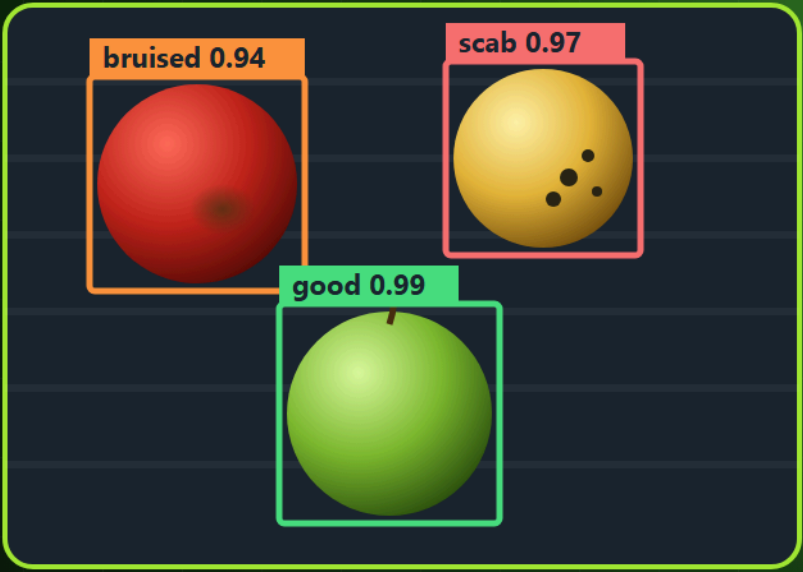


Confusion matrix

good	150	0	0
bruis.	12	78	0
scab	2	0	58

recall: bruised 0.867

rows = true, cols = predicted

```python
class AppleNet(nn.Module):
  def forward(self, x): # (N, 3, 64, 64)
    return self.classifier(self.features(x))

label, conf, lane = decide_lane(probs)
# ('bruised', 0.85, 'juice')
torch.onnx.export(model, (x,), "applenet.onnx")
```

A HANDS-ON GUIDE WITH PYTORCH

Computer Vision with Deep Learning

Image classification, object detection, segmentation, and deployment

CNNs · Augmentation · Transfer Learning · Vision Transformers
Faster R-CNN · U-Net · Grad-CAM · Calibration · ONNX · Serving

PIXELS · MODELS · BOXES · MASKS · PRODUCTION

THENEXBOOKS

Computer Vision with Deep Learning

Image Classification, Object Detection, Segmentation, and Deployment with PyTorch
A hands-on guide — from pixels to a working grading system

First Edition

THENEXBOOKS

Computer Vision with Deep Learning
Image Classification, Object Detection, Segmentation, and Deployment with PyTorch

First Edition, 2026

Copyright © 2026 thenexbooks (thenexbooks.gumroad.com). All rights reserved.

No part of this publication may be reproduced, distributed, or transmitted in any form or by any means without the prior written permission of the publisher, except for brief quotations in reviews and certain other non-commercial uses permitted by copyright law. Code examples may be used freely in your own projects.

The information in this book is provided "as is" without warranty of any kind. While every precaution has been taken in its preparation, the publisher assumes no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein. Libraries such as PyTorch, torchvision, Hugging Face Transformers, ONNX, and ONNX Runtime change between releases; APIs, defaults, and outputs can differ in the version you use. Training results vary between machines even with fixed random seeds. Pretrained model weights are distributed under their own licenses and terms; check them before any commercial use. A model that grades synthetic images well is not proof that it will work on real produce: validate any vision system on your own data before relying on it.

PyTorch is a trademark of The Linux Foundation. ONNX is a trademark of The Linux Foundation. Python is a trademark of the Python Software Foundation. Hugging Face is a trademark of Hugging Face, Inc. NVIDIA and CUDA are trademarks of NVIDIA Corporation. Windows is a trademark of the Microsoft group of companies. Other product and company names are trademarks of their respective owners. Their use in this book is for identification only and does not imply endorsement. This book is independent and is not affiliated with, sponsored by, or endorsed by any of these organizations.

Greenvale Orchards, the apple packhouse used as the running example in this book, is fictional, as are its people, apple varieties (Valecrest, Goldmere, Fernhill), packing lines, and business rules. All images in the book's dataset are synthetic, drawn by a seeded generator printed in the book; they are not photographs of real fruit.

