DeepGini:一个对大量测试数据进行排序以降低标签成本的工具

简介：深度神经网络（DNN）已在许多安全关键型场景中被广泛采用。尽管这些基于 DNN 的系统在许多任务上显示出了惊人的效果，但它们仍可能表现出错误的行为，并导致事故和损失。因此，除了经典的基于准确性的评估之外，能够检测不正确的 DNN 行为并提高 DNN 质量的测试技术是极其必要和关键的。但是，基于 DNN 的系统的测试通常需要大量的人工来标记测试用例的真实信息。在本文中，为了解决这一挑战，我们引入了 DeepGini，该工具可帮助开发人员在没有人工标签的对大量的测试用例进行优先级排序，从而在早期阶段检测出不正确的行为。通过我们的排序，得到可能对系统正确性影响比较大的测试用例，用户只需要对这些关键的测试用例进行标签，从而节省大量的人工标记成本。 DeepGini 在熵理论的启发下，利用网络输出层的统计信息来实现测试数据的优先级排序。我们进行了广泛的实验以评估其效果，包括最新的 DNN 模型和流行的开源数据集，例如 MNIST，CIFAR10，SVHN 和 Fashion-MNIST。结果表明，就覆盖率和效率而言，DeepGini 优于其他基于神经元覆盖的优先排序技术。该演示视频可从 https://youtu.be/GKp5Z8UtqZs 获得。

关键词：深度神经网络，测试用例优先级

1：介绍

深度神经网络（DNN）的巨大进步推动了深度学习应用的出现，这些应用为人类带来了商业利益，并且它们已逐渐部署在更关键的领域，例如医学诊断，自动驾驶汽车和空中交通管制[5、6、17、19]。 它们是基于与传统软件应用程序不同的数据驱动编程模型构造的。 尽管深度学习在一些特定的问题上已经超越了人类，但它仍然存在许多缺陷，并可能导致意外行为，这在某些安全攸关的场景下是无法容忍的，例如由特斯拉开发的自动驾驶汽车造成的事故 [4]和 Uber [7]。

但是，几乎所有现有的研究都只专注于追求高性能基准，而只有很少的工作专注于检测 DNN 的不良行为或提出建议以帮助开发人员提高基于 DNN 的软件质量。从另一个角度看，在基于 DNN 的软件上进行的测试与常规软件有很大的不同，因为前者主要基于数据驱动的编程模型，而后者主要基于业务逻辑。因此，需要大量的测试用例来检测基于 DNN 的系统的不良行为。更糟的是，DNN 经常面临这样一个问题，即自动 oracle 通常不可用。一种解决方案是投入大量的人力来标记测试用例。例如，ImageNet 被认为是世界上最大的视觉识别数据集，由 20,000 多个类别和数百万个带有标签的数据组成，花费了 167 个国家的 4.9 万名员工超过 9 年的时间。此外，通过分析可能导致不正确行为的测试用例，软件开发人员可以从这些分析和诊断中汲取灵感，从而可以更好地提高系统质量[15]。这样的软件开发过程中的场景促使我们提出了一个测试用例优先级排序工具，以提高测试人

员在早期发现有价值的测试用例时的效率。此外，考虑到昂贵的人工标记过程，即使没有测试 oracle，我们的工具也设计为有效。

许多关于测试优先级的研究已经针对常规软件系统提出了[3，15，20]。代码覆盖率是指导这些研究优先级排序的首选指标[1、2]。不幸的是，由于常规软件和基于 DNN 的软件之间的固有差异，基于代码覆盖率的方法并不像预期的那样有效。一方面，常规软件通常具有非常清晰的执行逻辑，并且可以毫不含糊地在流程图中表达，而基于 DNN 的软件的构建在某种程度上就像是构建块。在此过程中没有明确的逻辑流程，也没有明确的数据流程，因此，使基于代码覆盖率的方法不再有效。另一方面，如果我们尝试将代码覆盖率应用于 DNN，则会发现很少的测试用例（约占总测试集的 1％）可以达到测试集的最大覆盖率。另外，给定一个 DNN 模型，每个测试输入都可以达到相同的 top-k 神经元覆盖率[10]，这验证了我们前面提到的假设。结果，当面对基于 DNN 的系统时，基于代码覆盖率的测试优先级排序方法变得毫无意义且不适用。

