南京大學

研 究 生 毕 业 论 文

（申请工程硕士学位）

论 文 题 目 基于变异图像熵值引导的神经网络模糊测试技术

作 者 姓 名 柯轶东

学科、专业名称 工程硕士（软件工程领域）

研 究 方 向 软件工程

指 导 教 师 陈振宇 教授

2022 年 **5** 月 **20** 日

学　　号：**MF20320074**

论文答辩日期：**2022** 年 **5** 月 **20** 日

指 导 教 师：　　（签字）

Fuzzing Deep Neural Networks Based on Image Mutant Entropy Guidance

by

Yidong Ke

Supervised by

Professor Zhenyu Chen

A dissertation submitted to
the graduate school of Nanjing University
in partial fulfilment of the requirements for the degree of
Master of Engineering
in
Software Engineering

Software Institute
Nanjing University

May 23, 2022

南京大学研究生毕业论文中文摘要首页用纸

毕业论文题目：基于变异图像熵值引导的神经网络模糊测试技术

工程硕士（软件工程领域）专业 2020 级硕士生姓名：柯轶东

指导教师（姓名、职称）：陈振宇 教授

摘　要

近年来，随着深度学习系统的不断发展，深度神经网络测试的重要性开始引起人们的注意。为了对深度学习系统进行有效的测试，学术界提出了基于神经元覆盖率引导的模糊测试技术。然而神经元覆盖率指标需要人工设定阈值，不同的阈值可能会对测试结果的稳定性造成影响。因此，需要一种更加有效的指标来对神经网络的模糊测试进行引导，从而提高模糊测试的效率，增强深度学习系统的健壮性。

本文首次提出了基于变异图像熵值引导的神经网络模糊测试技术。熵可以度量一个事件的自信息量，如果大多数测试生成的输入都触发了以前未有的程序行为，那么输出的熵值就会很高，测试的效率也就更高。本文使用熵值作为权重分配指标来引导变异图像的生成，通过最大化熵值来最大化模糊测试的效率，以揭示更多关于神经网络的信息。同时，本文使用基于蜕变关系的图像变异方法来保留变异图像的语义。最后本文结合了工业表面缺陷检测的领域知识，将熵值引导的图像变异方法应用到工业缺陷图像数据扩增上，并使用工业缺陷数据集来验证本文方法的实际应用效果。

本文在 MNIST、CIFAR10 两个流行公开可用数据集上与基于覆盖率引导的神经网络模糊测试框架 DeepHunter 进行了对比实验，并在工业缺陷数据集 NEU-DET 上使用变异图像熵值引导的图像扩增方法进行了图像数据扩增，使用 Yolov5 模型进行了训练与测试验证。实验结果表明，与 DeepHunter 相比，本文的基于变异图像熵值引导的神经网络模糊测试方法能够更加有效地提高神经元覆盖率。同时，将熵值引导的图像变异方法应用于工业表面缺陷检测可以有效地提高缺陷检测的准确率，具有一定的实际应用价值。

关键词： 深度学习，模糊测试，熵值引导，图像扩增，缺陷检测

南京大学研究生毕业论文英文摘要首页用纸

THESIS: Fuzzing Deep Neural Networks Based on Image Mutant Entropy Guidance

SPECIALIZATION: Software Engineering

POSTGRADUATE: Yidong Ke

MENTOR: Professor Zhenyu Chen

Abstract

In recent years, with the development of deep learning systems, the importance of testing deep neural networks began to attract people's attention. In order to test the deep learning system effectively, academia has proposed coverage-guided fuzz testing for deep neural networks. However, the threshold of neurons' activation needs to be set manually, and different thresholds may affect the stability of test results. Therefore, a more effective metric is needed to guide the fuzz testing of neural networks, so as to improve the efficiency of fuzz testing and enhance the robustness of deep learning systems.

This thesis first proposes a method of fuzzing deep neural networks based on image mutant entropy guidance. Entropy can measure the amount of self-information about an event. If most fuzzer-generated inputs exercise previously unseen program behaviors, then the entropy will be high and the fuzzer will perform much better at discovering new behaviors. Entropy is used as the weight scheduling metric to guide the generation of mutated images and the test efficiency is promoted by maximizing the entropy value to reveal more information about the neural networks. Besides, this thesis uses an image mutation method based on metamorphic relation to preserve the semantics of mutated images. Finally, this thesis combines the domain knowledge of industrial surface defect detection, applies the entropy-guided image mutation method to the augmentation of industrial defect image data, and uses the industrial defect dataset to verify the practical application effect of this method.

This thesis conducts a comparative experiment on two popular publicly available data sets, MNIST and cifar10, with DeepHunter, a coverage-guided neural network

fuzzing framework, and performs image data augmentation on the industrial surface defect data set NEU-DET using a mutant image entropy-guided image augmentation method, using the Yolov5 model for training and test validation. The experimental results show that, compared with DeepHunter, the method proposed by this thesis can improve the neuron coverage more effectively. Besides, applying the entropy-guided image mutation method to industrial surface defect detection can effectively improve the accuracy of defect detection and is certainly of practical application value.

keywords: Deep Learning, Fuzz Testing, Entropy Guidance, Image Augmentation, Defect Detection

目　　录

目　　录 …… v
插图清单 …… vii
附表清单 …… ix
第一章　引言 …… 1
1.1　研究背景及意义 …… 1
1.2　深度神经网络测试国内外研究现状 …… 3
1.3　论文研究内容及组织结构 …… 5
第二章　相关技术概述 …… 7
2.1　深度神经网络与卷积神经网络 …… 7
2.1.1　深度神经网络 …… 7
2.1.2　卷积神经网络 …… 8
2.2　深度神经网络测试 …… 10
2.3　蜕变测试 …… 13
2.4　数字图像处理技术 …… 14
2.4.1　图像的基本表示方法 …… 14
2.4.2　图像的几何变换 …… 15
2.5　信息熵 …… 17
2.6　本章小结 …… 18
第三章　基于变异图像熵值引导的神经网络模糊测试方法 …… 19
3.1　技术路线与研究内容 …… 19
3.2　基于蜕变关系的图像变异方法 …… 20
3.3　基于熵值引导的灰盒模糊测试 …… 26
3.3.1　基于黑盒模糊测试的概率框架 …… 26
3.3.2　模糊器效率的信息论度量 …… 29
3.3.3　熵值引导的灰盒模糊测试的具体实现 …… 32
第四章　熵值引导的模糊测试方法在工业缺陷图像扩增上的应用 …… 35
4.1　工业表面缺陷检测 …… 35

4.2 基于工业表面缺陷特征的图像数据扩增方法 ····· 37
4.2.1 工业图像采集过程模拟扩增方法 ····· 37
4.2.2 工业表面缺陷形态特征模拟扩增方法 ····· 42
第五章 实验设计与分析 ····· **47**
5.1 实验一：神经元覆盖率水平评估 ····· 47
5.1.1 实验目的 ····· 47
5.1.2 实验对象 ····· 47
5.1.3 实验指标 ····· 49
5.1.4 实验设计 ····· 49
5.1.5 实验结果与分析 ····· 50
5.2 实验二：熵值引导的工业表面缺陷图像扩增效果评估 ····· 52
5.2.1 实验目的 ····· 52
5.2.2 实验对象 ····· 53
5.2.3 实验准备 ····· 54
5.2.4 实验指标 ····· 54
5.2.5 实验设计 ····· 56
5.2.6 实验结果与分析 ····· 56
5.3 本章小结 ····· 58
第六章 总结与展望 ····· **59**
6.1 本文研究内容总结 ····· 59
6.2 本文工作内容局限与展望 ····· 60
致 谢 ····· **61**
参考文献 ····· **63**

插图清单

1-1 传统软件测试与深度学习软件测试 …… 2
1-2 DNN 结构测试标准之间的关系 …… 4

2-1 神经元计算示意图 …… 7
2-2 常用的激活函数图像 …… 8
2-3 卷积神经网络基本结构图 …… 8
2-4 卷积操作示意图 …… 9
2-5 常用的池化操作示意图 …… 10
2-6 全连接层网络结构示意图 …… 11

3-1 技术路线示意图 …… 19
3-2 图像变异示意图 …… 20
3-3 像素值变换示意图 …… 21
3-4 仿射变换示意图 …… 22
3-5 图像感知哈希的映射关系 …… 23
3-6 传统软件程序的分支层次关系 …… 26
3-7 物种发现曲线 …… 27
3-8 放回抽样示意图 …… 29
3-9 模糊器的熵估计 …… 31

4-1 缺陷检测的问题定义 …… 36
4-2 RGB 彩色立方体示意图 …… 38
4-3 线性插值法示意图 …… 41
4-4 双线性插值法示意图 …… 41

5-1 MNIST 数据集示例图 …… 48
5-2 CIFAR-10 数据集示例图 …… 48
5-3 不同模型的覆盖率变化趋势 …… 51
5-4 NEU-DET 数据集示例图 …… 53

5-5 Yolo 格式标签示意图 55
5-6 数据集扩增前后模型训练结果对比 57

附表清单

5-1 实验数据集和模型神经元数量统计表 …… 48
5-2 不同配置的模糊测试在 10 次运行中的平均覆盖率（%） …… 52
5-3 Yolov5s 模型扩增前后的覆盖率（%） …… 58

第一章　引言

1.1　研究背景及意义

近年来，深度神经网络（DNN）已经被广泛应用于数据挖掘，自然语言处理，语音识别，图像处理等前沿领域。这些应用给人们的生活带来了巨大的改变，提高了生产效率，让人们看到了更多未来世界的可能性。基于 DNN 的软件虽然和传统软件一样通过了完整的测试流程，仍然可能存在着会导致严重安全问题的隐藏缺陷，比如被报道过的特斯拉自动驾驶事故[①]。这自然引出了一个问题，即应用 DNN 的软件应该如何进行测试、验证和最终认证，以满足相关安全标准 [1] 的要求。深度学习系统的质量和安全保障的重要性开始引起人们的注意，特别是那些应用于安全关键任务场景的。

与传统软件一样，基于 DNN 的软件也需要在部署前进行系统测试。通过对 DNN 模型进行系统测试，可以在早期阶段识别其可能隐藏的缺陷，从而减少损失和排除安全隐患。同时，检测出的各种缺陷可以为开发人员提供反馈，以深入分析导致缺陷产生的原因，增强系统的鲁棒性。

对于传统软件来说，软件测试已经是一种成熟且常用的对软件质量进行定量分析的技术。为了更加详细和全面地对被测程序的执行状态进行分析和研究，学者们提出了很多不同粒度级别的覆盖标准。在代码级别的粒度上，有语句覆盖、分支覆盖、数据流覆盖 [2] 和突变测试 [3]。其中，分支覆盖追踪程序逻辑结构中的每一个控制分支是否被覆盖到了，而语句覆盖则追踪目标程序中的每个可执行语句是否被执行到了。数据流覆盖关注的是每个变量在程序路径上的变化情况。在模型级别的粒度上，有基于转换的覆盖率 [4] 和状态覆盖率 [5]。基于模型的覆盖标准通过使用抽象的模型来更全面地分析程序行为。以上的各种覆盖准则能够适应不同的测试方法和粒度级别 [6]。

覆盖率引导的模糊测试技术（CGF）可以用于发现真实软件中存在的漏洞 [7]。在传统计算机程序中，两个最流行的覆盖率引导的模糊器是 AFL[②]和

[①] https://electrek.co/2016/07/01/understanding-fatal-tesla-accident-autopilot-nhtsa-probe/.
[②] https://github.com/google/AFL

libFuzzer [①]。在覆盖率引导的模糊测试中，一个模糊处理过程维护一个包含被测程序输入的语料库，模糊器根据变异算子对这些输入进行随机变异，变异后的输入在进行新的覆盖率计算后被保留在语料库中 [8]。通过这种方式，如果一个新的输入导致代码在 if 语句上执行到了不同的分支，那么覆盖率就会增加 [9]。CGF 在提高覆盖率以及检测软件缺陷方面比随机测试更加高效。

CGF 在识别传统软件的缺陷上有着良好的表现，所以人们会很自然地联想到它是否可以应用于 DNN。传统的覆盖率度量跟踪那些已经执行的代码行和分支。而在 DNN 中，模型由一层层的神经层（卷积层、全连接层等）构建而成，每个神经层的组成元素包括神经元、神经元的值、神经元之间的连接、权重以及激活函数。这些元素的底层实现可能包含大量分支语句，但是其中大部分只与神经网络的结构有关，并不会因为输入的变化而执行不同的分支。对于不同的输入，神经网络很有可能执行了相同的代码分支，但仍然产生了不同的行为差异。因此，传统软件与基于 DNN 的软件在编程范式和开发过程中存在着本质性的区别，无法直接将传统软件的测试技术应用于基于 DNN 的软件，如图 1-1所示。这使得对基于 DNN 的软件的测试成为一个新的挑战。

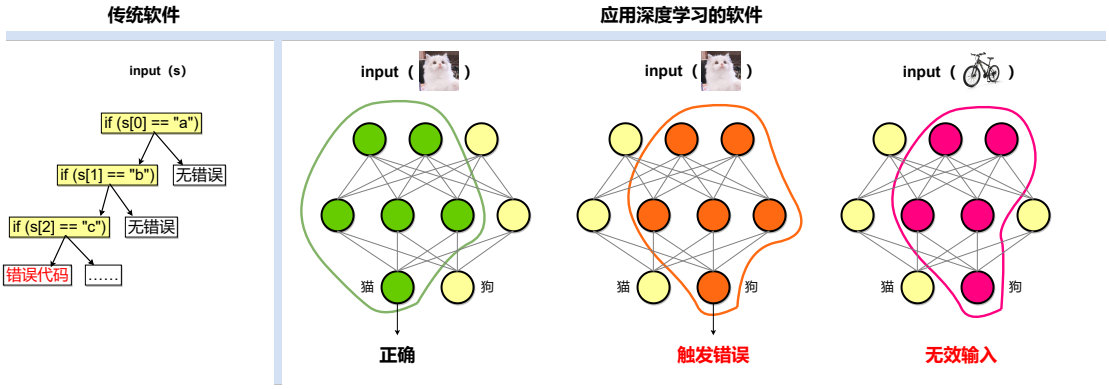


图 1-1: 传统软件测试与深度学习软件测试

为了更好地评估 DNN 测试的充分性，需要全新的覆盖标准。Huang 等人参考 MC/DC 测试方法 [10]，将 MC/DC 方法应用到了深度神经网络测试中，提出了四种 MC/DC 思想的覆盖准则 [11]。Ma 等人认为不能仅仅依靠深度学习系统的输出来对其质量进行评估，并从功能区、边界区和层级三个不同的角度提出了一系列的 DNN 测试标准 [12]，之后又将组合测试的方法引入了 DNN 测试中并制定了两种覆盖准则 [6, 13]。

然而，神经元覆盖率指标需要人工设定阈值，而不同层的神经元以及同层

[①]https://llvm.org/docs/LibFuzzer.html

的不同神经元的输出值在统计分布上都有一定的差异。在同一组测试集中，不同层或者不同区域的神经元的输出可能具有不同的平均值和方差，人工设定的阈值可能会对测试结果造成一定的影响。因此，如果使用神经元覆盖率来引导模糊测试，可能无法保证测试效果的稳定性，需要一种更加有效的指标来对神经网络的模糊测试进行引导，从而提高模糊测试的效率，增强深度学习系统的健壮性。

1.2 深度神经网络测试国内外研究现状

深度学习系统已经在众多方面得到了广泛的应用，包括自动驾驶、语音识别、图像处理、人脸识别、自然语言处理等等。然而，尽管深度学习系统已经展现出其强大的能力，但由于一些原因，如过度拟合和欠拟合、有偏差的训练数据等，可能会在极端情况下表现出预期之外的错误行为。在安全关键领域中，这些错误行为可能会引发严重的安全事故。例如近年发生的一起特斯拉自动驾驶事故，事故原因是特斯拉的软件系统未能在明亮的天空下识别出白色的拖车[①]。在攸关性命的安全关键领域，我们不允许 DNN 做出错误或者超出预期的行为，因此需要对其进行充分的测试。

Pei 等人设计了 DeepXplore [14]，这是第一个用于大规模深度学习系统的白盒测试框架。DeepXplore 首次提出了神经元覆盖率的概念，即通过一组输入所激活（神经元的值高于给定阈值）的神经元数量来衡量深度学习系统的逻辑，这也是针对深度学习系统测试提出的第一个测试充分性准则。DeepXplore 还支持用户添加自定义约束，以模拟不同类型的真实输入。实验表明，DeepXplore 生成的测试数据也可以用来重新训练相应的深度学习系统，与对相同数量的随机输入或对抗输入进行再训练相比，将分类准确度提高了 3%。

Ma 等人提出了 DeepGauge [12]，一套多粒度的深度学习系统测试标准。他们将 DNN 的神经元分布划分为了主功能区域和可能会产生缺陷的边界区域。给定一组输入，DeepGauge 的标准可以度量它对主功能区域和边界区域的神经元的覆盖程度。他们的实验结果表明，目前的大部分测试数据对主功能区域的覆盖程度远大于边界区域。DeepGauge 的标准有着更高的覆盖范围，可以更有效地检测 DNN 的缺陷。DeepGauge 为构建更通用、更健壮的深度学习系统提供了启示。

[①] https://electrek.co/2016/07/01/understanding-fatal-tesla-accident-autopilot-nhtsa-probe/.