Preface

Cameras are cheap, and pictures are everywhere: on packing lines, in warehouses, in clinics, on phones. What used to be missing was software that can look at a picture and reliably say what is in it. Deep learning changed that. A modest neural network, trained on examples instead of programmed with rules, can now sort products, find defects, count objects, and outline regions in an image — often well enough to run in production on an ordinary computer.

Getting there is not magic, though. A vision system is a pipeline: images become tensors, tensors flow through a model, the model's outputs become decisions, and every step has traps. Images get normalized twice. Augmentation creates pictures the real camera could never take. A model scores 95% on the test set and fails the day a new lamp is installed. An exported model gives slightly different answers from the one you trained. This book teaches the whole pipeline, with the traps, by building one complete system step by step.

We write every model in PyTorch and torchvision, run every example on a plain CPU, and show the real output we got. Where something needs a GPU or a package we did not run, we say so.

Who This Book Is For

- **Python developers** who want to add image understanding to their applications and need a practical, end-to-end path.
- **Data scientists and ML practitioners** who know tabular machine learning and want to move into images, detection, and segmentation.
- **Engineers in manufacturing, agriculture, logistics, and quality control** who are evaluating or building camera-based inspection.
- **Students** who want a hands-on companion to a computer vision course, with code that runs on a laptop.

You should be comfortable with basic Python: functions, classes, lists and dictionaries, installing packages, and running scripts. We recap the deep-learning basics we need (tensors, layers, losses, optimizers) as we go, always in a vision context.

What You Will Learn

- How images are stored and represented as tensors, and how to build a reproducible image dataset.
- Classic image processing — filters, thresholds, morphology — and why a good rule-based baseline matters.
- Convolutional neural networks from first principles, trained from scratch with honest evaluation: confusion matrices, per-class precision and recall, and decision thresholds.
- Data augmentation that matches how your images are really made — and how the wrong augmentation hurts.

- Modern architectures, transfer learning, Vision Transformers, and pretrained foundation models for embeddings and zero-shot classification.
- Object detection with Faster R-CNN, including IoU, non-maximum suppression, and mean average precision computed by hand.
- Semantic segmentation with a U-Net and a pretrained model, and measuring defect area from masks.
- Explainability with Grad-CAM, robustness to distribution shift, calibration, label noise, and reproducible experiments.
- Deployment: ONNX export, ONNX Runtime, quantization, honest benchmarking, a prediction API, and monitoring in production.

How This Book Is Organized

Part I — Foundations (Chapters 1–5) introduces computer vision, sets up the workbench, treats images as tensors, builds the Greenvale dataset, and creates a rule-based baseline. **Part II — Image Classification** (Chapters 6–10) covers data pipelines, convolutions, our first CNN, augmentation, and evaluation. **Part III — Modern Architectures and Pretrained Models** (Chapters 11–15) covers training techniques, architecture families, transfer learning, Vision Transformers, and foundation models. **Part IV — Detection and Segmentation** (Chapters 16–20) finds and outlines apples and defects. **Part V — Trust and Data Quality** (Chapters 21–24) covers explainability, robustness, calibration, label quality, and reproducibility. **Part VI — Deployment and Production** (Chapters 25–28) exports, speeds up, serves, and monitors the grader. The appendices provide a cheat sheet, templates and reference code, a production checklist, and a glossary.

Each chapter ends with exercises under **Try It Yourself** and a **Key Takeaways** box for quick review.

The Running Example: Greenvale Orchards

Throughout the book we build the vision system for **Greenvale Orchards**, a fictional family-run apple orchard with its own packhouse. Apples roll along a conveyor belt under a camera, and every apple must be graded as **good**, **bruised**, or **scab** and sent to the right lane: fresh pack, juice, or processing — or to a manual check when the system is unsure. Ingrid, the operations manager, owns the business rules; Dario, the ML engineer, writes the code; Lisbet, the quality lead, labels images and checks the doubtful apples.

