚拟 CPU（每个图形处理器 1 个）。请注意，在 AlexNet 和 ResNet50 两个示例中，我们都观察到，尽管使用图形处理器执行数据预处理流水线有助于流水线运行，却会降低吞吐量。

最后，我们将注意力转向探究使用不同存储介质托管训练数据对性能的影响。为此，我们部署了 AlexNet 和 ResNet18，并在 EC2 p3dn.24xlarge 实例上使用 4 个 V100 GPU 进行训练，每个 GPU 配备 12 个虚拟CPU 核心。我们更换存储设备进行测试，包括 EBS（最高 7500 每秒输入输出操作数）、两个 NVMe 固态硬盘（EC2 默认配置）和动态随机存取存储器。图 6 展示了端到端训练性能。

我们观察到，直接从 EBS 和 NVMe 固态硬盘加载数据样本（对应图 1 中的 4），对于这两个模型而言，训练性能几乎相同。这是因为 EBS 是一种经过高度优化、采用 SSD 的存储选项，其提供的 I/O 带宽与所连接的 NVMe 固态硬盘相近。有趣的是，从动态随机存取存储器加载数据样本时，ResNet18 的训练性能仅提升了 8.8%。这是因为 ResNet18 和 ResNet50 一样，都是计算密集型模型，而 EBS 和 NVMe 固态硬盘的带宽足以满足图形处理器训练阶段的需求。相比之下，使用动态随机存取存储器可使 AlexNet 的训练性能相较于另外两种存储选项提升 1.84× 倍。这是因为 AlexNet 的计算速度远快于 ResNet18，因此需要更大的 I/O 带宽，超出了 EBS 和 NVMe 固态硬盘的容量。因此，存储配置对性能的影响取决于模型。对于某些模型，投资于高带宽存储可以显著提升深度神经网络训练的整体成本效益。

## 5 结论与未来方向

人们普遍认为，深度神经网络训练需要大量的图形处理器/加速器资源和网络基础设施。尽管许多研究都聚焦于通过更好地管理这些资源来加快训练速度，但我们认为，另一个同样重要的方面迄今在很大程度上被忽视了。具体而言，必须向加速器供给数据，才能充分利用加速器。

在这项工作中，我们考察了广泛使用的深度学习模型训练框架中的数据准备流程。我们发现，不同框架的数据加载和预处理流程高度相似，但在主流深度神经网络模型中，数据准备相对于内存中模型训练的开销差异显著。根据性能测试结果，我们认为，该流程目前效率低下，常常造成大量资源浪费，尤其是昂贵的 GPU 资源未得到充分利用。我们进一步验证了 CPU-GPU 数据准备中混合方式的有效性，并提出了针对特定模型的细粒度云实例配置方案，以实现高效且经济的模型训练。