Tian 等人设计、实现和评估了 DeepTest [15]，用于检测基于 DNN 的自动驾驶软件的错误行为。DeepTest 利用神经元覆盖率作为引导来生成能最大化激活神经元数量的测试输入，对不同类型的驾驶行为进行了全面的探索，检测其中的错误行为。他们使用 DeepTest 测试了 Udacity 驾驶挑战赛中表现最好的三个 DNN 模型，检测到了数千个错误行为，其中许多都有引发致命车祸的可能性。

Huang 等人受 MC/DC 覆盖准则 [10] 的启发，提出了一组根据 DNN 的结构特征和语义量身定制的测试标准 [11]。他们验证了这些标准，证明通过他们提出的覆盖率标准所生成的测试输入能够捕获 DNN 中期望之外的行为。图 1–2给出了 DNN 现有结构测试覆盖标准之间关系的示意图总结，箭头代表“弱于”的关系。通过与现有的方法进行比较，Huang 等人表明他们的标准在错误检测效率和生成测试用例的计算成本之间取得了平衡。他们的实验是在最先进的 DNN 上进行的，使用流行的开源数据集，包括 MNIST、CIFAR-10 和 ImageNet。他们用自己的测试覆盖标准对不同规模（从几百到数百万数量的神经元）的 DNN 进行实验，证明了它们在错误检测、DNN 安全统计、测试效率、DNN 内部结构分析四个方面的实用性。他们的标准是第一个能够捕获和量化 DNN 中存在的因果关系的工作，这些因果关系对理解神经网络的行为至关重要。

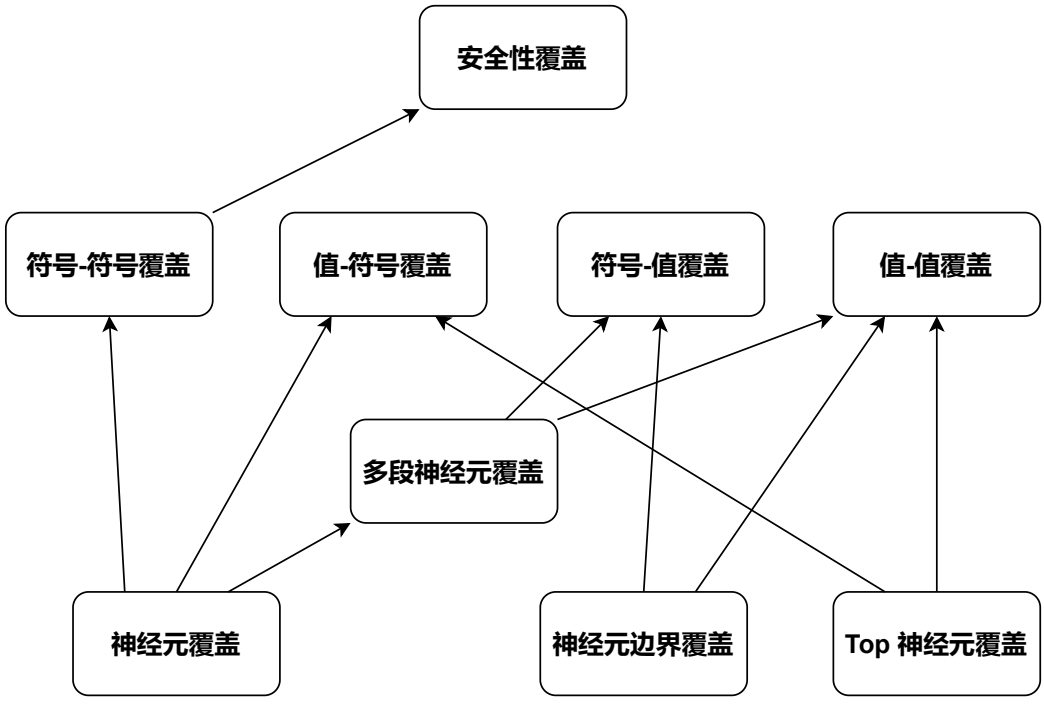


图 1–2: DNN 结构测试标准之间的关系

Xie 等人提出了 DeepHunter [16]，一个覆盖率引导的 DNN 模糊测试框架。他们首先提出了一种能够在很大程度上保留种子语义的种子变异策略，以及使用一种基于种子模糊处理次数的种子选择策略。同时，他们使用了多个可扩展的覆盖标准作为引导机制来生成新的测试用例。大规模实验表明，DeepHunter 的种子变异策略可以生成保留语义的新测试用例，其有效性高达 98%；DeepHunter 在缺陷识别的覆盖范围、数量和多样性方面均优于目前的技术水平。

1.3 论文研究内容及组织结构

本文在对深度神经网络的模糊测试中引入了信息熵作为权重分配指标来引导变异种子的生成，通过最大化熵值来提高模糊测试的效率，以揭示更多关于 DNN 的信息。同时，本文结合工业表面缺陷检测的领域知识，使用工业缺陷数据集来验证评估本文方法的效果。本文的主要结构如下：

第一章为引言，主要介绍了本文的研究背景和研究意义以及深度神经网络测试的发展现状和存在的问题，并且简单叙述了本文的主要研究工作，最后介绍了本文的内容与组织结构。

第二章为相关技术概述，主要介绍了本文所涉及的技术以及相关的理论知识。主要内容包括深度神经网络测试以及相关覆盖标准，图像处理的相关技术和原理以及信息熵相关的理论知识。

第三章主要介绍了基于变异图像熵值引导的神经网络模糊测试技术。本章首先介绍了基于蜕变关系的图像变异方法来对图像种子进行变异，接着介绍了基于熵值引导的模糊测试框架，将种子按照熵值大小来进行权重分配从而提高测试效率。最后介绍了基于工业表面缺陷特征的图像数据扩增方法，用于验证本文方法的有效性。

第四章介绍了基于工业表面缺陷特征的图像扩增方法，是为了将本文方法应用于实际场景，验证评估本文方法的实际应用效果。

第五章分为两个实验，实验一将本文方法与基于覆盖率引导的神经网络模糊测试框架 DeepHunter 进行了对比实验，实验二结合基于工业表面缺陷特征的图像扩增方法对本文方法的实际应用效果进行了验证。

第六章对本文的工作进行了总结与分析，并提出了论文中存在的不足和局限性以及对未来工作的展望。

第二章　相关技术概述

2.1　深度神经网络与卷积神经网络

2.1.1　深度神经网络

深度神经网络（Deep Neural Networks，DNN）是深度学习中最基础的网络模型，各种网络都是以此为基础衍生而来。深度神经网络的核心是多层感知机（Multilayer Perception，MP）和反向传播（Backpropagation，BP）。多层网络的设计使得深度神经网络可以更好地对复杂的非线性多维函数进行拟合，相比于单层神经网络能够解决更多复杂问题。

神经元（Neuron）是 DNN 结构的基本组成单元。McCulloch 等人通过模拟动物的神经细胞提出了经典的“M-P 神经元模型” [17]。DNN 的神经元相当于一个带权重的函数，上一层的神经元将数据作为入参传递给该神经元执行计算，并将计算结果返回给下一层的神经元。经过多层神经元计算后，就可以提取到高抽象层次的数据特征。神经元的计算过程如图 2-1所示：

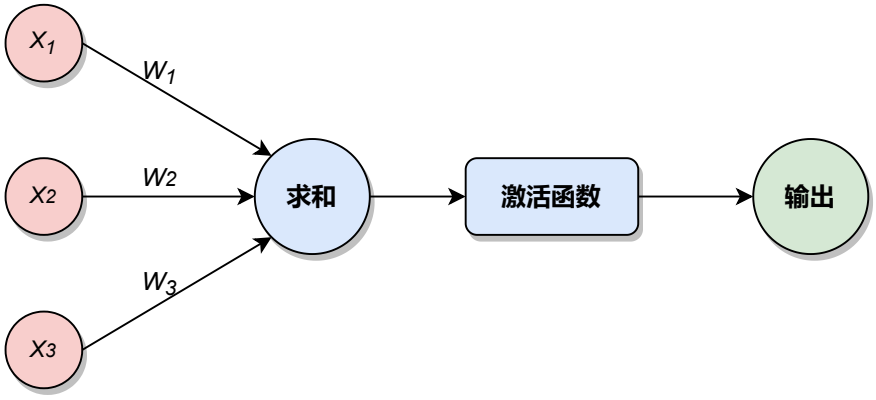
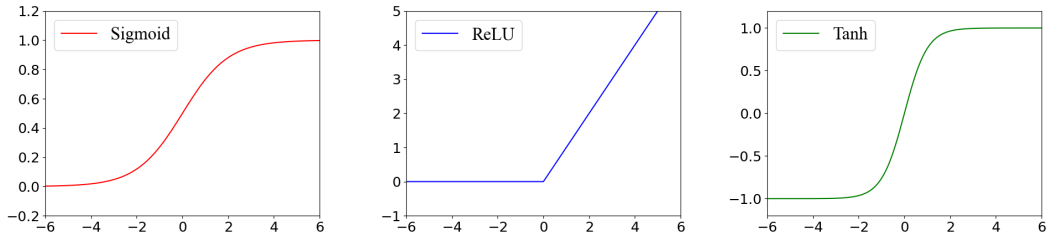


图 2-1: 神经元计算示意图

每个神经元的计算过程可以表示为 $y = \sigma(\sum_i w_i x_i + b)$。其中 x_i 代表前一层神经元的输出值，w_i 代表输出值 x_i 所对应的权重，b 表示偏置项，那么该神经元的计算结果就可以表示为 $\sum_i w_i x_i + b$。为了更好地对非线性函数进行拟合，还需要在网络中引入非线性的激活函数来弥补线性加权求和的局限性。将每层

神经元的计算结果输入到非线性的激活函数 σ 中，最后得到对原输入的非线性映射，从而能够更好地逼近其他函数 [18]。常用的激活函数有（a）Sigmoid 函数，（b）ReLU 函数，（c）Tanh 函数，如图 2-2所示。



(a) Sigmoid 函数 (b) ReLU 函数 (c) Tanh 函数

图 2-2: 常用的激活函数图像

2.1.2 卷积神经网络

在传统 DNN 的结构中，每两个相邻层之间的神经元都是连通的，如果直接将 DNN 应用于处理图像数据，可能会使模型中的参数数量爆炸增长，导致过拟合而学习到一些无法泛化到测试集的训练集属性。为了避免参数膨胀和产生过拟合，提高网络的泛化能力，LeCun 提出的卷积神经网络（Convolutional Neural Networks，CNN） [19] 引入了卷积（Convolution）层和池化（Pooling）层。此后，CNN 开始被广泛应用于解决目标检测和图像分类等各种问题。CNN 的基本结构如图 2-3所示, 下面详细介绍 CNN 的主要结构。

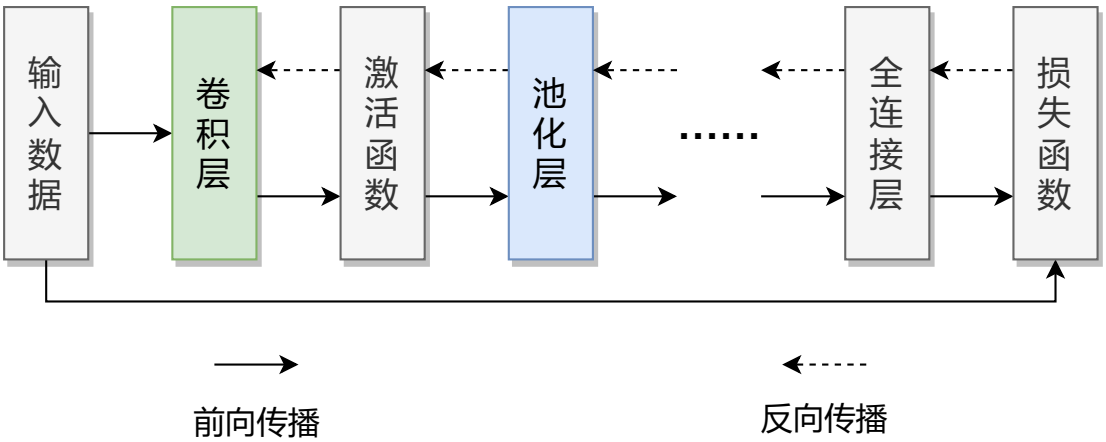


图 2-3: 卷积神经网络基本结构图

（1）卷积层

卷积层是 CNN 的核心组成部分，它的主要功能是提取图片的特征信息。CNN 通过引入卷积核实现了稀疏交互，减少了神经网络层之间的连接数。卷积核即滤波器（Filter），用于提取图像的特征信息。一般而言，图像中的物体特征之间的相关性只局限于该图像的一个子区域内。如果网络结构只采用全连接的方式进行数据整合，那么整个图像中的所有像素间的弱相关性特征都会被引入，这对于 DNN 的目标任务而言是毫无意义的。如图 2-4所示，在卷积操作中，卷积核只与其在图像矩阵中所映射的局部区域进行卷积运算，然后继续滑动给定的步长重复上述运算过程。所有卷积操作完成后即可得到具有高语义特征的卷积特征图（Feature Map）。

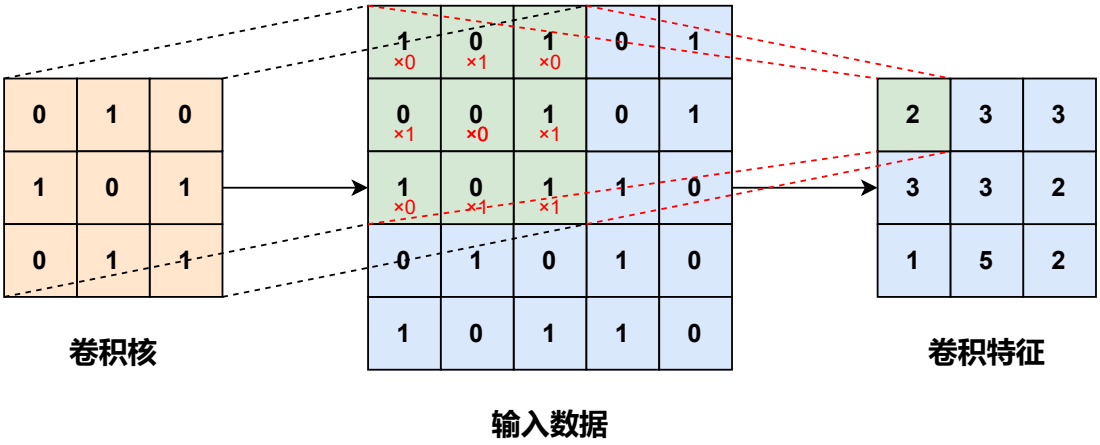


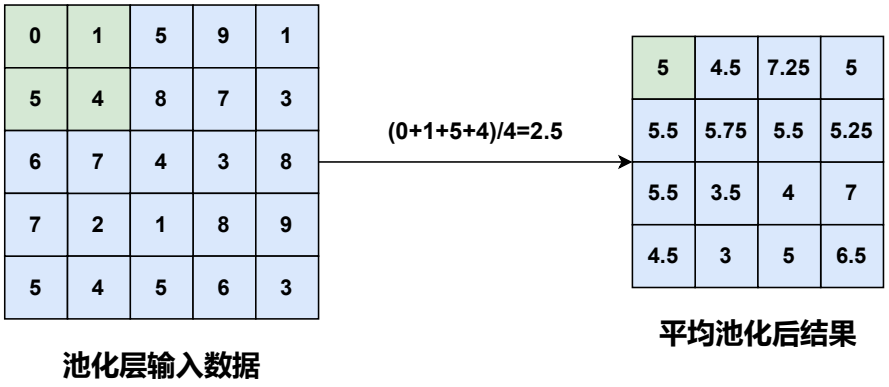
图 2-4: 卷积操作示意图

（2）池化层

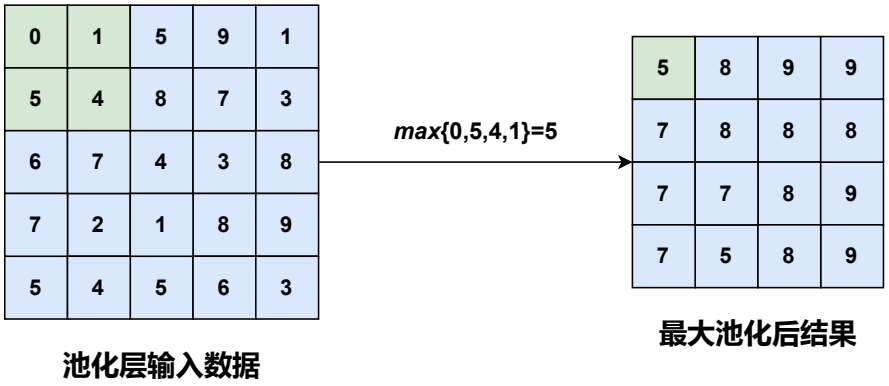
池化层会对卷积特征图进行降维，减少参与卷积运算的参数数量，降低时间复杂度。常用的池化操作有两种，如图 2-5所示。平均池化运算选取待池化窗口中的所有元素的平均值作为池化结果矩阵的子元素，而最大池化运算选取待池化窗口中所有元素的最大值作为池化结果矩阵的子元素，然后滑动池化窗口重复上述操作，最终得到池化结果矩阵。

（3）全连接层

全连接层可以整合卷积层和池化层所提取到的特征信息，其结构如图 2-6所示。在全连接层中，每一个神经元都会与相邻层的每个神经元相互连接。全连接层的计算过程如下所示，用前一层的输出 x_i 乘以权重系数 W_{ij}，然后加上偏置系数 b。



(a) 平均池化



(b) 最大池化

图 2-5: 常用的池化操作示意图

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} W_{11} & W_{12} & W_{13} \\ W_{21} & W_{22} & W_{23} \\ W_{31} & W_{32} & W_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

其中，W_{ij} 为每个节点的权重，x_1，x_2，x_3 分别表示各层的输入，b_1，b_2，b_3 分别表示各层的偏置。

2.2 深度神经网络测试

软件开发方法 [20, 21] 经过了几十年的经验积累和实践，在传统软件中已经十分成熟。然而，与传统软件不同的是，深度学习定义了一种新的数据驱动编程范式，并将其产物保持在编码 DNN 模型的形式 [22]。这些特点给基于 DNN 的软件的质量保证带来了新的挑战。传统软件的决策逻辑是由开发人员直接在源代码中所编写的。然而 DL 开发人员的主要工作是收集有代表性的数据

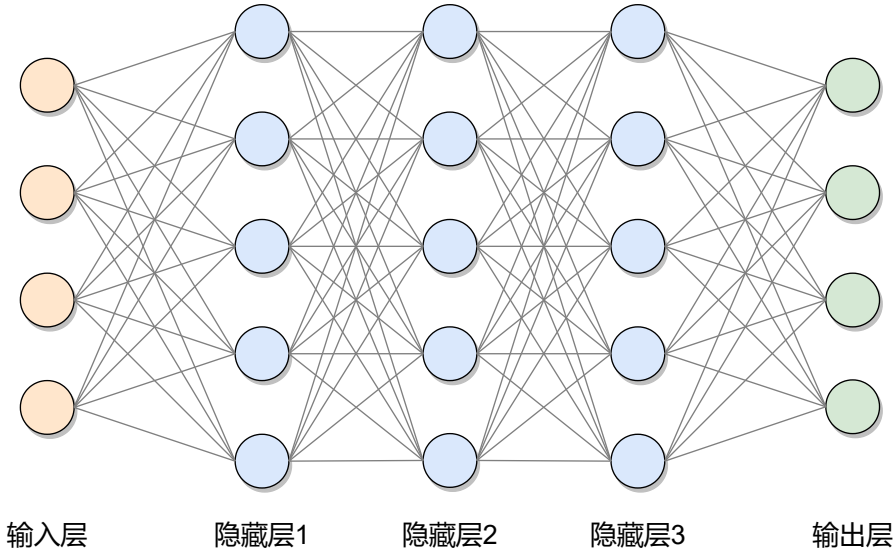


图 2-6: 全连接层网络结构示意图

和标签，设计 DNN 架构，并实现训练 DNN 模型的程序，然后通过模型训练自动形成 DNN 决策逻辑。一旦建立了一个适用的 DNN 模型，它通常会经历一个定制阶段，以满足终端用户平台特定的软件和硬件约束条件，如自动驾驶汽车和智能手机。量化可以降低深度学习模型的精度，从而提高计算效率，减少内存消耗和存储大小。业界已经对模型量化进行了广泛地研究，在大型模型从云端迁移到移动物联网设备的场景中，模型量化也成为了满足极端内存和计算需求的一种常见做法 [16]。

在传统软件中，测试标准被广泛应用于评估测试质量，并指导缺陷检测的测试生成。由于传统软件与基于 DNN 的软件在内部逻辑上存在着本质的区别，因此无法将传统软件测试中的覆盖准则直接应用到 DNN 测试中。近年来，学术界提出了一些专门针对 DNN 的测试标准，这些测试标准的设计考虑了 DNN 的结构，在不同粒度上监测神经元的活动以及内部网络之间的连接。下面介绍一些当前学术界主流的 DNN 测试标准。

设 $N = \{n_1, n_2, \ldots\}$ 是 DNN 的一组神经元，$T = \{x_1, x_2, \ldots\}$ 是测试输入的集合。我们使用 $\phi(\boldsymbol{x}, n)$ 表示一个函数，该函数返回在给定的测试输入 $\boldsymbol{x} \in T$ 下 $n \in N$ 的输出。设激活神经元的阈值为 t，DNN 总共有 l 层，L_i 表示第 i 层的神经元集合 $(1 \leq i \leq l)$。

神经元覆盖率（Neuron Coverage，NCov）：给定一个神经元 n，如果神经元 n 在测试输入 $\boldsymbol{x}$ 下是激活的，那么该神经元就是激活神经元。神经元覆盖

率为激活神经元占总神经元的占比。该标准可以被定义为如下：

$$NCov(T, \boldsymbol{x}) = \frac{|\{\exists \boldsymbol{x} \in T : \phi(\boldsymbol{x}, n) > t\}|}{|N|} \tag{2-1}$$

在神经元层面，通过对测试数据集进行训练可以得到神经元的输出分布，以此大致划分出 DNN 的主功能区域（Major Function Region）和边界区域（Corner-case Region）。对于神经元 n，设 $high_n$ 和 low_n 分别为其激活函数取值范围的上界和下界，$high_n$ 和 low_n 由训练数据集分析所得到 [12]。对于给定的测试输入 $\boldsymbol{x} \in T$，当且仅当 $\forall n \in N : \phi(\boldsymbol{x}, n) \in [low_n, high_n]$ 时，我们称 DNN 位于其主功能区域；当且仅当 $\exists n \in N : \phi(\boldsymbol{x}, n) \in (-\infty, low_n) \cup (high_n, +\infty)$ 时，我们称 DNN 位于其边界区域。

k-多段神经元覆盖率（k-multisection Neuron Coverage，KMNCov）：给定一个神经元 n，k-多段神经元覆盖率衡量测试输入集合 T 覆盖主功能区域 $[low_n, high_n]$ 的程度。为了量化这一点，我们将主功能区域 $[low_n, high_n]$ 划分为 k 个相等的部分 (即 k 段，$k > 0$)，并用 $S_i^n (1 \le i \le k)$ 来表示主功能区域的第 i 段。如果 $\phi(\boldsymbol{x}, n) \in S_i^n$，我们称第 i 部分被测试输入 $\boldsymbol{x}$ 所覆盖。该标准可以被定义为如下：

$$KMNCov(T, k) = \frac{\sum_{n \in N} |\{S_i^n | \exists \boldsymbol{x} \in T : \phi(\boldsymbol{x}, n) \in S_i^n\}|}{k \times |N|} \tag{2-2}$$