Because real fruit photos come with licensing and privacy questions and cannot be reproduced exactly, the Greenvale images are **synthetic**: a short, seeded Python generator — printed in the book — draws apples of three varieties with realistic shading, natural marks, bruises, and scab spots, plus whole trays with bounding boxes and pixel masks. Every number in the book can be reproduced on your own machine. Synthetic apples are simpler than real ones, and we say so wherever it matters.

Versions and Conventions

Examples were tested with **Python 3.12**, **PyTorch 2.14** and **torchvision 0.29** (CPU builds), recent versions of Hugging Face Transformers, ONNX, and ONNX Runtime, on Windows. They are written to run on Linux and macOS as well, and on a GPU when one is available, but every result shown was produced on a CPU. Training numbers can differ slightly on your machine; deterministic facts such as shapes and parameter counts should match exactly.

> **TIP**
> A recommended practice or a technique that makes your work easier.

> **NOTE**
> Background information, version differences, or a useful aside.

> **WARNING**
> A common mistake or something that can silently break your results.

Code and file names appear in `monospace`; long lines are wrapped to fit the page. A caption above a code block names the file it belongs to, and blocks captioned *Output* show what the code printed on our machine. Commands are written for a terminal opened in the project folder `greenvale-vision`.

Contents

Preface

PART I — FOUNDATIONS

1 What Computer Vision Is and How Deep Learning Changed It

2 Setting Up a Vision Workbench: Python, PyTorch, torchvision, and the Greenvale Project

3 Images as Tensors: Pixels, Channels, Color, and File Formats

4 Building the Greenvale Apple Dataset: A Seeded Synthetic Image Generator

5 Classic Image Processing: Filters, Thresholds, and a Rule-Based Baseline

PART II — IMAGE CLASSIFICATION

6 Datasets, DataLoaders, and Transforms for Images

7 Convolutions from First Principles

8 Your First CNN: Training AppleNet to Grade Apples

9 Data Augmentation That Matches the Packhouse

10 Evaluating Image Classifiers: Confusion Matrices, Per-Class Metrics, and Thresholds

PART III — MODERN ARCHITECTURES AND PRETRAINED MODELS

11 Making CNNs Train Well: Batch Norm, Residual Connections, and Regularization

12 A Tour of Modern Architectures: ResNet, MobileNet, EfficientNet, and ConvNeXt

13 Transfer Learning: Feature Extraction and Fine-Tuning

14 Vision Transformers: Patches, Attention, and When to Use Them

15 Vision Foundation Models: Embeddings, Similarity Search, and Zero-Shot Classification

PART IV — DETECTION AND SEGMENTATION

16 Object Detection Concepts: Boxes, IoU, Anchors, and Non-Maximum Suppression

17 Training a Detector: Fine-Tuning Faster R-CNN on Greenvale Trays

18 Evaluating Detectors: Precision, Recall, and Mean Average Precision

19 Semantic Segmentation: A U-Net for Defect Masks

20 Pretrained Segmentation, Instance Masks, and Measuring Defect Area

PART V — TRUST AND DATA QUALITY

21 Explainability: Saliency Maps, Grad-CAM, and Checking What the Model Sees

22 Robustness: Distribution Shift, Calibration, and Confidence Thresholds

23 Data-Centric Vision: Labeling, Label Noise, Class Imbalance, and Active Learning

24 Experiments and Reproducibility: Seeds, Checkpoints, Tracking, and Model Cards

PART VI — DEPLOYMENT AND PRODUCTION

25 Exporting Models: torch.export, ONNX, and ONNX Runtime

26 Fast Inference on CPU and Edge Devices: Quantization, Batching, and Honest Benchmarks

27 Serving the Grader: A Prediction API and the Packing Line

28 Monitoring and Capstone: The Greenvale Grading System End to End

APPENDICES

A Computer Vision Cheat Sheet

B Templates and Reference

C Production Checklist

D Glossary

PART I

Foundations

What computer vision is and what deep learning changed, a working PyTorch setup, images as tensors, a seeded synthetic dataset of Greenvale apples, and a hand-written rule-based grader that every model in this book must beat.

1 · What Computer Vision Is

2 · Setting Up a Vision Workbench

3 · Images as Tensors

4 · Building the Greenvale Apple Dataset

5 · Classic Image Processing

CHAPTER 1 · PART I — FOUNDATIONS

What Computer Vision Is and How Deep Learning Changed It

Computer vision turns pixels into decisions. In this chapter we map out the main vision tasks, see why learned features replaced hand-written rules, meet the apple packhouse that will be our running example, and lay out the path the rest of the book follows.