在未来的工作中，我们将把性能分析扩展到自然语言处理和视频处理模型。此后，

我们计划构建一种工具，以便根据不同深度学习训练工作负载的需求自动配置资源。

## 参考文献

[1] Pulkit Agrawal、Rajat Arya、Aanchal Bindal、Sandeep Bhatia、Anupriya Gagneja、Joseph Godlewski、Yucheng Low、Timothy Muss、Mudit Manu Paliwal、Sethu Raman 等。2019年。机器学习数据平台。载于*2019*年国际数据管理会议论文集（*SIGMOD '19*）。计算机协会，纽约，纽约州，美国，1803 1816 10 1145 3299869 3314050 – 。[2] 2020 https://doi.org/ 。 / 。AWSEBS 。*Amazon* 弹性块存储——易于使用、可在任意规模下提供高性能的块存储。检索于2020年3月，网址：https://aws.amazon.com/ebs/ [3] DALI 代码示例，2019年。指定 *DALI* 所支持的各种数据加载方法的训练脚本。检索于2020年3月，网址：https://github.com/NVIDIA/DALI/blob/master/docs/examples/ use_cases/pytorch/resnet50/main.py [4] Joel Dedrick。2019年。解耦云基础设施的必要性。检索于2020年3月，网址：https://www.sdwanresource.com/articles/441334-need-disaggregated-cloud-infrastructures.htm [5] Peter X. Gao、Akshay Narayan、Sagar Karandikar、Joao Carreira、Sangjin Han、Rachit Agarwal、Sylvia Ratnasamy 和 Scott Shenker。2016年。资源解耦的网络需求。载于第*12*届 *USENIX* 操作系统设计与实现大会论文集（*OSDI'16*）。USENIX 协会，美国，249–264。[6] Google GPU 实例，2020年。*Google Cloud* 虚拟机系列。检索于2020年3月，网址：https://cloud.google.com/compute/docs/gpus [7] Priya Goyal、Piotr Dollár、Ross B. Girshick、Pieter Noordhuis、Lukasz Wesolowski、Aapo Kyrola、Andrew Tulloch、Yangqing Jia 和何恺明。2017年。精确的大批量随机梯度下降：1小时内训练 ImageNet。*arXiv* abs/1706.02677（2017年）。[8] Sangjin Han、Norbert Egi、Aurojit Panda、Sylvia Ratnasamy、Guangyu Shi 和 Scott Shenker。2013年。支持下一代数据中心资源解耦的网络。载于第十二届 *ACM* 网络热点问题研讨会论文集。ACM，10。[9] EC2 GPU 实例。2020年。*AWS EC2* 虚拟机系列。检索于2020年3月，网址：https://aws.amazon.com/ec2/instance-types/ [10] Jia Deng、Wei Dong、Richard Socher、Li-Jia Li、Kai Li 和李飞飞。2009年。ImageNet：大型层次化图像数据库。载于*2009*年 *IEEE* 计算机视觉与模式识别会议。248–255。https://doi.org/10.1109/CVPR.2009.5206848[11] Joaquin Anton Guirao、Krzysztof Lecki、Janusz Lisiecki、Serge Panev、Michal Szolucha、Albert Wolant 和 Michal Zientkiewicz。2019年。使用 *NVIDIA DALI* 进行快速人工智能数据预处理。检索于2020年3月，网址：https: // devblogs.nvidia.com/fast-ai-data-preprocessing-with-nvidia-dali/ [12] 何恺明、Xiangyu Zhang、任少卿和Jian Sun。2016年。用于图像识别的深度残差学习。载于*2016*年 *IEEE* 计算机视觉与模式识别会议（*CVPR*）。770–778。https: //doi.org/ 10 1109 2016 90 。 /CVPR。 。[13] Aarati Kakaraparthy、Abhay Venkatesh、Amar Phanishayee 和 Shivaram Venkataraman。2019年。统一机器学习集群中数据加载的理由。载于第*11*届 *USENIX* 云计算热点研讨会（*HotCloud 19*）。USENIX 协会，伦顿，华盛顿州。https://www.usenix.org/conference/hotcloud19/presentation/kakaraparthy [14] Ana Klimovic、Christos Kozyrakis、Eno Thereska、Binu John 和 Sanjeev Kumar。2016年。闪存解耦。载于第十一届欧洲计算机系统会议论文集。ACM，29。[15] Alexandros Koliousis、Pijika Watcharapichat、Matthias Weidlich、Luo Mai、Paolo Costa 和 Peter Pietzuch。2019年。Crossbow：扩展深度使用小批量大小在多 GPU 服务器上进行学习。*VLDB* 会议论文集12，11（2019年7月），1399–1412。https://doi.org/10.14778/3342263.3342276[16] Alex Krizhevsky、Ilya Sutskever 和 Geoffrey E. Hinton。2017年。使用深度卷积神经网络进行 ImageNet 分类。*ACM* 通讯 60, 6 2017 84 90 10 1145 3065386 （5月），–[17] 2014 。https://doi.org/ 。 / Mu Li 、 Tong Zhang 、 Yuqiang Chen 和 Alexander J. Smola。 。面向随机优化的高效小批量训练。载于第*20*届 *ACM SIGKDD* 知识发现与数据挖掘国际会议论文集（*KDD '14*）。计算机协会，纽约，纽约州，美国，661–670。https://doi.org/10.1145/2623330.2623612[18] Xiuhong Li、Yun Liang、Shengen Yan、Liancheng Jia 和 Yinghan Li。2019年。一种用于在 GPU 上高效执行通用矩阵乘法的协同分块与批处理框架。载于第*24*届并行编程原理与实践研讨会论文集。229–241。[19] MXNet，2020年。MXNet：可扩展深度学习框架。检索于2020年3月，网址：https://mxnet.incubator.apache.org/ [20] Pieterluitjens。2019年。NVIDIA DALI：加速 PyTorch。检索于2020年3月，网址：https://towardsdatascience.com/nvidia-dali-speeding- up-pytorch-876 80182440 c Pytorch 。*Pytorch* 官方网站。检索于2020年[21] 2020 3月，网址：https://pytorch.org/ [22] Qiumin Xu、Manu Awasthi、Krishna T. Malladi、Janki Bhimani、Jingpei Yang 和 Murali Annavaram。2017年。容器化应用在本地与远程存储上的性能分析。载于第*33*届大规模存储系统与技术国际会议论文集（*MSST'17*）。[23] Robbie Gruener、Owen Cheng 和 Yevgeni Litvin。2018年。*Petastorm* 简介：*Uber ATG* 面向深度学习的数据访问库。检索于2020年3月，网址：https://eng.uber.com/petastorm/ [24] Yizhou Shan、Yutong Huang、Yilun Chen 和 Yiying Zhang。2018年。LegoOS：一种用于硬件资源解耦的分散式分布式操作系统。载于第*13*届 *USENIX* 操作系统设计与实现研讨会（*OSDI 18*）。USENIX 协会，卡尔斯巴德，加利福尼亚州，69–87。https://www.usenix.org/conference/osdi18/presentation/shan [25] TensorFlow，2020年。*TensorFlow*。检索于2020年3月，网址：https://www.tensorflow.org/ [26] Xiangyu Zhang、Xinyu Zhou、Mengxiao Lin 和 Jian Sun。2018年。ShuffleNet：一种面向移动设备的极高效卷积神经网络。载于*2018*年 *IEEE/CVF* 计算机视觉与模式识别会议。6848–6856。[27] Chih-Chieh Yang 和 Guojing Cong。2019年。加速深度神经网络训练中的数据加载。*2019*年 *IEEE* 第*26*届高性能计算、数据与分析国际会议（*HiPC*）（2019年12月）。https://doi.org/10.1109/hipc.2019.00037