为了覆盖边界区域，Ma 等人定义了两个覆盖准则，即神经元边界覆盖和强神经元激活覆盖 [12]。给定测试输入 $\boldsymbol{x}$，如果 $\phi(\boldsymbol{x}, n) \in (-\infty, low_n) \cup (high_n, +\infty)$，我们称输入入 $\boldsymbol{x}$ 覆盖了相应的边界区域。为了量化它，我们首先定义被覆盖的边界区域的数量，如下所示:

$$UpperCornerNeuron = \{n \in N | \exists \boldsymbol{x} \in T : \phi(\boldsymbol{x}, n) \in (high_n, +\infty)\} \tag{2-3}$$

$$LowerCornerNeuron = \{n \in N | \exists \boldsymbol{x} \in T : \phi(\boldsymbol{x}, n) \in (-\infty, low_n)\} \tag{2-4}$$

神经元边界覆盖率（Neuron Boundary Coverage，NBCov）：神经元边界覆盖率衡量给定的测试输入集 T 覆盖了多少边界区域（包括上边界和下边界）。该准则可以被定义如下：

$$NBCov(T) = \frac{|UpperCornerNeuron| + |LowerCornerNeuron|}{2 \times |N|} \tag{2-5}$$

强激活神经元覆盖率（Strong Neuron Activation Coverage，SNACov）：

强神经元激活覆盖率衡量给定的测试输入 T 覆盖了多少上边界区域。该标准可以被定义为如下：

$$SNACov(T) = \frac{|UpperCornerNeuron|}{|N|} \tag{2-6}$$

在神经网络层的层面上，我们使用最活跃的神经元及其组合（或序列）来描述 DNN 的行为。对于给定的测试输入 $\boldsymbol{x}$ 和同一层上的神经元 n_1 和 n_2，如果 $\phi(\boldsymbol{x}, n_1) > \phi(\boldsymbol{x}, n_2)$，则称 n_1 比 n_2 更活跃。对于第 i 层和给定的测试输入 $\boldsymbol{x}$，我们使用 $top_k(\boldsymbol{x}, i)$ 来表示在第 i 层上有最大输出的 k 个神经元。

top-k 神经元覆盖率（Top-k Neuron Coverage，TKNCov）： Top-k 神经元覆盖率衡量的是每一层网络中有多少神经元曾经是该层中最活跃的 k 个神经元。该标准可以被定义为如下：

$$TKNCov(T, k) = \frac{|\bigcup_{\boldsymbol{x} \in T}(\bigcup_{1 \le i \le l} top_k(\boldsymbol{x}, i))|}{|N|} \tag{2-7}$$

来自同一层的神经元往往扮演着相似的角色，来自不同层的最活跃的神经元是表征 DNN 主要功能的重要指标。为了更彻底更充分地测试 DNN，测试数据集应该覆盖更多的最活跃的神经元。

2.3 蜕变测试

软件测试是保证和验证软件质量的主流方法。然而，在软件测试的过程中存在着两个基本问题：测试预言（Test Oracle）问题和可靠测试集问题。

测试预言问题是指在某些特殊情况下，我们无法验证或者很难去验证给定测试用例的测试结果。通常情况下，在执行一个测试用例 t 之后，需要一个称为测试预言（或简单的预言）的系统机制来检查执行结果。如果执行结果与预期的结果不一致，我们就称测试用例 t 失败了，并将其作为导致失败的测试用例。否则，我们就称 t 成功了，并将其作为一个成功的或者不会导致失败的测试用例。然而，在许多实际场景中，预言可能并不存在，或者存在但是由于资源限制无法使用 [23]。

可靠测试集问题是指在通常情况下，我们不可能详尽地执行所有可能的测试用例，因此要有效地选择一个能够判断程序正确性的测试用例子集（可靠测试集）是一个艰巨的挑战。为了解决可靠测试集问题，人们提出了许多策略，

包括随机测试 [24]、覆盖测试 [25]、符号执行 [26] 和基于搜索的测试 [27]。与测试用例生成策略相比，只有少数技术被提出来解决预言问题，如断言检查 [28] 和 N 版本编程 [29]。当出现预言问题时，许多用于解决可靠测试集问题的策略的适用性和有效性都很有限。即使一个策略能够有效地生成导致失败的测试用例，除非它能导致被测试程序崩溃，否则这个失败的测试用例在发生预言问题时也有可能无法被识别出来 [23]。

在传统的软件测试过程中，测试人员判断被测程序是否正确的方法是验证实际输出和期望输出的一致性，但在一些情况下，我们无法准确地预测被测程序的期望输出来作出正确的判断。蜕变测试（Metamorphic Testing）作为一种特殊的黑盒测试方法能够有效地应对这种情况。蜕变关系（Metamorphic Relation）是蜕变测试的核心，是根据被测软件的领域知识和实现方法来建立的 [30]。蜕变关系是指多次执行目标程序时程序输入与输出之间所期望遵循的关系，是目标函数或算法在涉及多个输入及其期望输出时的必要属性。在进行蜕变测试时，一些程序的源输入首先被生成为源测试用例。在此基础上蜕变关系可以被用来生成新的输入作为后续的测试用例。蜕变测试既可以用于生成新的测试用例，也可以用于验证测试结果的正确性，因此解决了软件测试的两个基本问题 [31]。

2.4 数字图像处理技术

图像处理技术是指应用计算机技术对图像信息进行编辑和加工的技术，从而满足人的视觉、心理以及实际应用需求。数字图像处理技术可以编辑和修改图像的内容，从而扩增生成新的图像数据 [32]。

2.4.1 图像的基本表示方法

数字图像是由像素点（Pixel）构成的，对于每个像素点，人们用数值的大小来衡量其亮度的深浅。其中，像素点的属性包括灰度和位置，所有像素点通过排列与组合就能以多维矩阵的形式来表示图像。在实际应用中，通常把图像矩阵的左上角设为原点，用二维坐标来表示像素点在图像矩阵中的具体位置。根据图像的灰度级差异和通道数可以把数字图像归为二值图像、灰度图像和彩色图像三类图像。

（1）二值图像：二值图像的像素点的灰度属性的取值范围只有 0 和 1 这两

个数值。其中像素点为 1 代表该像素点为白色，为 0 代表该像素点为黑色。

（2）灰度图像：在二值图像中只有黑色和白色两种颜色，这种图像表示方法会使图像损失很多内容和细节。为了提高图像的质量和内容丰富度，人们提出了灰度图像。灰度图像也是以矩阵的形式来表示的，矩阵中每个元素的值代表该像素点亮度的深浅。与二值图像不同的是，灰度图像的像素点颜色在黑色和白色之间还划分了很多等级。我们一般用八位二级制数来表示一个像素点的灰度值，每个像素点的颜色可以划分为 256 个等级，其中 0 代表纯黑色，255 代表纯白色，从 0 到 255 像素点的亮度逐渐变浅。

（3）彩色图像：彩色图像是三通道图像。自然界中的绝大部分色彩都可以由三种基色（红绿蓝）按照一定的比例合成得到。因此，人们以三维矩阵的形式来表示彩色图像，彩色图像中每个像素点由 R、G、B 三种分量（或其他各种分量）组成，每种分量参照灰度图像划分亮度等级的方式进行表示。

2.4.2 图像的几何变换

图像的几何变换是指把原图像中像素的坐标位置映射到新图像中像素所对应的坐标位置的过程，在这个过程中图像的像素只有位置产生变化，而灰度值保持不变。图像的几何变换主要可以划分为以下几类：

（1）图像的平移：

图像的平移变换是指在 x 轴和 y 轴方向上对原图像的像素坐标进行算数操作。假设一个像素点在原图像中的位置坐标为 (x_0, y_0)，在经过平移量为 $(\Delta x, \Delta y)$ 的平移变换后，坐标变为 (x_1, y_1)，其中 $x_1 = x_0 + \Delta x$，$y_1 = y_0 + \Delta y$，该计算过程可以用矩阵表示为如下：

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix}$$

其中矩阵：

$$\begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix}$$

称为平移变换矩阵。

（2）图像的缩放：

图像的缩放变换是指将图像的 x 轴和 y 轴以一定的比例缩放生成新图像的过程。如果图像的 x 轴和 y 轴的缩放比例相同，则称之为全比例缩放。如果图像的的 x 轴和 y 轴缩放比例不同，会使图像比例失调，导致图像畸变。

在图像进行缩小变换时，图像中像素点的数量也会随之减少，因此需要按照一定的规则来保留图像中的一些关键像素点，从而维持图像内容的基本特征。常见的像素点选取方法有局部均值法和等间隔采样法。局部均值法首先将图像矩阵划分为特定大小的子矩阵，并取每个子矩阵中像素值的平均值作为新像素点来替代子矩阵。等间隔采样法首先生成一个间隔数，然后在图像矩阵中以间隔数为间距选取保留像素点。

在图像进行放大变换时需要通过插值法来对放大后的图像区域进行填充。常用的插值算法有线性插值法和最近邻域法。线性插值法对插入像素点附近邻域的像素灰度值进行线性插值来计算待填补区域像素点的灰度值。最近邻域法则是选取与该像素点距离最近的像素点来填补空白区域。图像缩放变换的计算过程可以用矩阵表示为如下：

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ S_y & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix}$$

其中矩阵：

$$\begin{bmatrix} S_x & 0 & 0 \\ S_y & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

称为缩放变换矩阵，S_x，S_y 分别为 x，y 方向上的缩放因子。

（3）图像的旋转：

图像的旋转变换就是将原图像中的所有像素点都围绕旋转中心顺时针或逆时针旋转一个给定的角度。设点 $P_0(x_0, y_0)$ 旋转 θ 度后对应的坐标点为 $P_1(x_1, y_1)$，则图像旋转变换的计算过程可以用矩阵表示为如下：

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix}$$

变换公式为：

$$x_1 = x_0 \cos\theta + y_0 \sin\theta$$

$$y_1 = -x_0 \sin\theta + y_0 \cos\theta$$

2.5 信息熵

信息是客观事物状态及其运动的一种普遍形式，它是确保客观世界或系统具有一定内部结构和功能的基础。信息论是关于信息的理论，它是运用概率理论和数理统计方法，从数量的角度出发研究信息的度量、获取、传递和处理等，以解决通信系统、数据传输、密码学和数据压缩等有关信息问题的一门学科 [33]。

在信息理论中，熵（Entropy）通常也称作信息熵或者 Shannon 熵 [34]，它主要以数值的形式来表示随机变量取值的不确定性程度，目的是刻画信息含量的多少。Shannon 在信息论中用一个事件发生的不确定性来度量该事件所包含的信息量：如果一个事件发生的概率越小，那么这个事件所包含的信息量就越多；反之则包含的信息量就越少。假设 X 是一个随机变量，$p(x)$ 表示变量 X 的取值是 x 的概率，那么它的不确定性程度可以用信息熵 $H(X)$ 形式表示如下：

$$H(X) = -\int_x p(x) \log p(x) \mathrm{d}x \tag{2-8}$$

由式 2-8可知，信息熵 $H(X)$ 与变量 X 的具体取值无关，只与 X 的概率分布有关。如果随机变量 X 的不确定性程度越高，即概率分布越大，那么它的信息熵也就越大，相应的其所需要的信息量越多。当变量 X 的所有取值出现的概率都相同时，变量 X 的不确定性程度达到最高，其熵值相应地也达到最大。在这种情况下，X 的具体取值只能随机选取任意一个值。反之，如果变量 X 的取值范围只有一个值 x 时，该值的概率分布 $p(x)$ 为 1，此时 X 是完全确定的，其熵值达到最小 [35]。

此外，由式 2-8可知：

$$H(X) = -E(\log p(x)) \tag{2-9}$$

其中 $E(x)$ 是 x 的概率期望。式 2-9说明信息熵在某种程度上可以理解为随机变量 X 的概率分布对数的期望值。假设 X 为离散随机变量，那么它的信息熵可以

表示为如下形式：

$$H(X) = -\sum_{x \in dom(X)} p(x) \log p(x) \tag{2-10}$$

其中 $dom(X)$ 为 X 的值域。

2.6 本章小结

本章主要介绍了在本文工作中所涉及到的相关技术及其理论基础。首先介绍了深度神经网络的概念，这是本文工作的理论和知识基础。然后介绍了当今主流的神经网络测试的覆盖标准，这些标准将为本文的神经网络模糊测试框架的评估和分析提供重要的理论支撑。接着介绍了蜕变测试，本文的图像变异方法是采用图像在变换过程中的蜕变关系作为预言来保证图像的语义。接下来介绍了数字图像处理相关的技术，主要包括图像的表示方法以及相关的计算和处理技术，这些理论和技术将为本文的图像变异方法提供重要支撑。最后，本文介绍了信息熵相关的理论知识，这些理论知识为本文的神经网络模糊测试框架的核心引导方法提供了重要理论支撑。

第三章　基于变异图像熵值引导的神经网络模糊测试方法

3.1　技术路线与研究内容

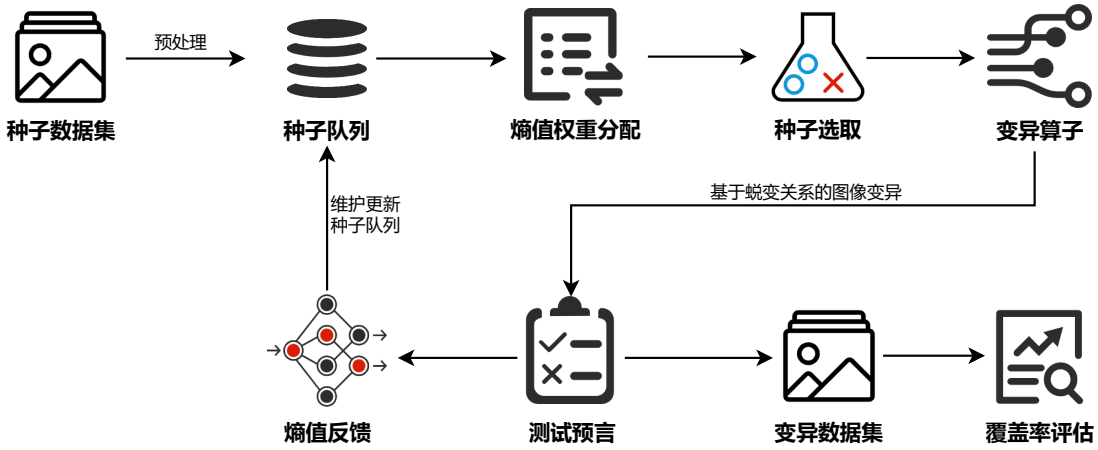


图 3-1: 技术路线示意图

本文的整体技术路线与研究内容如图 3-1所示。首先，我们对初始种子数据集进行预处理，将图像数据集中的种子转换为适合进行图像变异的格式，并以此构建初始种子队列。接着，对于队列中的种子，我们根据熵值来为其分配相应的权重，并以权重为概率选取给定数量的种子进行基于蜕变关系的图像变异。对于原始种子的每个有效变异，我们会用测试预言对其进行验证，对于验证失败或者触发了错误的变异图像我们将其添加到变异数据集中。对于新生成的变异图像我们计算并更新它的熵值，并将其添加到种子队列中。经过给定的时间或给定的变异次数后，我们使用经过变异所得到的扩增后的种子数据集对待测模型进行覆盖率评估，总结分析得到的实验结果。我们还结合工业表面缺陷检测的领域知识对图像变异方法进行了改进，并将本文的方法应用于工业缺陷数据集 NEU-DET 来验证其应用效果。最后，我们根据实验结果对整体的研究内容进行分析，总结和评估整个系统中所存在的问题。

3.2 基于蜕变关系的图像变异方法

传统的模糊器（Fuzzer），例如 AFL [①]，通过逐位/逐字节翻转、块替换、交叉输入文件等方式来对种子进行变异，通过随机的变异来生成不同的测试用例以最大化模糊测试的覆盖率。如果变异体导致了异常行为（例如程序崩溃），那么就称之为失败测试用例。然而，种子变异在 DNN 测试中的效果并不明显，因为缺乏显式预言来验证变异体的结果。例如，图像变异可能会生成无意义的图像。如图 3-2所示，我们对原图像添加椒盐噪声来进行变异。人们通常使用信噪比（Signal-Noise Rate, SNR）来衡量图像的噪声。SNR 越小，噪声占比越大，图像也就越模糊不清。当 SNR=0.9 时，可以看出变异图像与原图像只有细微的差别，我们可以认为它是一个有效变异。但是当 SNR=0.3 时，变异图像已经十分模糊，无法辨认其轮廓，我们可以认为它是一个无效变异。但是在实际的测试中，通过人工来验证变异体的正确性和有效性不仅效率低下，而且人力成本也很高，因此测试预言显得尤为重要。本文采用变异图像在变换过程中的蜕变关系作为预言。

(a) 原图像 (b) 添加椒盐噪声（SNR=0.9） (c) 添加椒盐噪声（SNR=0.3）

图 3-2: 图像变异示意图

本文对基于蜕变关系的图像变异定义如下：

基于蜕变关系的图像变异（Metamorphic Image Mutation）： 给定一个理想的测试预言 O 和一个特定输入域的测试用例 x，我们定义在 x 上的基于蜕变关系的变异策略 M，对于 $\forall x' \in M(x)$，有 $O(x') = O(x)$。

基于蜕变关系的图像变异生成了一个预言，从人类的角度来看，变异体 x' 的语义和 x 的语义是一样的。在本文中，我们实现了一个能在很大程度上保留原有语义的种子图像变异策略。为了使变异策略更加有效，要在维持变异体原

[①]https://llvm.org/docs/LibFuzzer.html

有语义的同时保证其有一定程度的变化。如果变异策略对种子的改变微乎其微，那么新生成的测试用例触发错误的概率以及提高覆盖率的可能性也就很低。如果对种子改变过大，那么新生成的测试用例可能会改变原有的语义而变成无效输入。我们的变异策略选择了两种类别的 8 种图像变换：

- 像素值变换（Pixel Value Transformation）$\mathcal{P}$：图像亮度、图像对比度、图像模糊和图像噪声。如图 3-3所示。
- 仿射变换（Affine Transformation）$\mathcal{G}$：图像缩放、图像平移、图像剪切和图像旋转。如图 3-4所示。

(a) 原图像 (b) 亮度变换

(c) 对比度变换 (d) 模糊变换 (e) 噪声变换

图 3-3: 像素值变换示意图

像素值变换会改变图像中像素点的值，而仿射变换会移动图像中像素点的位置。这些图像变换操作在 DeepTest [15] 中已经被证明是有效且实用的。我们将单次变异定义为如下：

单次变异（One-time Mutate）：给定一个变换 $t \in \mathcal{P} \cup \mathcal{G}$，一个原图像 s，如果 s' 是由变换 t 在 s 上作用生成的，那么我们说 s' 是由 s 经过单次变异而得到的，记作 $s \xrightarrow{t} s'$。

在经过大量实验后，我们对各个图像变换的参数作出了适当调整，可以假设新图像经过单次变异后保持原语义不变。然而，在模糊测试的过程中，一个种子图像可能会经历一系列的变异，在经历多次变异后很难保持其原有语义和有效性。

(a) 原图像 (b) 缩放变换

(c) 平移变换 (d) 剪切变换 (e) 旋转变换

图 3-4: 仿射变换示意图

为了使变异图像的语义接近原始种子，我们采取比较保守的策略，至多使用一次仿射变换（我们假设只经过一次仿射变换的变异不会影响图像的语义），因为多次仿射变换更容易导致图像变得无法识别。我们可以多次使用像素值变换来提高可变性，并用 L_0 和 L_∞ 来约束像素级别的变化。假设图像 s' 是由图像 s 经过像素值变换而得到的，那么当 $f(s, s')$ 被满足时，我们认为 s' 是一个有效的变异。

$$f(s,s') = \begin{cases} L_\infty(s,s') \le 255, & \text{if } L_0(s,s') < \alpha \times \text{size}(s) \\ pHash(s,s'), & \text{otherwise} \end{cases} \tag{3-1}$$