A Camera Above a Conveyor Belt

Picture a long, narrow room that smells of fresh fruit. Apples tumble out of wooden bins onto a conveyor belt, roll forward in a single file, and pass under a camera mounted in a small hood with two lamps. A few meters further on, the belt splits into lanes. Some apples go to the boxes that will end up on supermarket shelves. Others go to a juice press, and others to a line that makes sauce and dried slices. Somebody, or something, has to decide which lane each apple takes, and it has to decide quickly, because the next apple is already right behind it.

For many years that somebody was a row of people standing beside the belt, picking out bad fruit by eye. People are good at this. They notice a soft brown patch that means the apple was dropped, a cluster of dark, rough spots that means a fungal infection, or a stem that has been torn out. They also tire, and they disagree on borderline cases. The camera is there because the packhouse would like a second pair of eyes that never gets tired and grades every apple by the same rule.

That camera produces nothing but numbers: a grid of pixels, each with a red, green, and blue intensity. Getting from that grid to "this apple is bruised, send it to juice" is what **computer vision** is about. This book teaches you to build that path yourself with deep learning and PyTorch, one working program at a time, using a single realistic project from the first chapter to the last.

What Computer Vision Is

Computer vision is the part of computing that extracts useful information from images and video. "Useful" depends entirely on the job. A phone camera wants to know where the faces are so it can focus on them. A warehouse wants to read the label on a parcel. A hospital might want an outline of an organ in a scan. A packhouse wants to know whether each apple is fit for the fresh market.

It helps to sort the most common jobs into a small number of **tasks**. Each task has a characteristic output, a characteristic kind of label you must collect to train it, and a characteristic way of measuring success. When you understand which task you are really solving, a large part of the design follows automatically.

Image classification

Classification answers the question "what is in this image?" with one label from a fixed list. The model looks at the whole picture and produces a score for every class; the highest score wins. At Greenvale Orchards, our running example, the camera captures one apple per image, and the classifier chooses between three classes: **good**, **bruised**, and **scab**. Classification is the simplest vision task to label (one word per image), the simplest to evaluate (did it pick the right word?), and the place where most people start. Part II of this book is devoted to it.

Classification assumes there is one main subject per image. If a single photo shows a whole tray of apples, some good and some bad, a single label cannot describe it. That is where the next two tasks come in.

Object detection

Detection answers "what is in this image, and where?" The output is a list of **bounding boxes**, each with a class label and a confidence score. A detector looking at a tray of apples might report three boxes: a good apple at the top left, a bruised one in the middle, and a scab apple near the bottom. Labels are more expensive to collect, because someone has to draw a box around every object, and evaluation is subtler, because a box can be almost right. We build and evaluate a detector in Chapters 16 to 18.

Segmentation

Segmentation answers "which pixels belong to what?" **Semantic segmentation** gives every pixel a class, such as background, apple skin, bruise, or scab, without separating one apple from another. **Instance segmentation** also separates the individual objects, so you know that these pixels belong to the second apple and those to the third. Segmentation is the right tool when the size or shape of a region matters. If Greenvale's buyers ever say "a bruise smaller than a fingernail is acceptable," you need to measure the bruise, and a box will not do that. Chapters 19 and 20 cover segmentation.

A comparison for the packing line

The table below shows how the same physical situation, one apple with a bruise lying on the belt, looks through the lens of each task.

Task	Question answered	Output for one bruised apple	Label you must collect
Classification	What is it?	"bruised" plus three class probabilities	One word per image
Detection	What and where?	A box around the apple, labeled bruised, with a score	A box and a class per object
Semantic segmentation	Which pixels are what?	A mask: background, skin, bruise pixels	A class for every pixel

Instance segmentation	Which pixels belong to which object?	A separate mask per apple plus its class	A mask per object

Notice the trade-off running down the rows. Each task gives you richer information than the one above it, and each costs more to label and more to compute. A recurring theme of this book is choosing the cheapest task that actually answers the business question. Greenvale's single-file conveyor shows one apple at a time, so classification is the natural starting point; trays with several apples make detection and segmentation worth their cost later on.