DeepGini [16]是专门解决上述问题而设计的，通过统计角度将测量错误分类的可能性的问题转换为测量测试集纯度的问题。这种转换实际上遵循了 Gini 杂质[14]的想法，它主要用于测量在数据集上进行随机抽样时被错误分类的概率。凭直觉，如果 DNN 输出的 softmax 为每个类别生成相似的值，则倾向于将测试用例分类错误。根据 Gini 杂质的定义，当 DNN 输出每个类别的概率相同时，我们的方法将输出最大值。与代码覆盖方法相比，我们的方法具有以下优点。第一且最重

要的一点是，我们的方法基于扎实的理论并且可以很好地解释，而基于代码覆盖率的方法只是 DNN 测试领域中的机械应用。其次，我们的方法效率更高，因为我们只需要利用最后一层的信息即可。但是，基于代码覆盖率的方法需要记录每个神经元的信息以计算覆盖率，这既浪费时间又浪费资源。第三，考虑到 DeepGini 仅利用最后一层的输出矢量，我们的方法更像是黑盒测试，并且对网络体系结构的依赖性较小，这对于希望保护 DNN 中嵌入的敏感信息的研究人员具有强烈的吸引力。我们的工具已经在多个流行的开源数据集上进行了评估，并附带了最新的 DNN 模型。

总之，DeepGini 具有以下功能：

•DeepGini 是基于 Web 的测试优先级划分工具，用于帮助开发人员及早发现 DNN 错误行为。 基于 Gini 杂质的理论[14]，DeepGini 提供了一种在大规模测试集中选择容易出错的测试用例的有效方法。

•DeepGini 旨在在不了解 DNN 架构的情况下工作，DNN 架构被认为更安全，更高效。

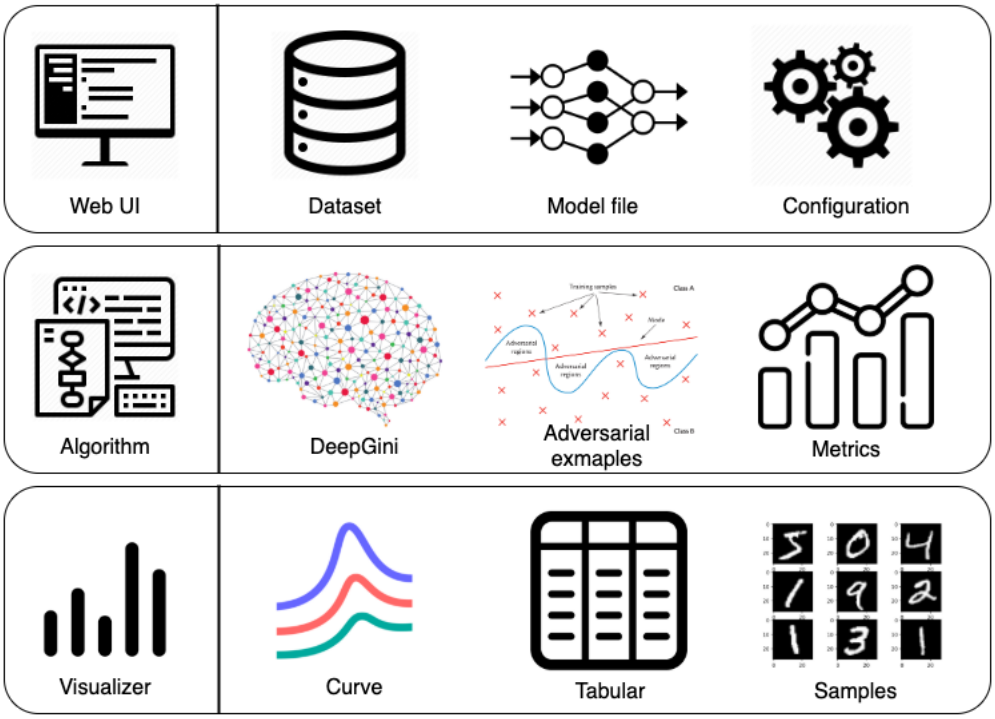
•DeepGini 应用程序具有通过单击相应值来可视化测试用例的功能，这在分析和诊断阶段很有用。