其中，$0 < \alpha, \beta < 1$，$L_0(s, s')$ 表示 s 与 s' 之间发生变化的像素点的最大数量，L_∞ 表示一个像素发生改变的最大值，$\text{size}(s)$ 表示图像 s 中的像素点总数，$pHash(s, s')$ 表示感知哈希算法。

感知哈希算法（Perceptual Hash Algorithm，PHA），是由传统的哈希算法演变而来的一种生成图像数据指纹的算法。感知哈希是一类图像数据集到感知摘要集的单向映射，即把图像数据唯一地映射为一段数字摘要，并满足感知安全性和鲁棒性 [36]。图像感知哈希的映射关系如图 3-5所示。

感知哈希算法的步骤如下：

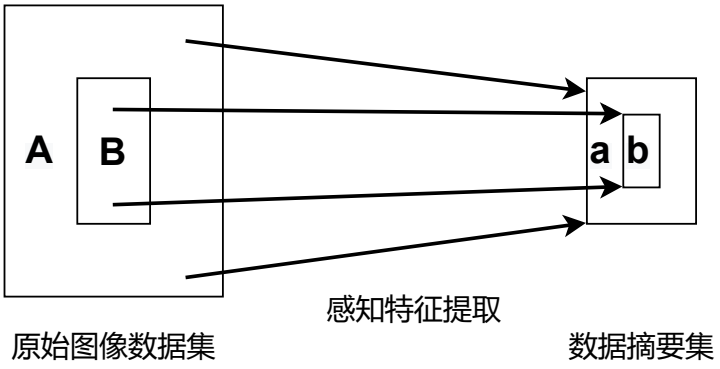


图 3-5: 图像感知哈希的映射关系

1. 缩小尺寸：将图片缩小到 8×8 的 64 像素大小的尺寸，从而在一定程度上消除不同尺寸和比例所导致的图片差异。
2. 简化色彩：将缩小后的图片转换成灰度图像，进一步简化计算。
3. 计算 DCT（离散余弦变换）：DCT 将图像转化为频率和标量的集合，得到 32×32 的 DCT 系数矩阵。
4. 缩小 DCT：虽然得到 DCT 矩阵是 32×32 大小，但只需要保留左上角呈现了图片中的最低频率的 8×8 的矩阵。
5. 计算平均值：计算 DCT 矩阵的 64 个值的平均值。
6. 进一步减小 DCT：在 8×8 的 DCT 矩阵中，把大于等于 DCT 均值的设为 1，小于 DCT 均值的设为 0。如果图像的整体结构不变，hash 结果值也不会发生变化。
7. 计算哈希值：将上一步的结果转换成 64 位的长整型，就得到了这个图像的指纹。得到指纹以后，就可以通过计算不同图像指纹之间的汉明距离（Hamming Distance）来比较不同图像的相似度。

直观地说，如果变异图像中发生改变的像素数量比较少（少于 $\alpha \times$ size(s)），我们可以认为该变异不改变其原有的语义，那么 L_∞ 可以为任何数。如果发生改变的像素数量超过了我们所设定的阈值，那么我们就使用感知哈希算法来对变异图像和原图像之间的语义进行判断。

然而对于经过仿射变换的变异图像，其像素点可能会失去与原图像一一对应的关系，为了计算其 L_0 和 L_∞，我们定义了参考图像：

参考图像（Reference Image）：给定一个由 s_0 经过单次变异或一系列变异得到的图像 s（s_0 是一个初始种子中的图像），即 $s_0 \xrightarrow{t_0,\ldots,t_n} s$，其中 $n \geq 0$，参考

图像（用 s'_0 表示）可以定义为如下：

$$s'_0 = \begin{cases} s_j, & \exists 0 \le j \le n.\ t_{j-1} \in \mathcal{G} \wedge s_0 \xrightarrow{t_0,\ldots,t_{j-1}} s_j \\ s_0, & \text{otherwise} \end{cases} \tag{3-2}$$

参考图像就是原始图像（如果没有经历过仿射变换）或经历仿射变换后所生成的图像。注意，在一个变异序列中只能有一次仿射转换。对于包含仿射变换 t_{j-1} 的变异序列，如 $s_0 \xrightarrow{t_0,\ldots,t_{j-1}} s_j \xrightarrow{t_j,\ldots,t_{n-1}} sn$，$s_0$ 与 s_n 之间的 L_0 和 L_∞ 的计算如下所示：

$$\begin{aligned} L_0(s_0, s_n) &= L_0(s_0, s_{j-1}) + L_0(s_j, s_n) \\ L_\infty(s_0, s_n) &= MAX(L_\infty(s_0, s_{j-1}), L_\infty(s_j, s_n)) \end{aligned} \tag{3-3}$$

算法 3.1展示了基于蜕变关系的图像变异方法的实现细节。s 是原始图像种子，K 是待生成的变异图像总数，输出 T 是新生成的变异图像数据集。我们首先记录初始种子 s_0 和参考图像 s'_0 的信息（第 1 行），然后生成 K 个新的变异图像（第 3- 23 行）。为了生成一个变异图像，我们会尝试最多 MAX_TRY 次（第 5-19 行）来对种子图像进行变异。如果种子图像与参考图像相同，则说明没有使用仿射变换。在这种情况下，可以选择仿射变换和像素值变换（第 7 行），否则就只能选择像素值变换（第 8 行）。对于被选中的变换 t，我们随机选择一个参数（第 10 行）并进行变异（第 11 行）。我们计算参考图像 s'_0 和变异图像 s' 之间的 L_0 和 L_∞ 以及感知哈希算法来判断 s' 是否有意义（第 12 行）。如果仿射变换被选中，那么它会更新参考图像（第 13-14 行）。如果在 MAX_TRY 次尝试之后仍没有成功的变异，我们就将原始图像 s 添加到 T 中（第 21 行）。

种子的优先级决定了其被选中的概率。在传统软件程序之中，如果一个种子覆盖了一个分支，模糊器更有可能通过对该种子进行变异来覆盖其下面的分支，因为分支之间存在层次关系，如图 3-6所示。因此，传统的覆盖率引导的模糊器通常更喜欢从队列中选择新生成的变异体。然而，该策略在 DNN 测试中的有效性仍未得到充分的验证。

除了基于熵值的种子选取策略以外，我们还实现了两个种子选取策略：（1）均匀抽样策略：随机从队列中选择一个种子。（2）基于变异次数的种子选取策略，选择种子 s 的概率计算公式如下：

算法 3.1 基于蜕变关系的图像变异

Input: s:Seed, K:Total number of tests to be generated,
MAX_TRY:The maximum number of trials
Output: T:A set of new generated tests

```
1: (s_0, s'_0) ← info(s);
2: T ← ∅;
3: for i from 1 to K do
4:    Success ← False;
5:    for j from 1 to MAX_TRY do
6:       if s'_0 is the same with s_0 then
7:          t ← randomPick(G ∪ P);
8:       else
            t ← randomPick(P);
9:       end if
10:      p ← pickRandomParam(t);
11:      s' ← t(s, p);
12:      if isSatisfied(f(s_0, s')) then
13:         s'_0 ← t(s_0, p);
14:         info(s') ← (s_0, s'_0);
15:      end if
16:      Success ← True;
17:      T ← T ∪ {s'};
18:      break;
19:   end for
20:   if notSuccess then
21:      T ← T ∪ {s};
22:   end if
23: end for
24: return T;
```

$$P(s) = \begin{cases} 1 - g(s)/\gamma, & \text{if } g(s) < (1 - p_{min}) \times \gamma \\ p_{min}, & \text{otherwise} \end{cases} \tag{3-4}$$

其中，$g(s)$ 表示种子 s 经过变异的次数，$p_{min} > 0$ 是种子 s 能被选择的最小概率，γ 是一个约束概率下降比率的参数。

我们遍历种子队列，并根据其概率来确定是否要选择当前的种子。我们给新生成的变异体分配更高的概率，因为它更可能会增加覆盖率。为了保证种子变异的多样性，其他经过多次变异的种子也有一个最小的概率 p_{min} 会被选中。

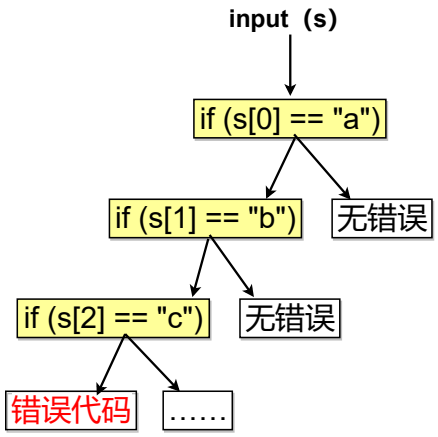


图 3-6: 传统软件程序的分支层次关系

3.3 基于熵值引导的灰盒模糊测试

在本文中，我们假设模糊器的效率是由每个生成的输入所揭示的程序行为的平均信息决定的。香农熵（Shannon's entropy）是一个经典的信息度量。如果模糊器所执行的程序行为基本相同，那么香农熵就比较小，每个输入的信息量就比较低，模糊器发现新行为的效率也就比较低。然而，如果大多数模糊器生成的输入都触发了以前从未见过的程序行为，那么香农熵就会很高，模糊器在发现新行为方面也会表现得更加高效 [37]。

基于上述见解，我们开发了基于变异图像熵值引导的神经网络模糊测试方法。该方法的目标是通过最大化变异图像的熵值来提高模糊测试的效率。模糊器对输入的种子图像进行变异来产生新的输入图像，并将生成的新输入添加到种子语料库中。种子的权重决定了种子被选择的概率。权重越大的种子越容易被选中进行变异。我们根据熵值来为种子语料库中的种子分配相应的权重。在理想情况下，我们希望将大部分权重分配给具有更大熵值的种子，以揭示更多关于 DNN 的信息。

3.3.1 基于黑盒模糊测试的概率框架

假设 $\mathcal{M}$ 是我们想要对其进行模糊测试的神经网络模型，我们称 $\mathcal{M}$ 能接受的所有输入的集合为 $\mathcal{M}$ 的输入空间，记作 $\mathcal{D}$。对 $\mathcal{M}$ 的模糊测试是从程序的输入空间中对 N 个输入进行放回抽样的随机过程，记为 $\mathcal{F}$，如式 3-5所示。我们称一个执行 $\mathcal{F}$ 的工具为非确定性黑盒模糊器（Non-deterministic Blackbox

Fuzzer）。

$$\mathcal{F} = \{X_n \mid X_n \in \mathcal{D}\}_{n=1}^{N} \tag{3-5}$$

我们将搜索空间 $\mathcal{D}$ 划分为 S 个独立的子空间 $\{\mathcal{D}_i\}_{i=1}^{S}$，并称其为物种。如果 $X_n \in \mathcal{D}_i$，且不存在之前抽样过的输入 $X_m \in \mathcal{F}$ 使得 $m < n$ 且 $X_m \in \mathcal{D}_i$（即 $\mathcal{D}_i$ 是第一次被抽样），则称输入 $X_n \in \mathcal{F}$ 发现了物种 $\mathcal{D}_i$。一个输入所属的物种是根据该输入执行的动态程序属性来定义的。例如，输入 $X_n \in \mathcal{D}$ 被输入到模型 $\mathcal{M}$ 后所预测的类别可以被标识为一个物种。

我们设 p_i 是第 n 次生成的输入 X_n 属于物种 $\mathcal{D}_i$ 的概率，如式 3-6所示，其中 $1 \le i \le S$ 且 $1 \le n \le N$。

$$p_i = P[X_n \in \mathcal{D}_i] \tag{3-6}$$

我们称 $\{p_i\}_{i=1}^{S}$ 为模糊器的全局物种分布。物种发现数量的期望值如式 3-7所示。

$$S(n) = \sum_{i=1}^{S}[1 - (1 - p_i)^n] = S - \sum_{i=1}^{S}(1 - p_i)^n \tag{3-7}$$

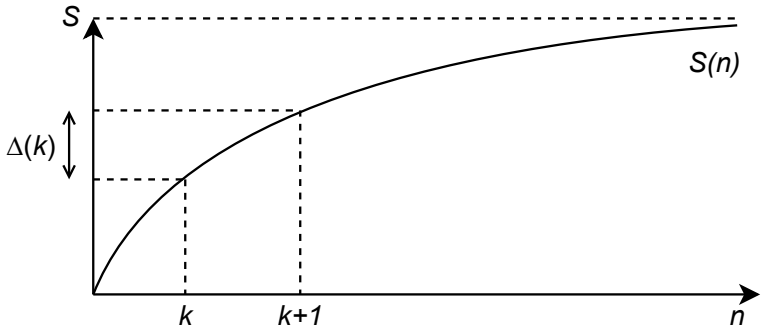


图 3-7: 物种发现曲线

图 3-7展示了一个物种发现曲线 $S(n)$ 的例子，即 n 个生成的测试用例所覆盖的物种数量。当生成的测试输入的数量 n 增加时，测试生成器发现的程序行为的预期数量为 $S(n)$。随着时间的推移，发现程序行为的速率 $\Delta(n) = S(n+1) - S(n)$。函数 $\Delta(n)$ 给出了生成第 n 次测试输入时的当前发现率。在极限情况下，所发现的程序行为的期望数目接近于渐近线 S。模糊器发现的物种数的极限 S 如式 3-8所示。

$$S = \lim_{n \to \infty} S(n) \tag{3-8}$$

发现率 $\Delta(n)$，即第 $(n+1)$ 次生成的测试输入所发现的物种的期望数量可以定义为 $\Delta(n) = S(n+1) - S(n)$。当长时间没有新的进展，即 $\Delta(n) < \theta$ 时，可以终止模糊测试。

设 C 是一组输入种子，我们称之为种子语料库，设 q_t 是模糊器选择种子 $t \in C$ 的概率。对于每个种子 t，设 $\mathcal{D}^t$ 为种子 t 运用所有变异算子所能生成的所有变异体的集合。对种子 t 的变异可以看作是对 N^t 个输入进行抽样，并用生成的种子 t 的变异体进行放回的一个随机过程。我们把所有能通过对一个种子 t 进行变异而发现的物种称为 t 的邻近物种。

我们设 p_i^t 是通过对种子 $t \in C$ 进行变异而生成的第 n 个输入 X_n^t 属于物种 $\mathcal{D}_i$ 的概率，如式 3-9所示，其中 $1 \le i \le S$ 且 $1 \le n \le N$。

$$p_i^t = P[X_n^t \in \mathcal{D}_i] \tag{3-9}$$

我们称 $\{p_i^t\}_{i=1}^S$ 为种子 t 的邻近局部物种分布。根据总期望定律，全局分布和局部分布的关系如式 3-10所示，其中 $1 \le i \le S$。

$$p_i = \sum_{t \in C} q_t \cdot p_i^t \tag{3-10}$$

与式 3-7合并之后可得式 3-11，$S(n)$ 是基于变异的黑盒模糊器的物种发现曲线。

$$S(n) = \sum_{i=1}^{S} [1 - (1 - \sum_{t \in C} q_t \cdot p_i^t)^n] \tag{3-11}$$

对于我们的非确定性黑盒模糊测试的概率模型，我们要求全局和局部物种分布在整个模糊测试过程中是不变的。也就是说，对于 $1 \le i \le S$ 和 $1 \le n \le N$，q_t 是模糊器选择 $t \in C$ 的概率，X_n 和 X_{n+1} 分别是模糊器生成的第 n 个和第 $n+1$ 个测试输入，X_n^t 和 X_{n+1}^t 分别是模糊器通过对种子 t 进行变异而生成的第 n 个和第 $n+1$ 个测试输入，式 3-12是成立的。

$$\begin{aligned} p_i &= P[X_n \in \mathcal{D}_i] = P[X_{n+1} \in \mathcal{D}_i] \\ p_i^t &= P[X_n^t \in \mathcal{D}_i] = P[X_{n+1}^t \in \mathcal{D}_i] \end{aligned} \tag{3-12}$$

如果对于程序输入空间中的任意程序输入 $d \in \mathcal{D}$，有 $P[X_n = d] = P[X_{n+1} = d]$，即在整个模糊测试过程中产生某个输入 d 的概率是不变的，那么上述式子是成立的。

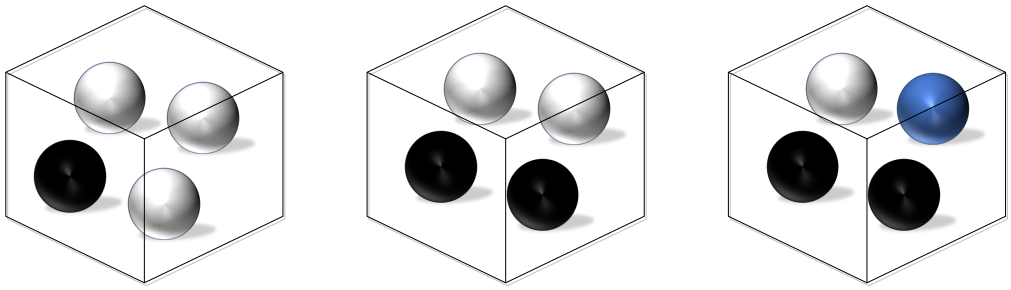
同时，考虑到黑盒模糊器可以从非均匀分布中产生测试输入，即对于任意两个输入 $d_1, d_2 \in \mathcal{D}$，第 n 次测试输入是 d_1 或 d_2 的概率完全有可能是不同的，即 $P[X_n = d_1] \neq P[X_n = d_2]$。除此之外，物种之间的分布可能不是均匀的，一些稀有物种（例如，p_i 非常小）可能聚集在输入空间的一个小区域内。

3.3.2 模糊器效率的信息论度量

在模糊测试的背景下，香农熵 H 有几种不同的解释。它量化了一个生成的测试输入所揭示的关于程序行为的平均信息。熵还提供了能够可靠地存储模糊器所能发现的所有行为集合所需的最小信息单元数。此外，熵还是多样性的一种度量。较低的熵意味着程序要么只展示了较少的行为，要么大多数生成的输入都只触发了相同的行为（即生成的输入都属于一个或几个数量最多的物种）。

香农熵 H 度量了在每个样本 $X_n \in \mathcal{F}$ 中对于执行模型 $\mathcal{M}$ 所能观察到的物种的平均信息量。当存在 S 个不同的物种时，熵 H 的定义如下：

$$H = -\sum_{i=1}^{S} p_i \log(p_i) \tag{3-13}$$



(a) $p_1 = \frac{3}{4}, p_2 = \frac{1}{4}, H = 0.56$　(b) $p_1 = p_2 = \frac{1}{2}, H = 0.69$　(c) $p_1 = \frac{1}{4}, p_2 = \frac{1}{2}, p_3 = \frac{1}{4}, H = 1.04$

图 3-8: 放回抽样示意图

图 3-8简单地阐明了熵的概念。图中的三个箱子里的每种颜色的球代表不同的物种，我们通过抽样来了解每个箱子中球的颜色分布情况。从每个箱子中所能获取的关于球的颜色信息都不一样。例如，在图 3-8(a)中，抽到白球的可能性是抽到黑球的三倍。与图 3-8(b)相比，在图 3-8(a)中抽到黑球需要更多的尝试次数。因此，在图 3-8(a)中，我们获得关于箱子中球的颜色信息的期望值就比较小。事实上，给定 S 种颜色的球，当抽到所有颜色的球都是等概率，即 $p1 = \ldots = p_S$ 时，熵取得最大值的。在这三个箱子中，图 3-8(c)的熵值最大，

能得到的关于箱子中球的颜色的信息也是最多的。尽管图 3-8(c)的箱子中黑球的数量最多，但我们也可以抽到白球和蓝球。

香农熵是为多项式分布定义的，其中每个输入只属于一个物种（例如，一个图片只属于一个类别）。然而，一个输入可能会属于多个物种，从而导致 $\sum_{i=1}^{S} p_i \geq 1$。例如，一个图像中可能含有一个以上的目标类别。当有可能存在 $\sum_{i=1}^{S} p_i \geq 1$ 的情况时，可以将概率进行归一化，如式 3-14所示。

$$H = \log\left(\sum_{j=1}^{S} p_j\right) - \frac{\sum_{i=1}^{S} p_i \log(p_i)}{\sum_{j=1}^{S} p_j} \tag{3-14}$$