Other vision tasks, briefly

Classification, detection, and segmentation are the core of this book, but they are not the whole field. You will meet these other tasks in practice, and many of them reuse the same building blocks:

- **Keypoint detection** finds specific points, such as the joints of a human body for pose estimation or the corners of a document for straightening a scan.
- **Tracking** follows objects from one video frame to the next, so a system knows that the apple seen now is the same apple seen a moment ago. On a conveyor it is the glue between a detection and a mechanical action further down the belt.
- **Optical character recognition (OCR)** reads printed or handwritten text from images, such as lot numbers on crates.
- **Image retrieval and similarity search** finds images that look like a query image. We use this in Chapter 15 to "find apples like this one."

The good news is that the skills transfer. A backbone network that has learned to recognize apple skin for classification is the same kind of network that sits at the heart of a detector or a segmentation model. Once you understand images as tensors, convolutions, training loops, and evaluation, the specialized tasks are variations on a theme rather than new subjects.

From Hand-Crafted Rules to Learned Features

Computer vision is much older than deep learning. For decades, engineers built vision systems by writing the rules themselves. To understand why deep learning changed the field so thoroughly, it helps to see what that older approach looks like, because it is still useful and because we will build one ourselves in Chapter 5.

The classic pipeline

A hand-crafted vision system usually follows a chain of steps. First you clean up the image: smooth away noise, correct the brightness. Then you separate the object from the background, perhaps by noticing that apples are more colorful (more saturated) than a grey conveyor belt. Then you measure things inside the object: how many pixels are much darker than the surrounding skin, how many

separate dark spots there are, how round the outline is. Finally, a set of thresholds turns those measurements into a decision. "If more than a small percentage of the apple is dark, it is defective; if the dark area is split into several small spots, call it scab; otherwise call it a bruise."

Over the years researchers designed very clever general-purpose measurements, called **features**: descriptions of edges, corners, gradients, and textures that could be computed for any image and fed into a classifier. Systems built this way read postal codes, inspected circuit boards, and counted cars. In a controlled setting, with fixed lighting, a fixed camera, and a narrow job, they can work well, and they have real advantages: they are fast, transparent, and need few or no labeled examples.

Where rules break

The weakness of hand-written rules is that every one of them encodes an assumption about the world, and the world keeps changing. A rule that says "dark pixels are defects" has to cope with the natural dark stem cavity at the top of every apple. A rule that separates apple from belt by color saturation works until someone replaces the grey belt with a tan one. A threshold tuned under one set of lamps quietly fails under warmer ones. Each fix adds another rule, each rule interacts with the others, and before long nobody is quite sure why the system makes the decisions it does.

We will see this happen to Greenvale in concrete numbers. In Chapter 5 we build a careful rule-based grader, tune it honestly on validation data, and it reaches a respectable test accuracy. Then we point it at images from a second packing line with a different camera, and it collapses. That collapse is the clearest possible motivation for the rest of the book.

Learning the features instead

Deep learning flips the arrangement. Instead of a person designing the measurements and a simple model combining them, a large neural network learns the measurements and the decision together, directly from labeled examples. You show it thousands of images with the right answers, it adjusts millions of internal numbers to reduce its mistakes, and along the way its early layers end up detecting edges and color changes, its middle layers detect textures and small shapes, and its later layers respond to whole concepts such as "brown soft patch" or "cluster of corky spots." Nobody writes those detectors. They emerge because they help the network get the answers right.

The turning point is usually dated to 2012, when a deep convolutional network called AlexNet, trained on a large labeled image collection with graphics processors, clearly outperformed the hand-crafted approaches of the time on a major image classification benchmark. In the following years, learned convolutional networks became the default for most vision problems, then deeper designs such as ResNet (2015) made very deep networks trainable, and later the Vision Transformer (2020) showed that attention-based models from language processing could also work on images. We meet each of these ideas in Part III.

Deep learning did not make vision easy. What it changed is where the effort goes: away from designing features by hand and toward collecting and labeling the right data, training, and evaluating honestly. That is why so much of this book is about data. The model code is often short; the judgment around it

is where projects succeed or fail.