2.工具设计

2.1 架构

我们在图 1 中展示了 DeepGini 的体系结构。由三个部分组成：前

端，由 WebUI 层和可视化层组成，用于处理人机交互并以更易理解的方式可视化最终结果，基于深度学习框架的后端进行实际计算，包括训练模型用户不提供运行 DeepGini 算法和其他基于 coverage 的方法的方法。详细而言，用户可以上传自己的数据集和模型文件，也可以选择我们提供的数据集和模型。然后，我们的后端将运行 DeepGini 算法，根据模型的输出确定数据集的优先级。算法完成后，最终结果将被馈送到可视化层，以便以人类便读的形式显示它。此时，用户可以看到不同百分比的故障检测以及优先测试用例的前 5 个样本，这些样本可以用来调试和诊断神经网络缺陷。我们选择将 DeepGini 实现为基于 Web 的微服务，以便可以在任何平台上部署此软件。另外，与桌面应用程序相比，用户只需要浏览器就可以访问 DeepGini，这可以大大简化环境配置和软件安装过程。



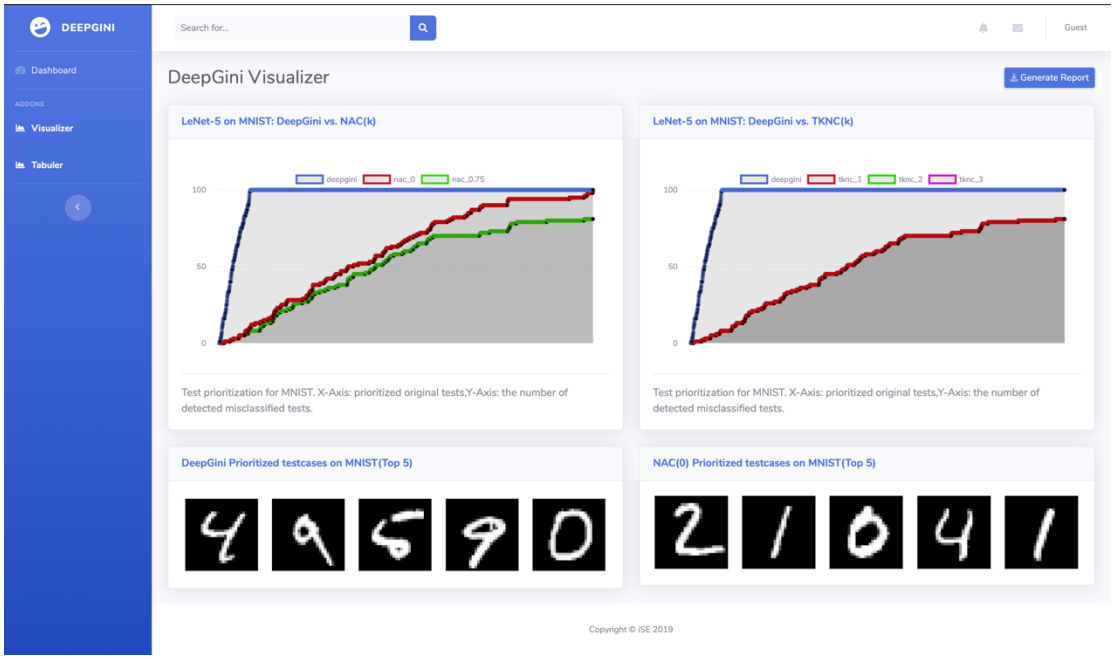
2.2 可视化

为了帮助我们的最终用户更好地了解 DeepGini 的计算结果，我们提供了一个精美的可视化层。 DeepGini 完成处理后，评估结果将显示在网页上，其中包括一个表格，该表格显示了原始数据集和对抗性样本上的 Coverage-Additional Method（CAM）的不同度量结果，以及一些图表来说明 不同的测试优先级排序方法。 此外，用户可以通过单击相应的曲线来可视化通过不同方法确定优先次序的前 5 个样本，这将有助于调试 DNN 模型的开发人员或测试人员。 我们还提供了说明不同覆盖标准性能的图表。 除了图表外，我们还提供了一个表格来详细显示数值结果。 用户可以通过检查表中的数字数据来比较执行时间并了解几个指标之间的细微差别。

2.3 接口和用法