我们可以注意到，在 $\sum_{j=1}^{S} p_j = 1$ 的特殊情况下，式 3-14可简化为式 3-13。

回顾式 3-9，我们把对一个种子 $t \in C$ 进行变异所产生的输入属于物种 $\mathcal{D}_i$ 的概率 $\{p_i^t\}_{i=1}^{S}$ 称为种子 t 的局部物种分布。此外，我们称物种的集合 $\{\mathcal{D}_i \mid p_i^t > 0 \wedge 1 \leq i \leq S\}$ 为种子 t 的邻域。根据 t 的局部物种分布，我们可以直接套用式 3-14来计算 t 的局部熵 H^t。

$$H^t = \log\left(\sum_{j=1}^{S} p_j^t\right) - \frac{\sum_{i=1}^{S} p_i^t \log(p_i^t)}{\sum_{j=1}^{S} p_j^t} \tag{3-15}$$

局部熵 H^t 量化了对种子 t 进行变异所揭示的关于 t 的邻域内物种的信息。

我们基于观察到的每个物种的数量来估计香农熵 H。物种 $\mathcal{D}_i$ 的物种数量 Y_i 是通过变异生成的测试输入属于物种 $\mathcal{D}_i$ 的数量，未被发现过的物种其物种数量 $Y_i = 0$。局部发现概率 p_i 的无偏估计是 $\hat{p}_i = Y_i/n$，其中 n 是生成的测试输入的总数。将 $\hat{p}_i$ 代入式 3-14，可以利用极大似然估计法来估计熵 H。在我们的模型中，H 的估计熵 $\hat{H}_{MLE}$ 如式 3-16所示。

$$\hat{H}_{MLE} = \log\left(\sum_{j=1}^{S} Y_j\right) - \frac{\sum_{i=1}^{S} Y_i \log(Y_i)}{\sum_{j=1}^{S} Y_j} \tag{3-16}$$

其中 $\sum_{j=1}^{S} Y_j > 0$。

在第 3.3.1节中，我们引入了非确定性黑盒模糊测试的概率模型，该模型满足全局物种分布 $\{p_i\}_{i=1}^{S}$ 在模糊测试过程中保持不变的假设。然而，这种假设对于利用程序反馈的灰盒模糊测试并不适用。模糊器生成的一些新输入（如增加了神经元覆盖率的输入）会被添加到种子语料库中，新添加的种子改变了全局物种分布（但不是每个种子的局部分布），从而改变了灰盒模糊器的全局

熵。也就是说，灰盒模糊器会随着时间的推移变得更加高效。

显然，我们的概率模型并不适合直接用于描述非确定性灰盒模糊器的效率，但是可以通过它来推导出一种合理的启发式算法来估计灰盒模糊器当前的全局熵。我们可以将灰盒模糊测试看作是一系列基于变异的黑盒模糊测试过程 $\{\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_m\}$，每个模糊测试过程都从一个固定大小的种子语料库开始，并将新生成的种子添加到种子语料库中。因此，对于灰盒模糊器，我们仅使用向种子语料库中添加最后一个种子后所计算得到的物种数量来估计全局熵，我们称之为去偏估计量。

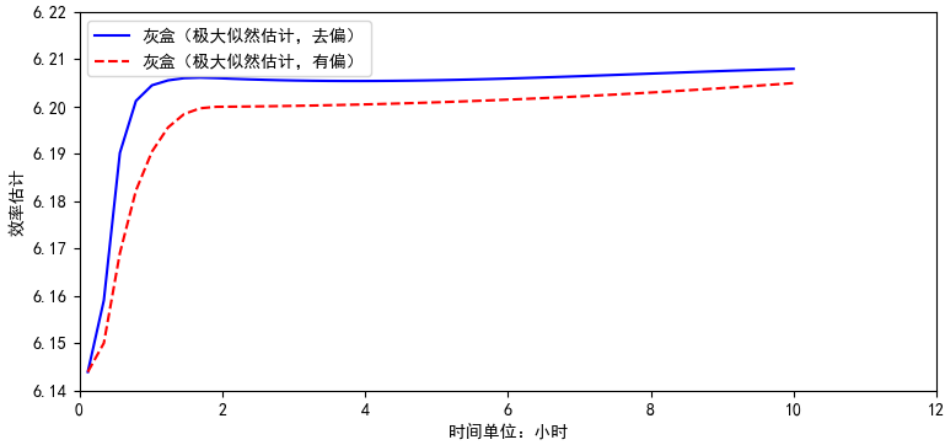


图 3-9: 模糊器的熵估计

图 3-9显示了灰盒模糊器的两个熵估计：有偏的极大似然估计以及去偏的极大似然估计随时间的变化关系。我们期望灰盒模糊器的效率和熵随着时间的增加而增加。随着越来越多的种子被添加到种子语料库，之前稀有的物种变得不那么稀有了，我们可以观察到一个单调增加的曲线。有偏的极大似然估计是由未修改的物种数量 Y_i 计算得到的，而去偏极大似然估计是根据每次添加新种子时就被重置为 0 的物种数量计算得到的。将灰盒模糊测试过程视为一系列的黑盒模糊测试过程，并对物种数量进行重置，可以避免偏置对当前熵估计的影响。

在图 3-9中我们可以观察到，去偏极大似然估计接近最终的熵估计只花费了一个小时左右的时间，而未修改的有偏极大似然估计接近最终的熵估计则花费了超过 10 个小时，速度远慢于去偏极大似然估计。相对地增加数据集质量可以在一定程度上弥补将物种数量重置为 0 所带来的不足。

3.3.3 熵值引导的灰盒模糊测试的具体实现

灰盒模糊器从输入的种子语料库开始，通过应用各种变异算子不断地对这些种子进行变异，将所生成的增加了神经元覆盖率的种子添加到种子语料库中。我们将种子被选择进行变异的概率称为种子的权重。将权重分配给种子的过程称为模糊器的权重分配。

算法 3.2 基于熵值引导的模糊测试

Input: $\mathcal{M}$:DNN model to be tested, C:Initial seed corpus,
MAX_LOOP:Max loop for the fuzzing campaign
Output: C:Augmented seed corpus

```
1:  initialize(C);
2:  for k from 1 to MAX_LOOP do
3:      for each t ∈ C do
4:          assignWeight(t);
5:      end for
6:      total = Σ_{t∈C} t.weight;
7:      for each t ∈ C do
8:          t.weight = t.weight / total;
9:      end for
10:     t = sample t from C with probability t.weight;
11:     t′ = metamorphicMutate(t);
12:     if M(t′) crashes or increases coverage then
13:         add t′ to C;
14:     end if
15:     for each covered elements i ∈ M execised by t′ do
16:         Y_i^t = Y_i^t + 1;
17:     end for
18: end for
19: return C;
```

算法 3.2展示了基于熵值引导的模糊测试的整个流程。在给定次数的循环中，模糊器以种子的归一化权重为概率分布来对种子 $t \in C$ 进行抽样（第 6-9 行）。种子的权重是使用 *assignWeight*() 方法计算的（第 4 行）。之后我们对种子 t 应用第 3.2节中的基于蜕变关系的图像变异算法来对其进行变异，以生成测试输入 t'（第 11 行）。如果对变异种子 t' 的执行错误（如分类错误、执行时间超出限制、内存/数值溢出等）或者增加了神经元覆盖率，则将 t' 添加到种子语料库中（第 13 行）。我们将对一个种子 $t \in C$ 进行变异而产生的属于物种 $\mathcal{D}_i$ 的输入数量称为 t 的局部物种数量 Y_i^t（第 16 行）。

基于熵的权重分配为那些揭示更多关于 DNN 的信息的种子分配了更多的权重。也就是说，模糊器会将大部分的执行时间分配给那些能够更加有效地触发 DNN 新行为的种子，从而提高模糊测试的效率。我们用种子的局部熵 H^t 来度量每个种子 t 的邻近物种的信息量。

在实验中，我们很快注意到不能直接使用式 3-16中的极大似然估计量 $\hat{H}_{MLE}$。一个从未经过变异的新种子 t 将总是被分配为零权重 $\hat{H}_{MLE} = 0$。因此，它永远不会被选择为变异的对象，直到整个模糊测试过程结束都会保持权重为 0。

为了解决这个问题，我们采用了贝叶斯方法。我们知道当所有可能的结果出现的概率都相等时，熵取得最大值。对于一个新的种子 t，我们假设一个对概率 p_i^t 的无信息先验，即 $p_1^t = \ldots = p_S^t$，其中 p_i^t 是对种子 t 进行变异生成的输入属于物种 $\mathcal{D}_i$ 的概率。随着对种子 t 的变异而新生成的输入不断增加，其相应的概率也不断地随之更新。后验是在 p_i^t 上的 Beta 分布，因此 p_i^t 的估计 $\hat{p}_i^t$ 是这个 Beta 分布的均值，也被称为拉普拉斯估计（Laplace Estimator）或加一平滑法（Add-one Smoothing），如式 3-17所示，其中 S 是全局的物种数量。

$$\hat{p}_i^t = \frac{Y_i^t + 1}{S + \sum_{j=1}^{S} Y_j^t} \tag{3-17}$$

因此，我们将改进后的熵估计 $\hat{H}_{LAP}^t$ 定义为式 3-18。

$$\hat{H}_{LAP}^t = \log\left(S + \sum_{j=1}^{S} Y_j\right) - \frac{\sum_{i=1}^{S}(Y_i + 1)\log(Y_i + 1)}{S + \sum_{j=1}^{S} Y_j} \tag{3-18}$$

第四章　熵值引导的模糊测试方法在工业缺陷图像扩增上的应用

除了神经元覆盖率以外，模型的准确性也是模型质量评估的一个重要标准。为了验证本文方法的实际应用效果，我们将熵值引导的图像变异方法与基于工业表面缺陷特征的图像扩增方法相结合，旨在增加工业缺陷数据集的数据量，丰富缺陷数据的多样性，提高检测模型的泛化能力。

4.1　工业表面缺陷检测

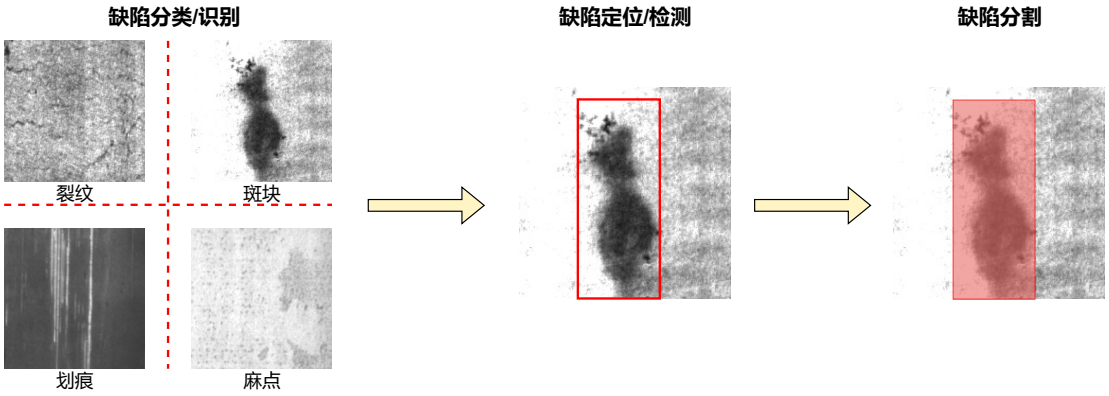
表面缺陷检测主要是检测物体表面局部的化学或者物理性质不均匀的区域，常见的表面缺陷有塑料或者金属制品表面的斑点、孔洞和划痕，玻璃等非金属制品表面的污点、杂质、平整度，纸制品表面的凸起、凹陷，纸张表面的污点、色差、破损等。

人工检测目前仍然是大部分工件制造企业进行表面缺陷检测的重要手段。质检工人为了提高检测效率通常使用显微镜来对工件表面的缺陷进行检测，然而显微镜的辅助效果有限，仅仅依靠人眼无法迅速准确地判断工件表面的缺陷状况。因此，人工检测法不仅会耗费较高的人力成本，而且检测效率偏低，还有较高的误检可能性。如今机器视觉技术日趋成熟，越来越多的公司开始应用机器视觉来进行表面缺陷检测。

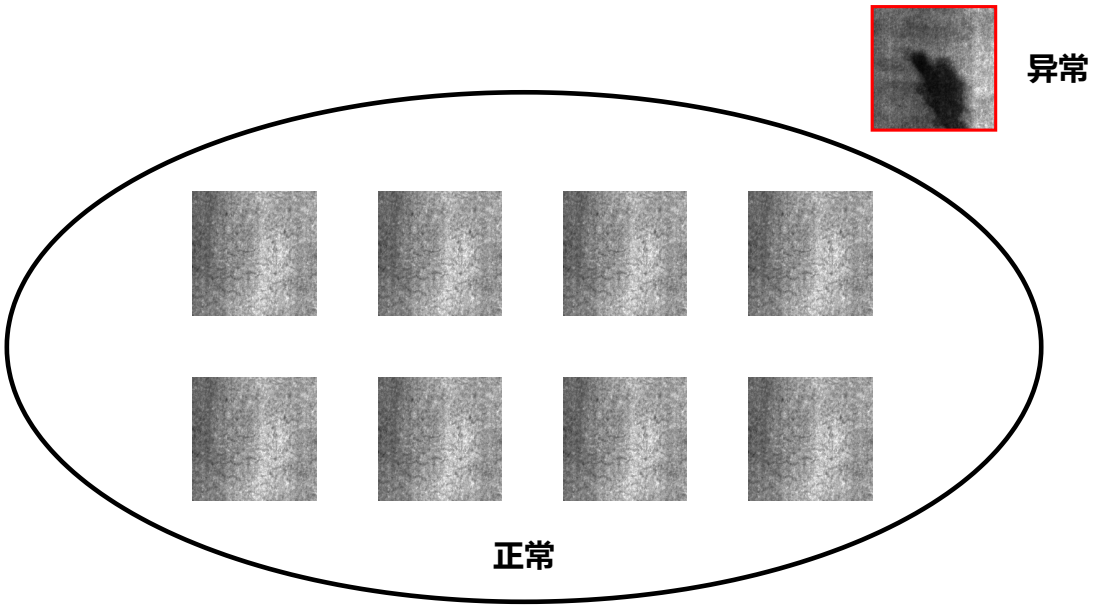
表面缺陷检测是机器视觉领域中的一项重要研究内容，也被称为 ASI（Automated Surface Inspection），是一种通过机器视觉设备来采集图像以判断获取的图像中是否存在缺陷的技术。传统的基于机器视觉的表面缺陷检测方法通常是采用常规图像处理算法或人工设计特征加分类器的方式实现的。一般情况下，根据被检测表面或缺陷的不同性质有针对性地设计成像方案，以此获得光照均匀的图像，并将被检测物体表面的缺陷明显地体现出来 [38]。

近年来，深度学习已经成功被应用于诸多计算机视觉领域，例如目标检测、人脸识别、自动驾驶和场景文字检测等，很多基于深度学习的缺陷检测方

法也被广泛应用于各种工业场景之中。本文将熵值引导的模糊测试方法与基于工业表面缺陷特征的图像数据扩增方法相结合，旨在增加工业缺陷数据集的数据量，丰富缺陷数据的多样性，提高模型的泛化能力。



(a) 有监督的缺陷检测方法



(b) 无监督的缺陷检测方法

图 4-1: 缺陷检测的问题定义

在机器视觉任务中，缺陷是一个由人类经验所定义的概念，并没有严谨的数学定义。不同的缺陷模式认知对应着不同的检测方法。以热轧带钢表面缺陷检测为例，图 4-1(a)展示了有监督的检测方法，将打过标签（包括缺陷类别、矩形框或逐像素等）的缺陷图像数据输入到神经网络中进行训练。此时“缺陷”意味着标记过的图像或者图像区域。因此，该方法更关注缺陷的特征，例如在训练阶段将包含大片深色范围的图像或者图像区域标记为“斑块”缺陷用

于网络的训练。在测试阶段，当网络检测到大片深色的特征时，即认为出现了“斑块”缺陷。

图 4-1(b)展示了无监督的缺陷检测方法，通常只将没有缺陷的正常样本数据输入网络进行训练，也被称为 One-class Learning。该方法更关注没有缺陷的正常样本的特征，当缺陷检测过程中发现没有见过的特征（即异常特征）时，就将其认定为缺陷。此时“缺陷”意味着异常，因此该方法也被称为异常检测（Anomaly detection）。

类似于计算机视觉中明确的分类、定位和分割任务，缺陷检测的需求可以划分为缺陷分类、缺陷定位和缺陷分割三个阶段。第一阶段的缺陷分类如图 4-1(a)中分类四种缺陷类别: 裂纹、斑块、划痕和麻点，仅仅给出缺陷的类别信息。第二阶段的缺陷定位才是严格意义上的缺陷检测，除了获取图像中存在的缺陷类型，同时也给出缺陷的具体位置，如图 4-1(a)中红色矩形框标记所示。第三阶段的缺陷分割将缺陷逐像素地从背景中分割出来，并进一步得到缺陷的面积、长宽等一系列信息，这些信息可以为产品的质量评估提供依据 [38]。

4.2 基于工业表面缺陷特征的图像数据扩增方法

工业表面缺陷检测的图像数据扩增方法可以分为两种类型。一种方法是对工业图像的采集过程进行模拟，通过改变图像变异算子的一些参数来模拟采集过程中的各种不同环境条件（如相机视角、光照条件等）来模拟生成在不同环境下采集到的图像数据，主要的变异算子包括图像的色彩变换、分辨率变换和视角变换。另一种方法是针对缺陷本身的形态特征进行模拟，通过对缺陷进行一些图像变换操作，在保持缺陷本身特征不变的基础上来模拟生成更多不同的缺陷图像数据，从而达到丰富图像数据的目的，主要的变异算子包括图像的方位变换、缩放变换、旋转变换和翻转变换。下面本文将对上述的所有图像变异算子进行详细介绍。

4.2.1 工业图像采集过程模拟扩增方法

（1）色彩变换

为了模拟真实检测环境中可能出现的不同光照条件，我们可以对缺陷图像进行色彩变换，包括色调、亮度和饱和度变换的叠加。

在 RGB 色彩空间中，每个彩色像素点可以用 RGB 坐标系统中从坐标原点延伸到该点的一个向量来进行表示，如图 4-2所示。

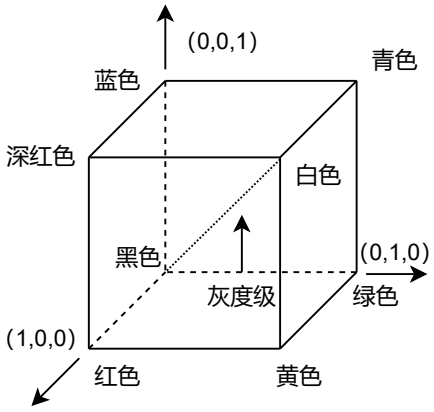


图 4-2: RGB 彩色立方体示意图

令 $\boldsymbol{c}$ 代表 RGB 色彩空间中任意的一个向量：

$$\boldsymbol{c} = \begin{bmatrix} c_{\boldsymbol{R}} \\ c_{\boldsymbol{G}} \\ c_{\boldsymbol{B}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{R} \\ \boldsymbol{G} \\ \boldsymbol{B} \end{bmatrix} \tag{4-1}$$

该式表明了 $\boldsymbol{c}$ 的分量只是一幅彩色图像在一个点处的 RGB 分量。我们可以认为彩色分量是坐标 (x, y) 的函数，表示为如下：

$$\boldsymbol{c}(x, y) = \begin{bmatrix} c_{\boldsymbol{R}(x,y)} \\ c_{\boldsymbol{G}(x,y)} \\ c_{\boldsymbol{B}(x,y)} \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}(x, y) \\ \boldsymbol{G}(x, y) \\ \boldsymbol{B}(x, y) \end{bmatrix} \tag{4-2}$$

对于大小为 $\boldsymbol{M} \times \boldsymbol{N}$ 的图像，有 $\boldsymbol{MN}$ 个这样的向量 $\boldsymbol{c}(x, y)$，其中 $x = 0, 1, 2, \ldots, \boldsymbol{M} - 1$；$y = 0, 1, 2, \ldots, \boldsymbol{N} - 1$。