> **NOTE**
>
> Classic image processing did not disappear. Resizing, color conversion, normalization, and cropping are still done with classic operations before an image reaches a network, and simple rules remain a valuable baseline. A deep model that cannot beat a page of thresholds is not worth deploying, which is why we build that baseline first.

What Deep Learning Needs

Deep learning is powerful, but it is not free. Before you commit to it, be honest about three requirements: data, labels, and compute.

Data

A network learns only what its training images show it. If every training apple was photographed under the same lamps, the network has no reason to learn what an apple looks like under different ones. The training data must cover the variation the model will face in use: all the apple varieties, sizes, positions on the belt, lighting conditions, and kinds of defect. Small datasets can still work, especially when you start from a network that was pretrained on millions of general images, a technique called transfer learning that we cover in Chapter 13. But "small" still means hundreds or thousands of representative examples, not a dozen.

Labels

Supervised learning needs the right answer for every training image, and the right answer usually comes from a person. Labeling is slow, it is expensive, and it is never perfect. Two experienced graders will disagree on a faint bruise. A tired labeler will occasionally click the wrong button. These mistakes are called **label noise**, and every real dataset has some. Our dataset deliberately contains a small number of wrong labels so that, in Chapter 23, we can practice finding them the same way you would in a real project.

Compute

Training a large network on a large dataset needs a lot of arithmetic, and graphics processors (GPUs) do that arithmetic much faster than ordinary processors. That is why GPUs appear in almost every deep-learning discussion. However, a great deal of useful vision work fits comfortably on an ordinary laptop processor if the images are small and the models are compact. Every example in this book was designed and verified to run on a normal CPU, limited to two threads, in a few minutes per script. Where GPUs genuinely change the picture, such as training at higher resolutions or serving many cameras at once, we say so and show device-agnostic code, clearly labeled as not run in our tests.

When not to use deep learning

Sometimes the honest answer is that you do not need a neural network. If the scene is perfectly controlled and the decision depends on one obvious measurement, such as whether a bottle cap is present, a simple rule may be cheaper, faster, and easier to certify. If you cannot collect representative labeled data, a deep model will learn the wrong thing confidently. If a decision must be explained in exact, auditable terms, a transparent rule may be required. Deep learning earns its place when the visual patterns are too varied or subtle to describe by hand and you can collect enough examples of them. Bruises and scab on apples of three different colors, under varying light, are a good example of exactly that.

> **TIP**
>
> Before starting any vision project, write down the decision the system must make, who acts on it, and what each kind of mistake costs. Those three sentences will guide every later choice, from the task type to the evaluation metric to the confidence threshold.

Meet Greenvale Orchards

Our running example is **Greenvale Orchards**, a fictional family-run apple orchard with its own packhouse. Everything about Greenvale is invented for this book, but the problems it faces are the ordinary problems of a real vision project, and we will meet them in the same order a real team would.

The packing line

Apples ride a conveyor belt under a fixed camera. Two lamps in the camera hood **always light the fruit from the upper left**; remember that detail, because it will matter a great deal in Chapter 9. Greenvale grows three varieties: **Valecrest**, a red apple with darker streaks; **Goldmere**, a yellow apple; and **Fernhill**, a green apple. The training images in this book come from the camera on **Line 2**. Later, Greenvale installs a new camera on **Line 3** with warmer lamps, a tan belt, and a slightly softer lens, and we will see what that innocent-sounding change does to a model trained only on Line 2.

The people

Three people will occasionally speak up in these pages, each representing the kind of question a real team member asks:

- **Ingrid Halloway**, the operations manager, owns the lane rules and asks the business questions: how many apples will end up in manual inspection, and what happens if the camera is wrong?
- **Dario Quenneville**, the machine learning engineer, owns the code, the training runs, and the deployment.
- **Lisbet Ferrandis**, the quality lead, labels images, writes the grading guidelines, and re-inspects doubtful apples by hand at her table.