图 2 显示了 Deep-Gini 计算的最终结果分析。我们可以看到几条覆盖曲线，它们代表不同的优先方法。用户可以清楚地区分哪种方法在指定的数据集和 DNN 模型下效果最好。此外，DeepGini 选择了一些优先测试用例的样本，并提供给我们的用户，以便他们了解哪些样本可能会导致当前模型的缺陷。此外，我们的工具还提供了一个选项，可以使用 FGSM，JSMA 和 CW 等最新的对抗性攻击算法生成一些对抗性示例，以利用测试过程。这些样本也将显示给用户。我们还为用户提供了下载按钮和生成报告按钮，以便他们可以方便地检查整个优先数据集并更好地了解其模型的整体性能。通过选择表格页面，在

DeepGini 过程中计算的 CAM 和 CTM 以及执行时间将汇总到一个表中。



2.4 实现

我们基于客户端/服务器架构实现了 DeepGini。考虑到 Tensorflow 提供了强大的 API 支持并且拥有强大的社区。 在我们的实现中，DeepGini 采用 TensorFlow 作为计算引擎。 此外，对于后端，我们采用 Node.js 框架来处理请求，并使用远程过程调用与计算引擎进行通信。 我们的设计和实现使用户能够在分布式计算机上部署计算引擎和后端。 请注意，通过采用远程过程调用作为通信策略，我们将后端与计算引擎分离。 最终用户可以选择其他深度学习引擎来支持计算。

3 评估

我们已经结合其他指标进行了全面的实验，以评估 DeepGini 的性能。 我们的实验中使用的主要指标是故障检测的平均百分比（APFD）和用于衡量有效性和效率的时间成本。 较高的 APFD 值表示更快的错误分类检测率。 当将检测到的误分类测试的百分比与优先测试的数量作图时，可以将 APFD 计算为绘制线下方的面积。 形式上，对于其中有 k 个测试将被错误分类的 n 个测试的排列，令 oi 为揭示第 i 个错误分类测试的第一个测试的顺序。 可以按以下方式计算此排列的 APFD 值：