图像的色彩变换的具体实现如算法 4.1所示。我们首先对参数进行前置校验（第 1 行），如果入参为空那么就为其赋一个给定范围内的随机值。接着计算图像的均值和方差（第 2 行），由于每个通道的像素值范围都是 $[0, 255]$，基于均值和方差的计算可以防止数值溢出。然后遍历图像的所有像素点分别对三个通道进行色彩变换（第 3-10 行），在变换后对新的值进行校验，防止溢出（第 9 行）。最后返回变换后的新图像。

算法 4.1 图像的色彩变换

Input: *defect_img*:Defect image, *light*:Image brightness,*saturation*:Image saturation,*tone_r*:Channel R, *tone_g*:Channel G, *tone_b*:Channel B
Output: *new_img*:new image

```
preCheck();
RGB_mean, RGB_var = calculate(defect_img);
for each i,j in defect_img.shape do
    for each tone,k in zip([tone_r, tone_g, tone_b], [0, 1, 2]) do
        new_mean = (1 + light) * RGB_mean + tone * RGB_mean;
        new_variance = (1 + saturation) * RGB_var;
        new_val = (defect_img[i][j][k] − RGB_mean)/RGB_var * new_variance +
        new_mean;
        new_img[i][j][k] = new_val;
        postCheck();
    end for
end for
return new_img;
```

（2）透视变换

在真实的检测环境中，采集图像的摄像头一般都是以固定的角度进行拍摄。如果摄像头的角度由于种种原因进行了调整或者出现了一些细微的偏差，都可能会影响到检测结果的准确性。为了模拟真实检测环境中摄像头角度的不同变化，我们可以对缺陷图像进行透视变换。

图像的透视变换是将原图像投影到一个新的视平面，也被称为投影映射。它是二维空间 (X, Y) 到三维空间 (X, Y, Z)，再到另一个二维空间 (X', Y') 的映射。透视变换相比于仿射变换具有更大的灵活性。图像的透视变换的通用变换公式如式 4-3所示：

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{4-3}$$

其中，设原目标点 $P(x, y)$ 在进行变换之前在三维空间中的坐标是 $(x, y, 1)$，在二维平面上的投影是 (x, y)，通过变换矩阵 M 变换成三维空间中的点 (X, Y, Z)，再通过除以三维空间中 Z 轴的值，转换成二维空间中的点 (x', y')，如式 4-4所示。

$$\begin{cases} x' = \frac{X}{Z} = \frac{m_{11}x+m_{12}y+m_{13}}{m_{31}x+m_{32}y+m_{33}} \\ y' = \frac{Y}{Z} = \frac{m_{21}x+m_{22}y+m_{23}}{m_{31}x+m_{32}y+m_{33}} \\ z' = \frac{Z}{Z} = 1 \end{cases} \tag{4-4}$$

令 $m_{33} = 1$，将式 4-4展开，得到一个点的情况：

$$\begin{cases} x' = m_{11}x + m_{12}y + m_{13} - m_{31}xx' - m_{32}yx' \\ y' = m_{21}x + m_{22}y + m_{23} - m_{31}xy' - m_{32}yy' \end{cases} \tag{4-5}$$

只需要 4 组点对就可以得到 8 个方程，即可求解出透视变换矩阵：

$$\begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -xx' & -yx' \\ 0 & 0 & 0 & x & y & 1 & -xy' & -yy' \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix} \begin{bmatrix} m_{11} \\ m_{12} \\ \vdots \\ m_{32} \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ \vdots \\ \vdots \end{bmatrix} \tag{4-6}$$

图像的透视变换的具体实现方法如算法 4.2所示。首先对参数进行前置校验（第 1 行）。然后扩展图像，保证图像内容不超出可视范围（第 2 行）。接着计算镜头与图像间的距离，为了保证在此可视角度下恰好显示整幅图像（第 3 行）。然后生成的透视变换矩阵并对图像进行透视变换（第 4-5 行）。之后重新生成对应的缺陷位置数据（第 6 行）。最后对生成的新图像进行剪裁等处理，返回生成的图像（第 7-9 行）。

算法 4.2 图像的透视变换

Input: *defect_img*:Defect image, *box*:Defect box
Output: *result*:new image, *new_boxes*:new defect boxes
1: *preCheck*();
2: Expand the image;
3: Calculate the distance between the lens and the image;
4: Generate perspective transformation matrices;
5: $result = cv2.warpPerspective(img, warpR, (h, w))$;
6: Generate new defect boxes;
7: $result = result[ymin : ymax, xmin : xmax]$;
8: $result = biLinearInterpolation(result, (ymax - ymin) * z/z\_old, (xmax - xmin) * z/z\_old)$;
9: **return** *result*, *new_boxes*;

（3）分辨率变换

工业相机是进行表面缺陷检测所不可或缺的重要设备。分辨率是相机最基本的一种参数，在很大程度上影响着采集图像的质量。更高的分辨率意味着在对同样大的视场成像时能够展示更多的细节。为了模拟真实检测环境中，相机的传感器故障、镜头污点等可能导致图像分辨率降低的问题，我们使用最常用的双线性插值算法来降低图像的分辨率。

线性插值在数学领域中是一种曲线拟合方法，它利用线性多项式来在已知数据点的离散集合范围内去构造新的数据点。已知两点 (x_0y_0) 和 (x_1y_1)，线性插值就是两点之间的直线，如图 4-3所示。

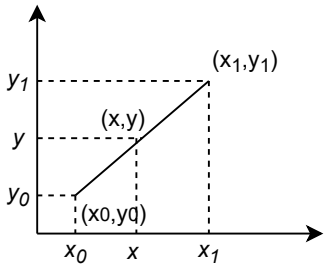


图 4-3: 线性插值法示意图

其中，直线 y 的方程如下：

$$y = \frac{x_1 - x}{x_1 - x_0} y_0 + \frac{x - x_0}{x_1 - x_0} y_1 \tag{4-7}$$

双线性插值是对线性插值的一种扩展，插值函数有两个变量，其核心思想是分别在两个方向进行一次线性插值。假设我们想得到未知函数 f 在点 $P = (x, y)$ 处的值，且已知函数 f 在 $Q_{11} = (x_1, y_1)$、$Q_{12} = (x_1, y_2)$、$Q_{21} = (x_2, y_1)$ 以及 $Q_{22} = (x_2, y_2)$ 四个点的值，如图 4-4所示

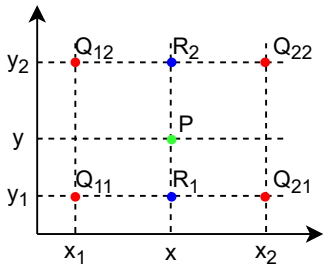


图 4-4: 双线性插值法示意图

首先，在 X 方向进行线性插值，即在 Q_{12}，Q_{22} 中插入蓝色点 $R_2(x, y_2)$，

Q_{11}，Q_{21} 中插入蓝色点 $R_1(x, y_1)$。然后继续在 Y 方向进行线性插值，通过上一步计算得到的 R_1 与 R_2 在 y 方向上的插值计算出 P 点。最终得到的插值公式如下所示。

$$
\begin{aligned}
f(R_1) &\approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}) \\
f(R_2) &\approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}) \\
f(P) &\approx \frac{y_2 - y}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2)
\end{aligned}
\tag{4-8}
$$

图像的分辨率变换的具体实现方法如算法 4.3所示。首先对参数进行前置校验（第 1 行）。然后通过双线性插值法对图像先进行放大再进行缩小（第 2-4 行），即可得到原图像大小的低分辨率版的缺陷图像。

算法 4.3 图像的分辨率变换

Input: *img*:Defect image
Output: *img_small*:new image
1: *preCheck*();
2: *img_large* = biLinearInterpolation(*img*, *img*.*shape*[0] ∗ *s*, *img*.*shape*[1] ∗ *s*);
3: *rows*, *cols*, *channels* = *img_large*.shape
4: *img_small* = biLinearInterpolation(*img_large*, *rows*/*s*, *cols*/*s*)
5: **return** *img_small*;

4.2.2 工业表面缺陷形态特征模拟扩增方法

相比于工业图像采集过程模拟扩增方法，工业表面缺陷形态特征模拟扩增方法更关注图像中缺陷本身的特征。因此，我们将缺陷从工件图像中分离出来，单独对缺陷部分的图像进行扩增处理，并使用泊松融合算法 [39] 将扩增后的缺陷图像融合进工件图像中，达到扩增缺陷图像数据集的目的。

本文使用 openCV 来实现图像融合，具体实现如算法 4.4所示。首先根据参数选择对应的融合方法（第 1-9 行），总共有三种融合方式法。*NORMAL_CLONE* 把待融合的图像完整地插入到目标图像之中，不会保留目标图像的纹理细节，目标区域的梯度只取决于源图像；*MIXED_CLONE* 相比 *NORMAL_CLONE* 会保留目标图像的纹理细节，对背景和轮廓特征实现透明通道混合，目标区域的梯度是由目标图像和源图像的组合计算得到的；*MONOCHROME_TRANSFER* 则是基于特征的迁移融合，只把特征融合到背

算法 4.4 图像融合

Input: *defect_img*:Defect image, *dst_img*:Destination image, *box*:Defect box, *merge_type*:Merge type
Output: *new_img*:New image

```
1: if merge_type == 'normal' then
2:     flags = cv2.NORMAL_CLONE;
3: end if
4: if merge_type == 'mixed' then
5:     flags = cv2.MIXED_CLONE;
6: end if
7: if merge_type == 'monochrome' then
8:     flags = cv2.MONOCHROME_TRANSFER;
9: end if
10: mask = 255 * np.ones(defect_img.shape, defect_img.dtype);
11: center = (int(box[0] * dst_img.shape[1]), int(box[1] * dst_img.shape[0]));
12: new_img = cv2.seamlessClone(defect_img, dst_img, mask, center, flags);
13: return new_img;
```

景图像之中，不保留源图像的颜色细节，只保留源图像的质地。之后根据参数生成目标图像上的感兴趣区域，即缺陷所在的区域（第 10 行）。然后生成目标图像的中心在背景图像上的坐标（第 11 行），进行图像融合（第 12 行）。

算法 4.5 色彩变换

Input: *defect_img*:Defect image, *light*:Image brightness,*saturation*:Image saturation,*tone_r*:Channel R, *tone_g*:Channel G, *tone_b*:Channel B
Output: *new_img*:new image

```
1: preCheck();
2: RGB_mean, RGB_var = calculate(defect_img);
3: for each i,j in defect_img.shape do
4:     if is_defect(i, j) then
5:         for each tone,k in zip([tone_r, tone_g, tone_b], [0, 1, 2]) do
6:             new_mean = (1 + light) * RGB_mean + tone * RGB_mean;
7:             new_variance = (1 + saturation) * RGB_var;
8:             new_val = (defect_img[i][j][k] − RGB_mean)/RGB_var * new_variance +
               new_mean;
9:             new_img[i][j][k] = new_val;
10:            postCheck();
11:        end for
12:    end if
13: end for
14: return new_img;
```

下面介绍工业表面缺陷形态特征模拟扩增方法。

（1）色彩变换

与工业图像采集过程模拟扩增方法中的色彩变换类似，这里的色彩变换只针对工件中所分离出来的缺陷本身进行色彩变换。具体实现方法如算法 4.5所示。在进行色彩变换前会先判断像素位置是否位于缺陷区域内（第 4 行），只对缺陷区域进行色彩变换。

（2）位置变换

工件表面的缺陷位置往往都是无规律的，没有固定的位置。方位变换可以改变缺陷在工件表面上的位置，模拟工件表面上随机出现的缺陷，从而达到扩增缺陷图像数据集的目的。具体实现方法如算法 4.6所示。首先随机生成变换后的缺陷中心坐标点，生成的同时确保整个缺陷图像在原工件图像的范围内（第 1-2 行）。接着计算变换位置后新缺陷的四个顶点的坐标（第 3-6 行），并根据新生成的缺陷坐标来生成新的图像（第 7-8 行）。

算法 4.6 位置变换

Input: $defect\_img$:Defect image, box:Defect boxes,
$defect\_w$:Width of defect, $defect\_h$:Height of defect
Output: new_img:New image

1: $centerx = random.randint(ceil(defect\_w/2), floor(width - defect\_w/2))$;
2: $centery = random.randint(ceil(defect\_h/2), floor(height - defect_h/2))$;
3: $xmin = int(centerx - defect\_w/2)$;
4: $ymin = int(centery - defect\_h/2)$;
5: $xmax = int(centerx + defect\_w/2)$;
6: $ymax = int(centery + defect\_h/2)$;
7: Add *xmin*,*ymin*,*xmax*,*ymax* to *new_box*;
8: Generate new image by *new_box*;
9: **return** *new_img*;

（3）缩放变换

工件表面的缺陷除了位置不固定以外，其形状和大小往往也具有一定的随机性，因此缩放变换可以改变缺陷在工件表面上的大小，从而达到扩增缺陷图像数据集的目的。具体实现方法如算法 4.7所示。首先对缺陷图像进行缩放变换（第 1 行）。接着对缩放后的图像进行校验（第 2 行），如果缩放后的缺陷图像大小超过了原工件图像的范围，那么需要对其进行重新缩放（第 3-5 行）。最后将缩放变换后的缺陷图像与原工件图像融合后生成新的图像。

算法 4.7 缩放变换

Input: $defect\_img$:Defect image, fx:Scale in X-axis, fy:Scale in Y-axis
Output: new_img:New image

```
1: res = cv2.resize(defect_img, None, fx, fy, cv2.INTER_CUBIC);
2: if postCheck() then
3:     fx_new = obj_img_width/res.shape[1] − 0.07;
4:     fy_new = obj_img_height/res.shape[0] − 0.07;
5:     res = cv2.resize(res, None, fx_new, fy_new, cv2.INTER_CUBIC);
6: end if
7: Generate new image by res;
8: return new_img;
```

算法 4.8 旋转变换

Input: $defect\_img$:Defect image, $angle$:Rotation angle, $flip$:Mirror flip
Output: new_img:New image

```
1: preCheck();
2: heightNew = int(defect_img.shape[0] ∗ fabs(sin(radians(angle))) + height ∗
   fabs(cos(radians(angle))));
3: widthNew = int(defect_img.shape[1] ∗ fabs(sin(radians(angle))) + width ∗
   fabs(cos(radians(angle))));
4: matRotation = cv2.getRotationMatrix2D((defect_img.shape[1]/2 ,
   defect_img.shape[0]/2) , angle , 1);
5: matRotation[0, 2]+ = (widthNew − defect_img.shape[1])/2;
6: matRotation[1, 2]+ = (heightNew − defect_img.shape[0])/2;
7: res = cv2.warpAffine(defect_img, matRotation, (widthNew, heightNew),
   borderValue = (255, 255, 255));
8: if postCheck() then
9:     fx_new = obj_img_width/res.shape[1] − 0.1;
10:    fy_new = obj_img_height/res.shape[0] − 0.1;
11:    res = cv2.resize(res, None, fx = fx_new, fy = fy_new, interpolation =
       cv2.INTER_CUBIC);
12: end if
13: if flip then
14:    res = np.fliplr(defect_img);
15: end if
16: Generate new image by res;
17: return new_img;
```

（4）旋转变换

除了缺陷的位置与大小以外，旋转变换可以改变缺陷的形状，从而达到扩增缺陷图像数据集的目的。具体实现方法如算法 4.8所示。首先对参数进行前置校验（第 1 行）。然后计算得到旋转矩阵（第 2-4 行）。图像旋转后一部分

图像会丢失，边缘也会产生黑边，把旋转后的图像放置白色背景中，防止丢失像素（第 5-6 行）。对图像进行旋转变换（第 7 行）后，对旋转后的图像进行后置校验，如果变换后的图像超过原工件的尺寸，则对其进行缩小（第 8-12 行）。根据参数还可以对缺陷图像进行镜像翻转（第 13-14 行）。最后将旋转变换后的缺陷图像与原工件图像融合后生成新的图像。

第五章　实验设计与分析

5.1　实验一：神经元覆盖率水平评估

5.1.1　实验目的

本实验的目的是为了评估基于变异图像熵值引导的神经网络模糊测试方法的神经元覆盖率水平。为了更加直观地反映本方法的效果，我们将DeepHunter [16] 作为本实验的对照方法。DeepHunter 是一个基于神经元覆盖率引导的模糊测试框架，使用一种基于种子模糊处理次数的频率感知的种子选择策略，并利用多个可扩展的覆盖标准作为反馈来引导测试用例的生成。本实验希望通过与 DeepHunter 的对比来回答以下两个研究问题：

- **RQ1**：基于变异图像熵值引导的神经网络模糊测试方法能够达到什么样的神经元覆盖率水平？
- **RQ2**：与基于覆盖率引导的神经网络模糊测试框架 DeepHunter 相比，基于变异图像熵值引导的神经网络模糊测试方法是否能够更加有效地提高神经元覆盖率？

5.1.2　实验对象

本实验的主要对象是深度神经网络（DNN），而不同 DNN 模型之间的结构可能各不相同，包括中间层数量以及神经元数量等各种参数可能都存在着较大的差异。为了使实验结果更具有普遍性，本实验使用不同的流行公开数据集在多个 DNN 模型上分别进行测试。

本实验选用的数据集是当前较为流行的 2 个公开数据集 MNIST [40] 和 CIFAR10 [41]。

（1）**MNIST** 数据集

MNIST 数据集是一个在深度学习领域中被使用最多、最广为人知的入门级计算机视觉数据集，它包含了各种 0 ~ 9 的手写数字图像，每个图像的尺寸都是 28 * 28 像素，如图 5-1所示。我们可以很容易地判断出是数字 5，4，0，9，

表 5-1: 实验数据集和模型神经元数量统计表

数据集	DNN 模型	隐藏层神经元数量	网络层数
MNIST	**Lenet-1**	16	6
	Lenet-5	226	8
CIFAR-10	**VGG-16**	5248	16
	VGG-19	13696	19

每个图片都有其对应的数字标签。MNIST 数据集中包含了 60000 个训练数据集的图像、5000 个验证数据集的图像以及 10000 个测试数据集的图像。

图 5-1: MNIST 数据集示例图

对于 MNIST 数据集，我们在 2 个模型上进行了实验，分别是手写字体识别模型 Lenet-1 和 Lenet-5 [42]。每个模型的网络层数和隐藏层（除了输入层、输出层以及池化层外）神经元数量如表 5-1所示。

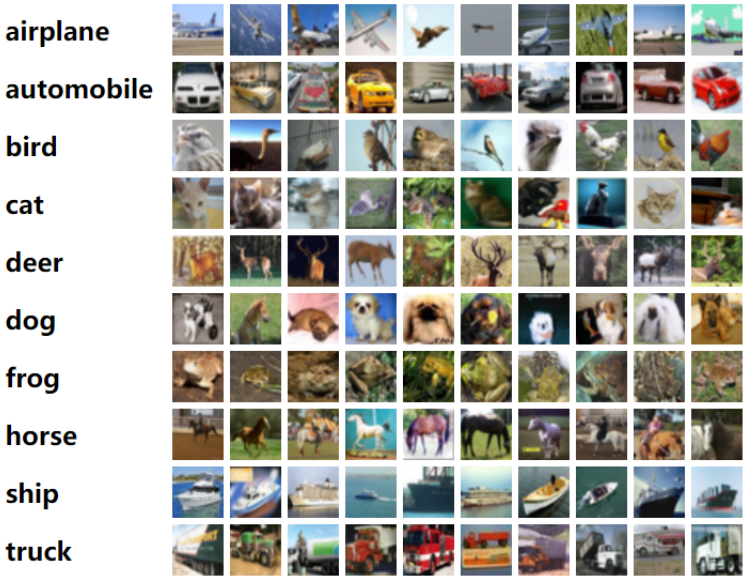


图 5-2: CIFAR-10 数据集示例图

（2）**CIFAR-10** 数据集

CIFAR-10 数据集包含了总共 60000 张 32 * 32 像素的彩色图像，分为 10 个类别，包括鸟类、汽车、飞机、轮船等等，每一类包含 6000 个图像，这 10 个类是各自独立的，互不重叠的，如图 5-2所示。CIFAR-10 数据集中包含 50000

个训练数据集的图像，构成 5 个训练批次，每个批次有 10000 个图像，其余的 10000 个图像用于测试数据集。测试数据集中的数据是从 10 个类中的每一类中随机取 1000 张，剩下的就随机排列组成了训练数据集。