The lanes and the rule

Greenvale's grading system has to put every apple into one of four lanes. The classifier produces three probabilities, one per class, and a small business rule turns them into a lane:

Predicted class	Lane	What happens there
good	`fresh-pack`	Boxed for the fresh market
bruised	`juice`	Pressed for juice
scab	`processing`	Sauce, slices, and other processed products
any class, below 0.80 confidence	`manual-check`	Lisbet's table for a human decision

The last row is important. If the model's highest class probability is below the confidence threshold of **0.80**, the apple does not go to any automatic lane; it goes to `manual-check`, where a person looks at it. This is a common and sensible pattern: let the machine handle the clear cases and route the doubtful ones to a human.

Not all mistakes are equal. A good apple wrongly sent to juice loses a little money. A bruised or scabby apple wrongly sent to `fresh-pack` ends up in a customer's shopping bag and produces complaints, returns, and lost trust with the buyer. That asymmetry means overall accuracy is not the number Greenvale cares about most. What matters more is how many of the defective apples the system catches, a quantity called **recall** on the bruised and scab classes. We define it carefully in Chapter 10 and keep coming back to it.

(Apple scab is a real fungal disease that leaves dark, corky spots; a bruise is mechanical damage from a knock or drop.)

A five-line preview

Here is the business rule in action. The function `decide_lane` lives in the project file `greenvale/grading.py`, which we set up in Chapter 2. It takes the three class probabilities in the order good, bruised, scab, and returns the winning label, its confidence, and the lane:

preview.py

```
from greenvale.grading import decide_lane

print(decide_lane([0.1, 0.85, 0.05]))
print(decide_lane([0.5, 0.3, 0.2]))
print(decide_lane([0.02, 0.03, 0.95]))
```

Output

```
('bruised', 0.85, 'juice')
('good', 0.5, 'manual-check')
('scab', 0.95, 'processing')
```

In the first line the model is fairly sure the apple is bruised, 0.85 is above the threshold, and it goes to juice. In the second line "good" wins, but only with 0.5, so the apple goes to Lisbet's table rather than straight into a box. In the third, scab wins clearly. Everything else in this book, from pixels to convolutions to deployment, exists to produce those three probabilities as reliably as possible. You do not need to run this code yet; Chapter 2 sets up the project so that you can.

> **WARNING**
>
> A confidence threshold is not a guarantee. A model can be confidently wrong, especially on images that differ from its training data, such as photos from a new camera. In Chapter 22 we measure how well the model's confidence matches reality and see that the threshold must be chosen and rechecked with data, not set once and forgotten.

A Word About Synthetic Data

Real labeled photographs of apple defects, with a license that allows us to ship them inside a book and let you modify them, are hard to come by. So we took a different route: every Greenvale image is drawn by a small, seeded Python program that you will read in Chapter 4. It paints a textured conveyor belt, an apple with a slightly irregular outline, shading from the upper-left lamps, natural pale dots called lenticels, an occasional stem, and, when requested, a bruise or a cluster of scab spots. Then it adds lighting variation and sensor noise.

This approach has real advantages for learning. You can regenerate the identical dataset on your own machine with one command. Every image can be reproduced individually. You get exact ground truth for every pixel, which makes segmentation possible without anyone drawing masks by hand. And we can create controlled problems, such as a second camera with different lamps or a known set of label mistakes, and measure our methods against the true answer.

It also has an important limitation, and we will be direct about it throughout. Real apples are much harder. Real lighting varies across the day, real apples overlap and hide each other, real orchards produce many more kinds of blemish, and a real bruise can be almost invisible on the skin for some time after the knock. The accuracy numbers in this book describe how our models do on our synthetic images. They do not predict how any model would perform in a real packhouse, and they are not agronomic findings. What does transfer is the method: how to build the dataset, train the model, find its weaknesses, and deploy it responsibly.

How This Book Is Organized

The book follows the life of Greenvale's grading system from an empty folder to a monitored service. It is divided into six parts.