$$APFD = 1 - \frac{\sum_{i=1}^{k} o_i}{kn} + \frac{1}{2n}$$

为了衡量实验方法的时间成本，我们记录每种方法的执行时间。

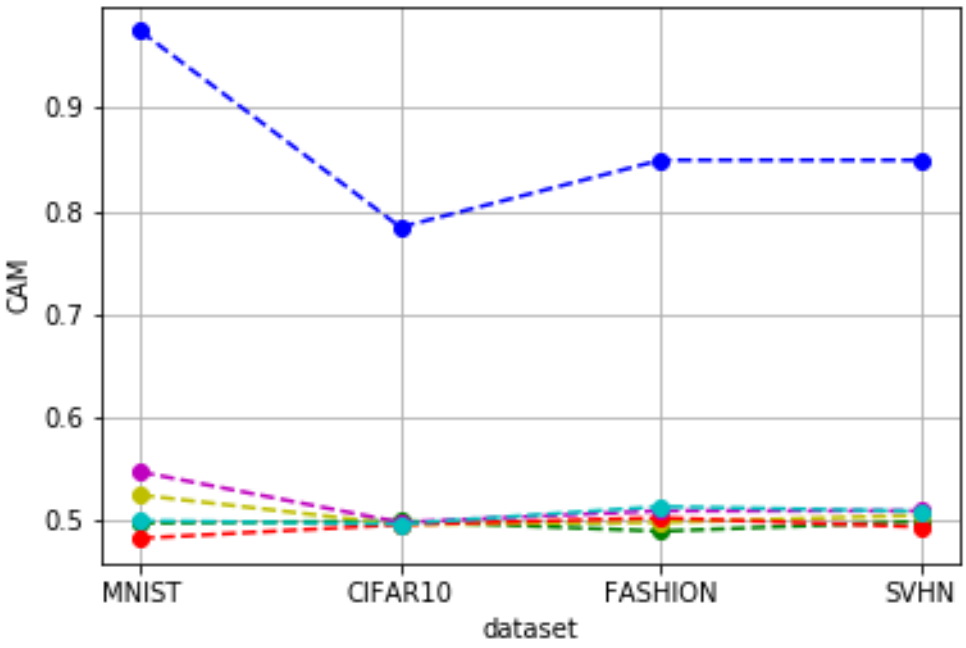
如表 1 所示，我们选择了 3 个流行的公开可用数据集，即 MNIST [9]，CIFAR-10 [8]，SVHN [12]。 在 MNIST 和 CIFAR-10 上，我们使用预先训练的 LeNet-5 和 ResNet-20 作为我们的实验模型。 对于 SVHN，由于我们找不到任何可用的预训练 DNN 模型，因此我们自己训练 DNN 模型。

我们将 DeepGini 与以下几种覆盖标准进行了比较：NAC（k），KMNC（k），TKNC（k）。 图 3 中的结果表明，与基于 NAC（k）的方法相比，无论 k 的值如何，我们都可以通过小于 0.5％的测试来达到最大覆盖率。 从这个图中我们有两个观察结果。 首先，我们的优先排序方法

比基于神经元覆盖的方法可以更快地找到更多错误分类的测试案例。其次，我们发现基于神经元覆盖的优先级有时甚至比随机优先级还要差。

与基于 TKNC（k）的方法相比，我们的方法要好得多，因为我们方法的曲线上升速度要快得多。 我们还发现，无论 k 的值如何，每个测试的覆盖率都为 TKNC（k）。 因此，如果我们使用 TKNC（k）对测试进行优先级排序，则 CTM 无法正常工作。 不幸的是，CAM 也无法使用此覆盖率指标。 主要原因是只有约 1％的测试足以达到最大覆盖率。 在确定 1％的测试的优先级之后，CAM 会退化为 CTM，无法按上述说明运行。 因此，我们只能随机分配其余测试的优先级。 在有效性方面，以 MNIST 为例，图 2 绘制了优先级排序结果，其中我们的方法的曲线上升速度远远快于基于 TKNC（k）的方法。 因此，我们的方法在有效性上要好得多。

如上所述，如果我们使用 KMNC（k）对测试进行优先级排序，则 CTM 将不起作用，因为几乎所有单个测试都具有相同的 KMNC（k）覆盖率，而与 k 的值无关。 因此，我们仅将基于 KMNC（k）的 CAM 与我们的优先级方法进行比较。 基于 KMNC（k）的 CAM 方法的有效性并不吸引人。 以 MNIST 为例，图 2 表明，我们的方法的曲线上升速度远远快于基于 KMNC（k）的方法。



5 结论

在本文中，我们介绍了 DeepGini，这是一个基于 Web 的工具，用于测量错误分类的可能性，可用于对测试进行优先级排序，以便降低人工标签的成本，只需要对一些关键的用例进行标签，再去训练神经网络，就可以大幅提高系统的性能。 评估结果表明，DeepGini 比基于神经元覆盖的方法更为有效，可以更快速的找出可能使系统出错的测试用例。 此外，DeepGini 提供了生动而简单的人机界面，以帮助最终用户全面了解优先测试用例，这对于快速查找代码缺陷非常有用。