对于 CIFAR-10 数据集，我们在 VGGNet 模型 [43] 上进行了实验。VGGNet 成功地构筑了 16 ~ 19 层深的卷积神经网络，与之前的最先进的神经网络结构相比，大幅度提高了准确率。本实验使用的模型是 VGG-16 和 VGG-19，其网络层数以及隐藏层神经元个数如表 5–1所示。

5.1.3 实验指标

为了回答 RQ1 和 RQ2，需要计算相关指标来对结果进行评估和展示。本实验参考 DeepGauge [12] 中的覆盖标准作为实验指标来进行计算和评估。DeepGauge 是一套为深度学习系统设计的多粒度测试标准。给定一组输入，DeepGauge 的标准可以度量它对主功能区域和边界区域的神经元的覆盖程度。他们的实验结果表明，目前的大部分测试数据对主功能区域的覆盖程度远大于边界区域。实验工作表明，DeepGauge 的标准有着更高的覆盖范围，可以有效地捕获原始测试数据和对抗样本之间的差异，能够更有效地检测 DNN 的缺陷，有助于从不同层次和角度理解 DNN 以及测试数据质量。

为了更好地与 DeepHunter 进行对比，本实验选取 DeepGauge 中的 k-多段神经元覆盖率（KMNC）、神经元边界覆盖率（NBC）、强激活神经元覆盖率（SNAC）、top-k 神经元覆盖率（TKNC）这 4 个神经元覆盖标准以及最基础的神经元覆盖率（NC）作为评估指标。关于各个神经元覆盖率指标的详细定义可以参照第 2.2节。

5.1.4 实验设计

为了回答 **RQ1** 和 **RQ2**，我们以神经元覆盖标准和种子选择策略作为控制变量进行了大规模的对照实验，具体实验配置如表 5–2所示。其中种子选择策略包括本文的基于熵值引导的方法（以下简称 DeepEntropy）、DeepHunter 以及随机方法（Random）作为对照，神经元覆盖标准包括 KMNC、TKNC、SNAC、NBC 以及 NC。

我们所使用的环境为：windows 11 + Anaconda3 4.9.2 + CUDA 11.3 +Python 3.6.13 + Tensorflow-gpu 2.5.0 + Opencv-python 3.4.3.18。对于每组配置的实验，

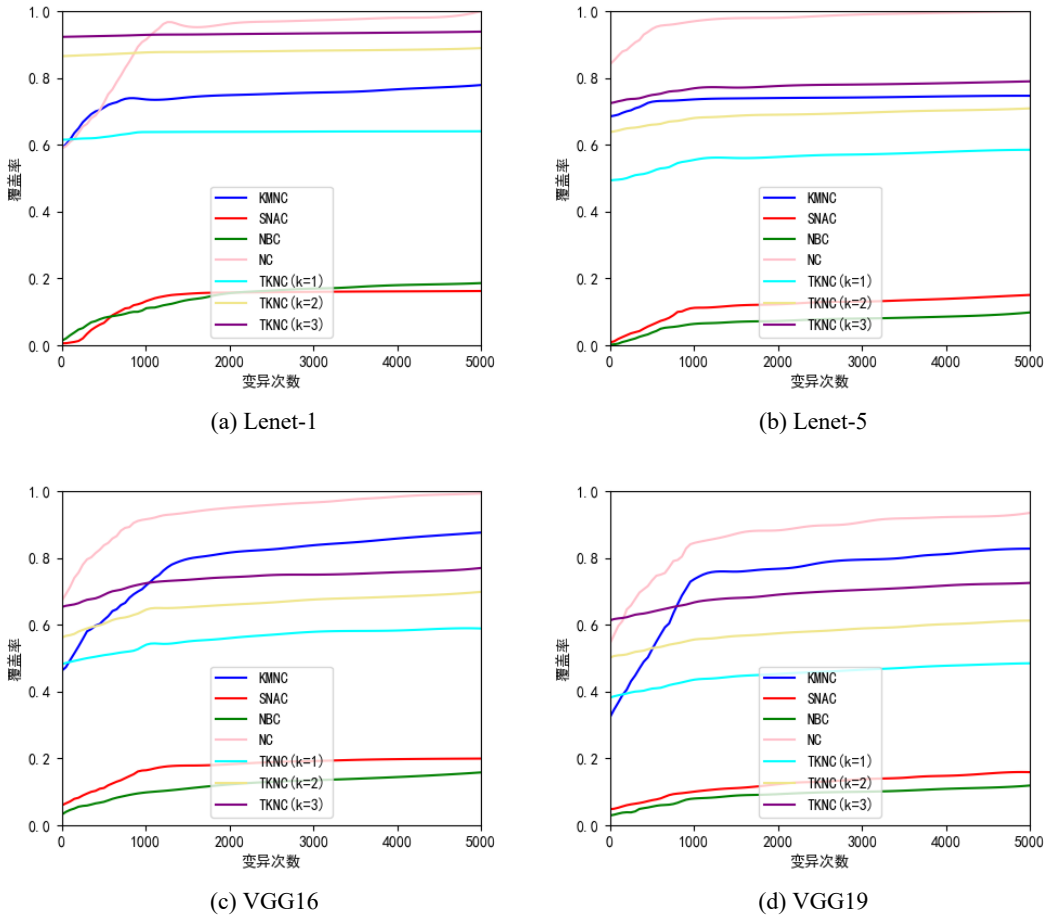
我们首先从原始测试数据集中随机选取 1000 个图像作为初始种子并对其进行预处理放入种子队列。接着我们对种子按照不同的策略进行权重分配，并以权重为概率来选取种子执行相同次数的变异（5000 次），记录程序的执行情况并计算相关指标作为反馈来引导后续变异的种子权重分配。为了避免模糊测试过程中的偶然性，我们将每组实验重复执行 10 次，并取平均值作为最终结果。

5.1.5 实验结果与分析

图 5-3显示了 DeepEntropy 在四个不同模型上的覆盖率变化趋势，其中每个图的横坐标是种子的变异总次数，纵坐标是覆盖率。对于神经元覆盖率 NC，DeepEntropy 在四个模型都可以达到比较高的覆盖率水平，其中 Lenet-1、Lenet-5 和 VGG16 都达到了接近 100% 的覆盖率水平。对于 k-多段神经元覆盖率 KMNC，DeepEntropy 在 Lenet-1 和 Lenet-5 上的提升幅度较小，最终达到 77% 左右的水平，而在网络结构更加复杂、神经元数量更多的 VGG16 和 VGG19 上提升幅度更加明显，可以达到 87% 左右的水平。对于 top-k 神经元覆盖率 TKNC，DeepEntropy 在 k=1,2,3 时在四个模型上的增长趋势都较为平缓，可以提高 1 ~ 5 个百分点。对于神经元边界覆盖率 NBC 和强激活神经元覆盖率 SNAC，DeepEntropy 在四个模型上的覆盖率水平都处于较低的水平，且增长曲线都比较平缓。

从图 5-3中我们可以很容易看出，不同覆盖率准则的变化趋势在相同模型的执行中存在着明显的差异，而提高不同覆盖准则的覆盖率的难度也是不同的。例如 NC 和 KMNC 的上升趋势明显要快于其他覆盖准则，只需要 1000 次左右的变异就可以达到接近峰值的水平，随后增速放缓。其他覆盖准则的变化趋势是随着变异次数的增加缓慢而稳定的上升，直到迭代结束。对于 NBC 和 SNAC 来说，它们代表的是边界情况，因为这两个覆盖准则是度量模型在测试运行过程中神经元的激活值高于训练集上边界值或低于下边界值的数量比例，而神经元激活值超出边界值的情况发生概率较低，因此覆盖率一直处于较低水平，更难覆盖。基于上述讨论和图 5-3中的数据指标，我们可以对 **RQ1** 作出回答：基于变异图像熵值引导的神经网络模糊测试方法能够达到较高的神经元覆盖率水平，相比于原始数据集，可以在一定程度上有效地提高神经元覆盖率。

表 5-2显示了不同配置下各个覆盖率标准的平均结果。每个模型的第一行显示了初始状态的种子所达到的覆盖率，粗体数字表示使用不同策略的各个覆盖标准的最佳情况。从表 5-2中可以很直观地看出，与无引导的随机测试相



(a) Lenet-1 (b) Lenet-5

(c) VGG16 (d) VGG19

图 5-3: 不同模型的覆盖率变化趋势

比，由覆盖率引导的 DeepHunter 和由熵值引导的 DeepEntropy 都能够更加有效地提高覆盖率。在绝大部分情况下，由熵值引导的 DeepEntropy 的覆盖率水平都优于 DeepHunter。

对于 Lenet-1 和 Lenet-5，DeepHunter 的神经元覆盖率 NC 均达到了 100%，略高于 DeepEntropy。除此之外，DeepHunter 的其他指标的覆盖率水平均低于 DeepEntropy，总体上差距较小，都在 3% 以内。对于 VGG16，DeepEntropy 除了 KMNC 略低于 DeepHunter 以外，其余指标的覆盖率水平都要比 DeepHunter 高出 1 ~ 6 个百分点。对于 VGG19，DeepEntropy 除了神经元覆盖率 NC 略低于 DeepHunter 以外，其他指标的覆盖率水平都比 DeepHunter 高接近 1 ~ 5 个百分点。

综合上述讨论和表 5-2 中的实验数据，我们可以对 **RQ2** 作出回答：与基于覆盖率引导的神经网络模糊测试框架 DeepHunter 相比，基于变异图像熵值引

表 5-2: 不同配置的模糊测试在 10 次运行中的平均覆盖率（%）

Model	Strategy	KMNC	TKNC			SNAC	NBC	NC
			k=1	k=2	k=3			
Lenet-1	Init.	58.90	61.52	86.51	92.28	0.57	1.36	76.59
	DeepHunter	74.45	63.49	88.52	92.90	14.12	16.96	**100**
	DeepEntropy	**77.87**	**64.03**	**88.91**	**93.24**	**16.23**	**18.60**	99.95
	Random	69.16	62.86	87.63	92.57	12.16	13.92	96.37
Lenet-5	Init.	68.47	49.31	63.79	72.41	0.85	0.14	84.02
	DeepHunter	71.91	54.95	66.54	74.47	10.23	7.41	**100**
	DeepEntropy	**74.16**	**56.14**	**68.81**	**76.37**	**13.05**	**8.82**	99.96
	Random	70.49	53.27	65.81	74.29	7.39	4.14	90.62
VGG16	Init.	46.19	48.16	56.13	65.29	5.94	3.16	67.12
	DeepHunter	**87.85**	52.76	64.28	70.24	16.10	11.26	98.81
	DeepEntropy	87.12	**58.27**	**68.49**	**75.71**	**19.95**	**15.81**	**99.30**
	Random	81.28	51.29	64.33	69.28	8.72	7.28	90.32
VGG19	Init.	32.19	38.61	50.13	61.29	4.80	2.91	54.31
	DeepHunter	81.65	43.64	58.19	68.47	11.23	8.63	**92.43**
	DeepEntropy	**82.77**	**48.74**	**60.35**	**72.52**	**14.98**	**10.02**	92.03
	Random	72.54	43.82	54.81	66.43	6.81	5.97	85.27

导的神经网络模糊测试方法在一定程度上能够更加有效地提高神经元覆盖率。

5.2 实验二：熵值引导的工业表面缺陷图像扩增效果评估

5.2.1 实验目的

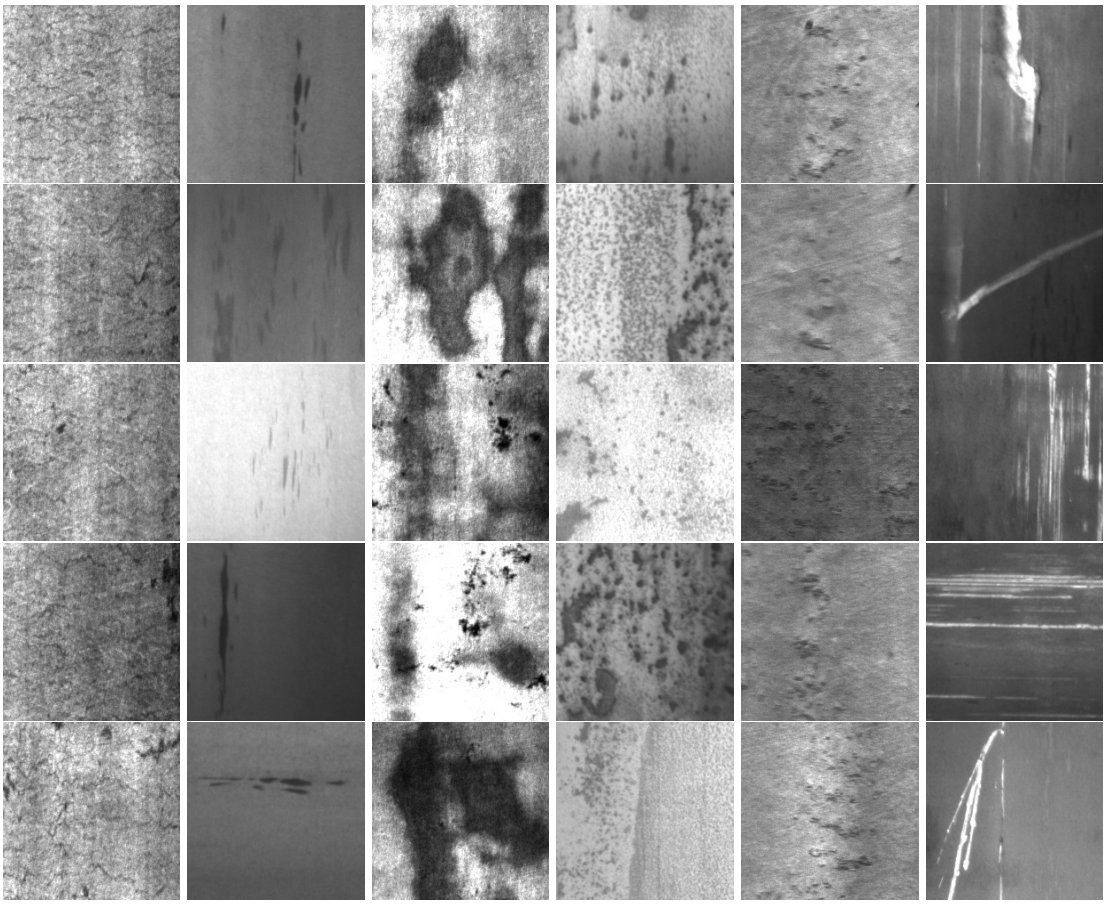
本实验的目的是为了评估将变异图像熵值引导的图像变异方法应用于工业表面缺陷图像扩增的效果，即测试和评估本文的方法是否能够应用到实际场景中，帮助到检测工人或者测试人员。基于实验二的主要目的，我们提出了下面两个研究问题：

- **RQ3**：基于变异图像熵值引导的工业表面缺陷图像扩增方法能否提高缺陷检测模型的覆盖率？
- **RQ4**：基于变异图像熵值引导的工业表面缺陷图像扩增方法能否提高缺陷检测的准确率？

5.2.2 实验对象

本实验使用东北大学的开源数据集 NEU-DET 和 Yolov5 目标检测算法作为实验对象进行图像数据扩增以及网络结构的训练，将 4.1节中介绍的图像扩增方法作用于待处理图像最终得到不同情况下的缺陷图像数据，并使用 Yolov5 进行训练和测试，评估最终的效果。

NEU-DET [44] 是东北大学的开源数据集，收集了热轧带钢的夹杂、划痕、氧化铁皮压入、裂纹、麻点和斑块 6 种典型的表面缺陷。该数据集总共有 1800 个灰度图像，6 种不同类型的典型表面缺陷各 300 个样本，图像尺寸为 $200 * 200$。



(a) 裂纹 (b) 夹杂 (c) 斑块 (d) 麻点 (e) 氧化铁皮压入 (f) 划痕

图 5-4: NEU-DET 数据集示例图

图 5-4显示了六种典型表面缺陷的样本图像，每个图像的原始分辨率为 $200 * 200$ 像素。从图 5-4中可以观察到，即使是同种类别之间的缺陷也可能有着很大的形态差异，例如，图 5-4(f)中的划痕方向并没有明确规律，水平、垂

直等各种方向的划痕均有出现的可能性。同时，不同类别的缺陷也可能有着类似的外观形态，例如氧化铁皮压入、裂纹和麻点。此外，光照和材料变化可能会导致缺陷图像的灰度值发生改变。总而言之，NEU-DET 数据集包含两个难题，即同一类别内缺陷存在较大的外观差异，而不同类别之间的缺陷具有相似的方面，缺陷图像受到照明和材料变化的影响。

Yolov5 [①]是由 Ultralytics LLC 团队推出基于 Yolov4 [45] 的目标检测网络模型。相比于 Yolov4，Yolov5 有着更快的识别速度和更高的检测精度。Yolov5 可以在 Tesla P100 上实现 140 帧/秒的快速检测，可以满足工业生产线上的实时检测需求。同时 Yolov5 中体积最小的权重文件只有 27MB 大小，因此该模型能够更加方便、简单地移植到工业生产线中。本实验使用 Yolov5s 来对图像数据集进行训练和检测。

5.2.3 实验准备

由于本实验使用 Yolov5 目标检测算法进行训练和检测，而 NEU-DET 也是 Yolo 格式的数据集，因此我们需要保证扩增后的缺陷数据也符合 Yolo 格式 [45]，才能够顺利进行后续的训练与测试。对 Yolo 模型进行训练需要.JPG 格式的图像数据，同时还需要.txt 格式的标签数据文件，其文件名前缀与对应的图像名前缀是相同的。与图 5-5对应的标签文件包含两个人（类别序号 0）和一条领带（类别序号 27）。

Yolo 格式的.txt 标签数据文件格式要求如下：

- 每行一个检测目标（Object）。
- 每行的数据格式为 *class x_center y_center width height*，如图 5-5所示。
- 锚框坐标必须是归一化的 *xywh* 格式（从 0 到 1）。如果锚框以像素为单位，则将 *x_center* 和 *width* 除以图像宽度，将 *y_center* 和 *height* 除以图像高度。
- 类别序号由 0 开始索引。

5.2.4 实验指标

为了回答 **RQ3**，我们仍然使用实验一所提到的覆盖率指标，包括 KMNC、NBC、SNAC、TKNC 以及 NC 作为本实验的神经元覆盖率评估指标。

[①]https://github.com/ultralytics/yolov5

图 5-5: Yolo 格式标签示意图

为了回答 **RQ4** 还需要引入缺陷检测的一些经典概念。下面介绍一些缺陷检测相关的指标。

（1）交并比 **IoU**

交并比（Intersection over Union，IoU）是目标检测中常用的一个概念，它定义了由网络生成的锚框与原标记锚框的重合度，即它们之间交集与并集的比值。给定区域 A 和 B，那么交并比的计算公式可以定义为如下：

$$IoU = \frac{A \cap B}{A \cup B} \tag{5-1}$$

（2）**GIoU**

GIoU [46] 即泛化版 IoU，在保持了 IoU 的主要性质的同时避免了 IoU 的缺点。给定区域 A 和 B，令 C 为能够包含它们的最小区域，那么 GIoU 的计算公式定义为如下：

$$GIoU = IoU - \frac{|C/(A \cup B)|}{|C|} \tag{5-2}$$

（3）平均精确度 **mAP**

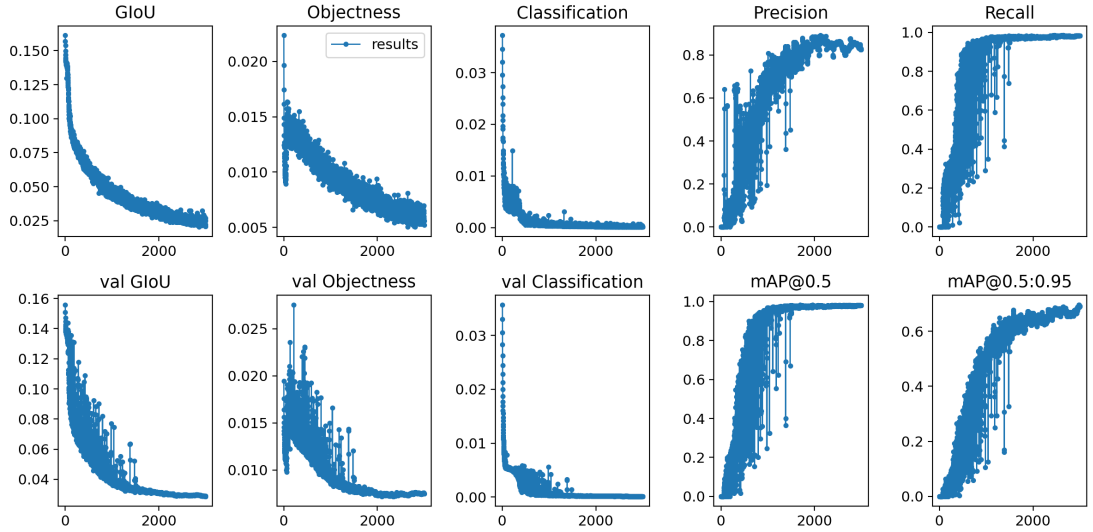
mAP（mean Average Precision）是一个在目标检测中被广泛使用的检测精度衡量指标。在目标检测任务中，可以根据设定的 IoU 阈值来判断模型的预测结果是否为正样本。通常情况下 IoU 阈值被设定为为 0.5，通过设定不同的 IoU 阈值，就可以得到多组召回率与准确率。在多目标检测中，以准确率为纵坐标，召回率为横坐标，对于每一个类别都可以绘制一条 Precision-Recall 曲线，常见的评价指标 AP 就是 Precision-Recall 曲线下的面积。该面积越大，AP 值就越大，算法性能也就越好，检测目标也越精确。mAP 是所有类别 AP 值的平均值，它反映了模型在所有类别上的召回率与准确率。常见的 mAP 指标有 mAP@0.5，即当设置 IoU 为 0.5 时的平均准确率，以及 mAP@0.5:0.95，即在不同 IoU（从 0.5 到 0.95，步长 0.05）设置下的平均值 [47]。