1. **Part I, Foundations (Chapters 1 to 5).** We set up the workbench, learn how images become tensors, build the synthetic dataset, and write a rule-based grader with classic image processing. That rule-based grader is the baseline every later model must beat.
2. **Part II, Image Classification (Chapters 6 to 10).** We load images efficiently, understand convolutions from first principles, train our first convolutional network, AppleNet, choose augmentations that respect the physics of the packhouse, and evaluate classifiers with confusion matrices, per-class metrics, and thresholds.
3. **Part III, Modern Architectures and Pretrained Models (Chapters 11 to 15).** Batch normalization and residual connections, a tour of well-known architectures, transfer learning from ImageNet, Vision Transformers, and vision foundation models used for embeddings, similarity search, and zero-shot classification.
4. **Part IV, Detection and Segmentation (Chapters 16 to 20).** Boxes, IoU, anchors, and non-maximum suppression; fine-tuning a Faster R-CNN detector on trays of apples; mean average precision; a small U-Net for defect masks; and measuring defect area.
5. **Part V, Trust and Data Quality (Chapters 21 to 24).** Grad-CAM and other ways of checking what a model looks at, distribution shift and calibration with the Line 3 camera, finding wrong labels, handling class imbalance, active learning, and the habits that make experiments reproducible.
6. **Part VI, Deployment and Production (Chapters 25 to 28).** Exporting to ONNX, fast CPU inference and honest benchmarking, a prediction API for the packing line, and monitoring, ending with a capstone that walks through the whole system.

The appendices collect a cheat sheet, the full source of the dataset generator and other templates, a production checklist, and a glossary.

This is a vision book. We briefly recap the deep-learning ideas you need where they first appear, but we do not re-teach automatic differentiation or language models; a general deep-learning book will fill those gaps.

How to Read the Code and Outputs

A few conventions will make the rest of the book easier to follow.

- **Everything runs.** Every listing in this book was run on our machine, and the output shown beneath it is the real output, trimmed only where a line of `...` marks the cut. Examples use PyTorch 2.x and torchvision (examples tested with PyTorch 2.14 on CPU); recent versions should behave the same way.
- **File labels.** A grey label above a listing names the file it belongs to, such as `greenvale/grading.py`. A label reading *Output* marks what the program printed.
- **Numbers vary a little.** Exact facts such as image counts, tensor shapes, and parameter counts are the same on every machine. Training results can differ slightly across processors, thread counts, and library versions even with fixed random seeds. When we report them we say "on our

machine we got," and you should see something close.

- **CPU first.** Scripts limit PyTorch to two threads so they behave predictably on shared or modest machines. Where a GPU would change things, we explain how, but we never print GPU results we did not measure.
- **One project folder.** All code lives in a project folder called `greenvale-vision/`, organized as a small Python package plus one script per experiment. Chapter 2 creates it.

One teaching habit runs throughout: when there is a common mistake, we show the mistake first and then the fix.

Try It Yourself

1. Pick a vision problem from your own work or daily life, such as reading meter values, spotting damaged parcels, or counting birds at a feeder. Decide whether it is best framed as classification, detection, or segmentation, and write one sentence explaining why the cheaper tasks would or would not be enough.
2. For the same problem, list the two most likely mistakes the system could make and estimate, in words, which one is more costly. How would that change the threshold you would choose for sending a case to a human?
3. Write three hand-crafted rules that might sort Greenvale's apples (for example, based on color, darkness, or spot count). For each rule, describe one situation on a real packing line that would break it.
4. Using the `decide_lane` preview, predict by hand the lane for the probabilities `[0.79, 0.20, 0.01]` and `[0.05, 0.10, 0.85]`. You will check your answers once the project is set up in Chapter 2.
5. List three ways in which real apple photographs would be harder than synthetic ones. Keep the list; in Chapter 22 you will compare it with the specific change that Line 3 introduces.

Key Takeaways

- Computer vision turns pixels into information a business can act on; the main tasks are classification (what), detection (what and where), and segmentation (which pixels).
- Richer tasks cost more to label and compute, so choose the cheapest task that answers the real question.
- Classic vision relies on hand-written features and thresholds; it is fast and transparent but brittle when lighting, cameras, or backgrounds change.
- Deep learning learns the features from labeled data, which moves most of the effort into collecting, labeling, and evaluating data well.
- Deep learning needs representative data, reasonably clean labels, and enough compute, but small images and compact models train fine on an ordinary CPU.
- Greenvale Orchards grades apples as good, bruised, or scab into the lanes fresh-pack, juice, and processing, with manual-check for any prediction below 0.80 confidence.
- A defective apple in fresh-pack is the costly error, so recall on bruised and scab matters more than overall accuracy.
- Our images are synthetic and seeded: perfect for learning and reproducibility, but the results do not predict real-world accuracy.