5.2.5 实验设计

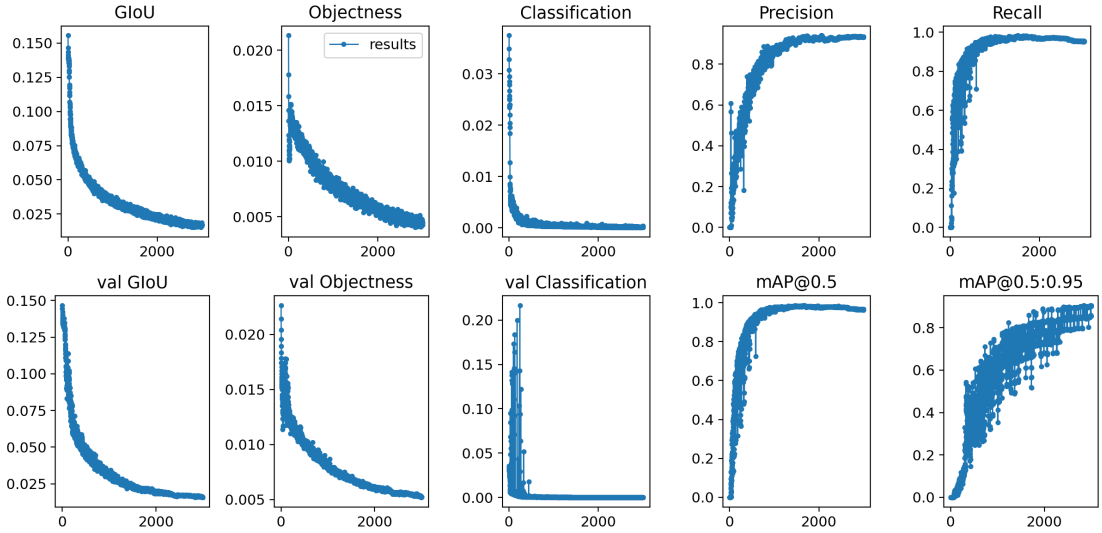
本实验中，我们所使用的环境为：windows 11 + Anaconda3 4.9.2 + CUDA 11.3 + Opencv-python 4.1.2.30 + Python 3.8.5 + Pytorch 1.10.2。为了回答 **RQ3**，我们使用熵值引导的工业表面缺陷扩增方法对 NEU-DET 数据集进行了数据扩增，最终生成 5000 张不同异常情况下的缺陷图像数据，同时我们还使用随机方法对 NEU-DET 数据集同样生成了 5000 张缺陷图像数据作为实验对照，并将缺陷图像数据输入 Yolov5s 模型进行覆盖率统计。为了回答 **RQ4**，我们使用 Yolov5 对原始数据集和扩增后的数据集进行训练和对比，并记录测试过程中的程序执行情况。其中对于 Yolov5 的训练参数，我们将 batch_size 设置为 8，epoch 设置为 3000，并使用 yolov5s.yaml 作为训练模型进行训练与测试，最后记录并观察测试结果，对测试结果进行汇总和统计，绘制相关图表，并对实验结果进行分析和总结。

5.2.6 实验结果与分析

表 5-3显示了 Yolov5s 在经过图像扩增前后的覆盖率情况，第一行显示了初始状态的种子所达到的覆盖率，粗体数字表示使用不同策略的各个覆盖标准的最佳情况。从表 5-3中我们可以很明显地观察到，相比于初始状态的数据集和经过随机方法扩增后的数据集，经过熵值引导的图像数据扩增后的数据集在各个覆盖率指标上都有着较大的提升，并且在所有覆盖率指标上都要优于随机



(a) 原始数据集的训练结果



(b) 扩增后数据集的训练结果

图 5-6: 数据集扩增前后模型训练结果对比

方法的扩增。其中 NC 和 KMNC 的提升最为明显，比初始状态分别提高了接近 26 和 45 个百分点，比随机方法扩增分别提高了接近 6 和 8 个百分点。对于 SNAC、NBC 和 TKNC 也都有比较明显的提升。基于上述讨论和表 5-3中的数据，我们可以对 **RQ3** 作出回答：基于变异图像熵值引导的工业表面缺陷图像扩增方法可以在一定程度上有效地提高缺陷检测模型的覆盖率。

图 5-6展示了检测模型在使用扩增前后的数据集经过 3000 轮迭代训练后的结果对比。从图 5-6中我们可以观察到，相比于原始数据集训练的模型，经过熵值引导的图像数据扩增后的数据集所训练后的模型的指标均有所提

表 5-3: Yolov5s 模型扩增前后的覆盖率（%）

Model	Strategy	KMNC	TKNC			SNAC	NBC	NC
			k=1	k=2	k=3			
Yolov5s	Init.	38.87	41.49	53.18	62.37	4.64	3.49	75.42
	DeepEntropy	**83.74**	**54.23**	**68.24**	**78.81**	**15.67**	**18.60**	**91.59**
	Random	75.42	48.37	60.27	68.76	9.49	10.41	85.61

升，其中 GIoU Loss 从原来的 0.013 降低到了 0.009，精度从原来的 0.895 提高到了 0.956，mAP@0.5:0.95 从原来的 0.784 提高到了 0.924。基于上述讨论和图 5-6所展示的数据，我们可以对 **RQ4** 作出回答：基于变异图像熵值引导的工业表面缺陷图像扩增方法可以在一定程度上有效地提高缺陷检测的准确率。

5.3 本章小结

在本章中我们主要进行了两个实验，以分别探讨基于变异图像熵值引导的神经网络模糊测试方法的有效性以及其在实际应用场景中的可行性。在实验一中，我们以覆盖率引导的模糊测试框架 DeepHunter 作为对比方法，通过分析实验结果，验证了本方法的有效性。在实验二中，我们将变异熵值引导的图像变异方法应用于工业表面缺陷检测中，并使用扩增后的数据集进行了训练与测试，实验结果表明，本方法可以在一定程度上有效提高缺陷检测的准确率，具有实际应用价值。

第六章　总结与展望

6.1　本文研究内容总结

近年来，深度学习已经在许多前沿的智能应用中取得了巨大成功，人们也看到了将基于深度神经网络的软件应用于自动驾驶等安全关键场景的巨大潜力。与传统软件类似，DNN 可能会因为隐藏的缺陷而表现出不正确的行为，从而导致严重的事故和损失。为了保障使用深度神经网络的安全关键应用能够可靠运行，需要对其进行充分的测试。学术界已经提出了基于神经元覆盖率引导的模糊测试技术，神经元覆盖率指标需要人工设定阈值，而不同神经元的输出值在统计分布上有着较大差异。在同一组测试集中，不同层或者不同区域的神经元的输出可能具有不同的平均值和方差，人工设定的阈值可能会对测试结果造成一定的影响。因此，如果使用神经元覆盖率来引导模糊测试，可能无法保证测试效果的稳定性，需要一种更加有效的指标来对神经网络的模糊测试进行引导，从而提高模糊测试的效率，增强深度学习系统的健壮性。

熵可以度量一个事件的自信息量，如果大多数测试生成的输入都触发了以前未有的程序行为，那么输出的熵值就会很高，测试的效率也就更高。本文首次提出了基于变异图像熵值引导的神经网络模糊测试技术，引入熵值来对种子进行权重分配，引导变异图像的生成，通过最大化种子的熵值来最大化测试的效率，以揭示更多关于 DNN 的信息。本文与基于覆盖率引导的神经网络模糊测试框架 DeepHunter 进行了对比实验，实验结果表明，本文的基于变异图像熵值引导的神经网络模糊测试方法在神经元覆盖率、k-多段神经元覆盖率、top-k 神经元覆盖率、神经元边界覆盖率、强激活神经元覆盖率五个覆盖率指标上都有一定程度的提高，相比于 DeepHunter 能够更加有效地提高覆盖率。

为了探索本文方法的实际应用效果，本文将熵值引导与工业表面缺陷检测的领域知识相结合，在东北大学的开源工业表面缺陷数据集 NEU-DET 上使用变异图像熵值引导的图像扩增方法以及随机方法作为对比进行了图像数据扩增，并使用 Yolov5 模型进行训练与测试验证。实验结果表明，扩增后的数据集所训练的模型相比于随机方法扩增的数据集所训练的模型在准确率、

mAP@0.5、mAP@0.5:0.95 等目标检测的主要评估指标上都有一定程度的提高，将熵值引导的图像变异方法应用于工业表面缺陷图像扩增可以有效地提高缺陷数据集质量和提升缺陷检测的准确率，具有一定的实际应用价值。

6.2 本文工作内容局限与展望

由于时间以及个人水平的限制，在本文的工作中仍存着在一些不足之处有待在未来的相关研究中进行改进：

1. 在本文的研究工作中所涉及到的深度学习模型以及数据集都是基于图像数据的，而本文的方法对于基于音频、文本等其他形式的数据的深度神经网络模型并不适用，在适用范围上具有一定的局限性。因此如果对种子的变异方法进行拓展，使其能够支持更多形式的数据作为种子，就可以进一步提高本文方法的适用性。
2. 对于深度学习模型测试的评估指标，本文只选取了 5 个目前较为主流和常用的指标，未来可以引入更多学术界所提出的覆盖指标以从更丰富的角度来测试和评估模型的质量。
3. 本文的图像变异方法不能保证在所有情况下生成的变异图像的有效性，未来可以拓展变异策略以及改进变异图像之间的蜕变关系来提高变异图像的有效性。
4. 本文将熵值引导的图像扩增方法应用于工业缺陷检测，由于个人电脑设备硬件的性能限制，只使用了 Yolov5 模型来进行训练与测试，单一的深度神经网络本身具有一定的局限性。因此未来可以将本文的方法应用在更多的模型上以及更多应用领域中进行探索与验证。

致　　谢

时光荏苒，两年的研究生生活即将进入尾声，回首过去的这两年美好时光，我在此向所有关心、鼓励、支持和帮助过我的每一个人表示最真挚的感谢。

桃李不言，下自成蹊。首先，我要感谢我的导师陈振宇老师和房春荣老师。在两年的研究生生涯中，两位老师都给予了我很多指导和帮助，在我论文选题和写作的过程中提出了很多宝贵的建议。iSE 实验室的各位老师们在为我们的学术研究和工程实践创造良好学习环境的同时也为我们树立了很好的榜样。

其次，我要感谢毕设期间帮助过我的所有同学和朋友。感谢实验室的章许帆学长，给我提供论文选题思路。感谢和我一起讨论论文进度以及为我的论文实验提供帮助的同学，让我能够顺利进行论文的写作。

感谢南京大学为我们提供美丽的校园和生活环境，感谢软件学院所有任课老师的教导，感谢辅导员老师们的辛苦付出。我会一直以南京大学为傲，诚朴雄伟，励学敦行。

感谢我的女朋友，很幸运能够认识你，给我鼓励和支持，陪伴我度过了最后一段美好的校园时光。

最后，感谢我的父母和家人们一直以来对我的支持和鼓励，陪伴我走过漫漫求学路。家永远是温暖的港湾，无论身在何方，我都会永远爱着我的家人。

我的求学生涯到此结束了，幼时遥不可及的未来如今已经到来。希望走上社会的未来的自己，不要被生活磨平棱角，保持积极乐观的人生态度，不忘初心，方得终始。

参考文献

[1] HUANG X, KROENING D, RUAN W, et al. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability[J]. Computer Science Review, 2020, 37.

[2] SU T, WU K, MIAO W, et al. A survey on data-flow testing[J]. ACM Computing Surveys (CSUR), 2017, 50.

[3] JIA Y, HARMAN M. An analysis and survey of the development of mutation testing[J]. IEEE transactions on software engineering, 2010, 37.

[4] WALTON G H, POORE J H. Generating transition probabilities to support model-based software testing[J]. Software: practice and experience, 2000, 30(10): 1095 – 1106.

[5] DIAS NETO A C, SUBRAMANYAN R, VIEIRA M, et al. A survey on model-based testing approaches: a systematic review[C] // Proceedings of the 1st ACM international workshop on Empirical assessment of software engineering languages and technologies: held in conjunction with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE) 2007. 2007: 31 – 36.

[6] 钱航. 深度神经网络测试覆盖率与对抗样本间的相关性研究 [D]. [S.l.]: 南京邮电大学, 2020.

[7] CHANG O, METZMAN J, MOROZ M, et al. OSS-Fuzz: Continuous Fuzzing for Open Source Software[J]. URL: https://github. com/google/ossfuzz, 2016.

[8] BöHME M, PHAM V-T, ROYCHOUDHURY A. Coverage-Based Greybox Fuzzing as Markov Chain[J]. IEEE Transactions on Software Engineering, 2019, 45.

[9] ODENA A, OLSSON C, ANDERSEN D, et al. Tensorfuzz: Debugging neural networks with coverage-guided fuzzing[C] // International Conference on Machine Learning. 2019.

[10] HAYHURST K J. A practical tutorial on modified condition/decision coverage[M]. [S.l.] : DIANE Publishing, 2001.

[11] SUN Y, HUANG X, KROENING D, et al. Testing deep neural networks[J]. arXiv preprint arXiv:1803.04792, 2018.

[12] MA L, JUEFEI-XU F, ZHANG F, et al. Deepgauge: Multi-granularity testing criteria for deep learning systems[C] // Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. 2018 : 120 – 131.

[13] MA L, ZHANG F, XUE M, et al. Combinatorial testing for deep learning systems[J]. arXiv preprint arXiv:1806.07723, 2018.

[14] PEI K, CAO Y, YANG J, et al. Deepxplore: Automated whitebox testing of deep learning systems[C] // proceedings of the 26th Symposium on Operating Systems Principles. 2017 : 1 – 18.

[15] TIAN Y, PEI K, JANA S, et al. Deeptest: Automated testing of deep-neural-network-driven autonomous cars[C] // Proceedings of the 40th international conference on software engineering. 2018 : 303 – 314.

[16] XIE X, MA L, JUEFEI-XU F, et al. DeepHunter: a coverage-guided fuzz testing framework for deep neural networks[C] // Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2019, Beijing, China, July 15-19, 2019. [S.l.] : ACM, 2019 : 146 – 157.

[17] MCCULLOCH W S, PITTS W. A logical calculus of the ideas immanent in nervous activity[J]. The bulletin of mathematical biophysics, 1943.

[18] 李岱. 基于可视化解释的深度神经网络研究与应用 [D]. [S.l.] : 电子科技大学, 2021.

[19] LECUN Y, OTHERS. Generalization and network design strategies[J]. Connectionism in perspective, 1989.

[20] PRESSMAN R S. Software engineering: a practitioner's approach[M]. [S.l.]: Palgrave macmillan, 2005.

[21] RUPARELIA N B. Software development lifecycle models[J]. ACM SIGSOFT Software Engineering Notes, 2010.

[22] 王赞, 闫明, 刘爽, et al. 深度神经网络测试研究综述 [J]. 软件学报, 2020, 31: 1255 – 1275.

[23] CHEN T Y, KUO F-C, LIU H, et al. Metamorphic testing: A review of challenges and opportunities[J]. ACM Computing Surveys (CSUR), 2018: 1 – 27.

[24] HAMLET R. Random testing[J]. Encyclopedia of software Engineering, 1994: 971 – 978.

[25] ZHU H, HALL P A, MAY J H. Software unit test coverage and adequacy[J]. Acm computing surveys (csur), 1997, 29(4): 366 – 427.

[26] CADAR C, SEN K. Symbolic execution for software testing: three decades later[J]. Communications of the ACM, 2013, 56(2): 82 – 90.

[27] HARMAN M, JIA Y, ZHANG Y. Achievements, open problems and challenges for search based software testing[C] // 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST). 2015: 1 – 12.

[28] ROSENBLUM D S. A practical approach to programming with assertions[J]. IEEE transactions on Software Engineering, 1995, 21(1): 19 – 31.

[29] MANOLACHE L, KOURIE D G. Software testing using model programs[J]. Software: Practice and Experience, 2001, 31(13): 1211 – 1236.

[30] 路晓丽, 董云卫. 软件测试实践教程 [M]. 北京: 机械工业出版社, 2010.

[31] CHEN T Y, CHEUNG S C, YIU S M. Metamorphic testing: a new approach for generating next test cases[J]. arXiv preprint arXiv:2002.12543, 2020.

[32] 徐黎明, 刘航江. 数字图像处理技术研究综述 [J]. 软件导刊, 2016: 181 – 182.

[33] COVER T M. Elements of information theory[M]. [S.l.]: John Wiley & Sons, 1999.

[34] SHANNON C E. A mathematical theory of communication[J]. The Bell system technical journal, 1948, 27(3): 379–423.

[35] 刘华文. 基于信息熵的特征选择算法研究 [D]. [S.l.]: 吉林大学, 2010.

[36] 刘霞. 基于尺度不变与视觉显著特征的图像感知哈希技术研究 [D]. [S.l.]: 西南大学, 2015.

[37] BÖHME M, MANÈS V J, CHA S K. Boosting fuzzer efficiency: An information theoretic perspective[C] // Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2020: 678–689.

[38] 陶显, 侯伟, 徐德. 基于深度学习的表面缺陷检测方法综述 [J]. 自动化学报, 2021, 47: 1017–1034.

[39] PÉREZ P, GANGNET M, BLAKE A. Poisson image editing[G] // ACM SIGGRAPH 2003 Papers. 2003: 313–318.

[40] LECUN Y. The MNIST database of handwritten digits[J]. http://yann. lecun. com/exdb/mnist/, 1998.

[41] RECHT B, ROELOFS R, SCHMIDT L, et al. Do CIFAR-10 classifiers generalize to CIFAR-10?[J]. arXiv preprint arXiv:1806.00451, 2018.

[42] LECUN Y, JACKEL L, BOTTOU L, et al. Comparison of learning algorithms for handwritten digit recognition[C] // International conference on artificial neural networks: Vol 60. 1995: 53–60.

[43] SIMONYAN K, ZISSERMAN A. Very deep convolutional networks for large-scale image recognition[J]. arXiv preprint arXiv:1409.1556, 2014.

[44] HE Y, SONG K, MENG Q, et al. An End-to-End Steel Surface Defect Detection Approach via Fusing Multiple Hierarchical Features[J]. IEEE Transactions on Instrumentation and Measurement, 2020, 69(4): 1493–1504.

[45] BOCHKOVSKIY A, WANG C-Y, LIAO H-Y M. Yolov4: Optimal speed and accuracy of object detection[J]. arXiv preprint arXiv:2004.10934, 2020.

[46] REZATOFIGHI H, TSOI N, GWAK J, et al. Generalized intersection over union: A metric and a loss for bounding box regression[C] // Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2019: 658–666.

[47] 张涤天. 基于深度学习的工业图像缺陷检测 [D]. [S.l.]: 苏州大学, 2020.