Chapter 6

Teaching Machines to See: Image Classification with CNNs

CNNs work by applying convolutional filters over image grids to extract features, pooling to reduce dimensions, and fully connected layers for final classification. The chapter builds on prior CNN knowledge and applies it to real-world data, where challenges such as noise, imbalance, and corrupted images occur.

The case study uses the tiny-ImageNet-200 dataset, a smaller version of the ImageNet challenge with 200 classes and 100,000 training images. The dataset contains hierarchical WordNet IDs (wnids) linked to human-readable labels. Figure 1 shows sample images from categories like animals, furniture, and food, while figure 2 depicts the dataset structure, consisting of train, validation, and test folders. The dataset is class-balanced with 500 training images per category (table 1). EDA reveals challenges such as poor-quality images, occlusions, and varying object contexts (e.g., basketballs partly hidden or shown in unusual colors).

Figure 1 Some sample images from the tiny-imagenet-200

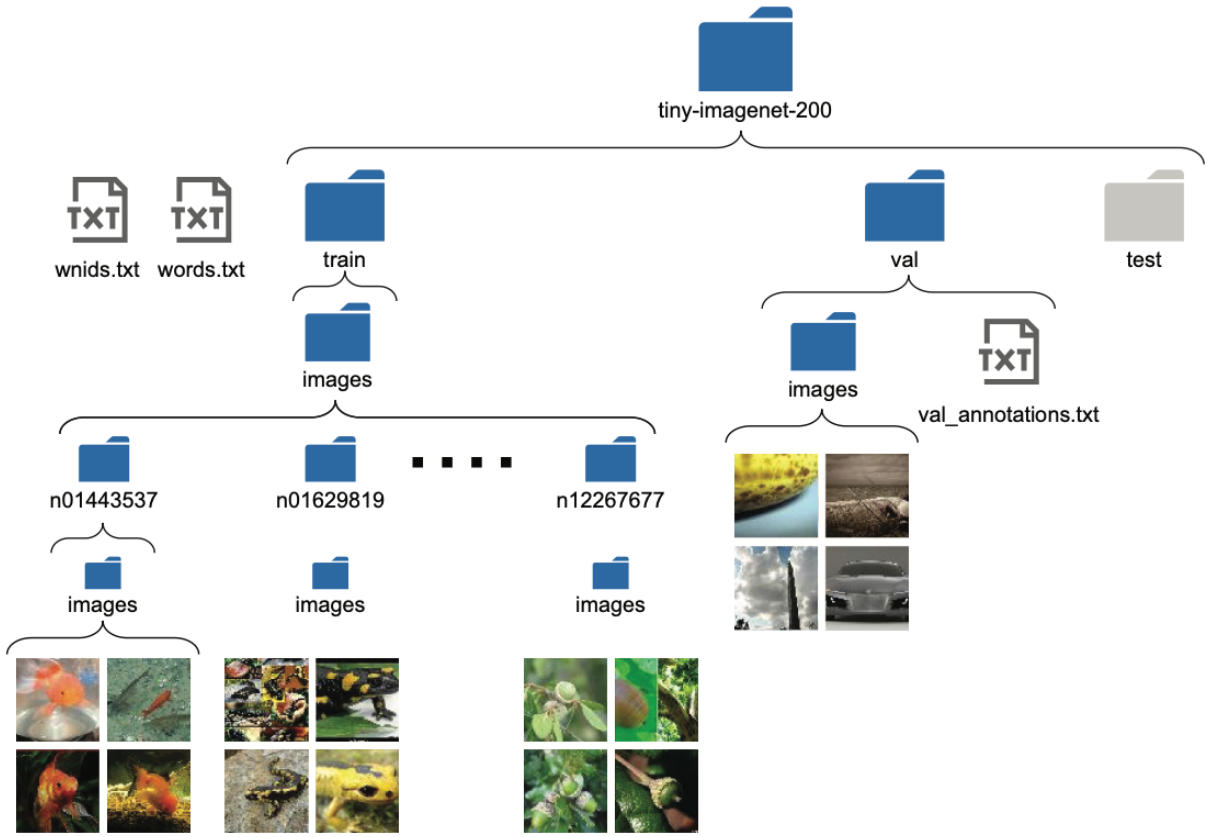

Figure 2 The overall structure of the tiny-imagenet-200 data set. It has three text files (wnids.txt, words.txt, and val/val_annotations.txt) and three folders (train, val, and test)

Table 1 Sample of data where n_train has been calculated

	Wind	Class	N_train
0	N02124075	Egyptian cat	500
1	N04067472	Reel	500
2	N04540053	Volleyball	500
3	N04099969	Rocking chair, rocker	500
4	N07749582	Lemon	500
5	N01651477	Bullfrog, rana catesbeiana	500
6	N02802426	Basketball	500
7	N09246464	Cliff, drop, drop-off	500
8	N07920052	Espresso	500
9	N03970156	Plunger, plumber's helper	500

Reproduced code for download & extract Tiny ImageNet:

```
import os, requests, zipfile


if     not     os.path.exists(os.path.join('data','tiny-imagenet-
200.zip')):
    url = "http://cs231n.stanford.edu/tiny-imagenet-200.zip"
    r = requests.get(url)


    if not os.path.exists('data'):
        os.mkdir('data')


    with open(os.path.join('data','tiny-imagenet-200.zip'), 'wb')
as f:
        f.write(r.content)


    with      zipfile.ZipFile(os.path.join('data','tiny-imagenet-
200.zip'), 'r') as zip_ref:
        zip_ref.extractall('data')
else:
    print("The file already exists.")
```

EDA is the first technical step in building an image classification model after defining the business problem. Its purpose is to deeply understand the dataset, spot potential issues, and prepare the data in a clean and structured form before model training. For the shopping-assistant case study, the dataset used is tiny-ImageNet-200, which contains 200 classes with 500 training images each (100,000 total). After EDA, the chapter introduces data pipelines using Keras's ImageDataGenerator, which handles batch loading, preprocessing (e.g., centering, normalization), and dataset splitting. Different generators are built for training, validation, and testing. Flow methods (flow_from_directory, flow_from_dataframe) streamline image ingestion from folders or annotation files. The pipelines are then adapted to meet Inception Net's multi-output requirement by replicating labels for auxiliary outputs.

The chapter's centerpiece is the Inception Net v1 architecture (GoogLeNet), a CNN designed to improve accuracy while managing computational cost. Instead of a single convolution per

layer, Inception blocks combine parallel convolutions of different kernel sizes (1×1, 3×3, 5×5) and pooling, then concatenate outputs. This design improves representational power while reducing parameters. For example, an Inception block with varied filters needs ~576 parameters, compared to 1,600 for a standard 5×5 convolution. The efficiency stems from sparsity and the use of 1×1 convolutions for dimensionality reduction (figure 3).

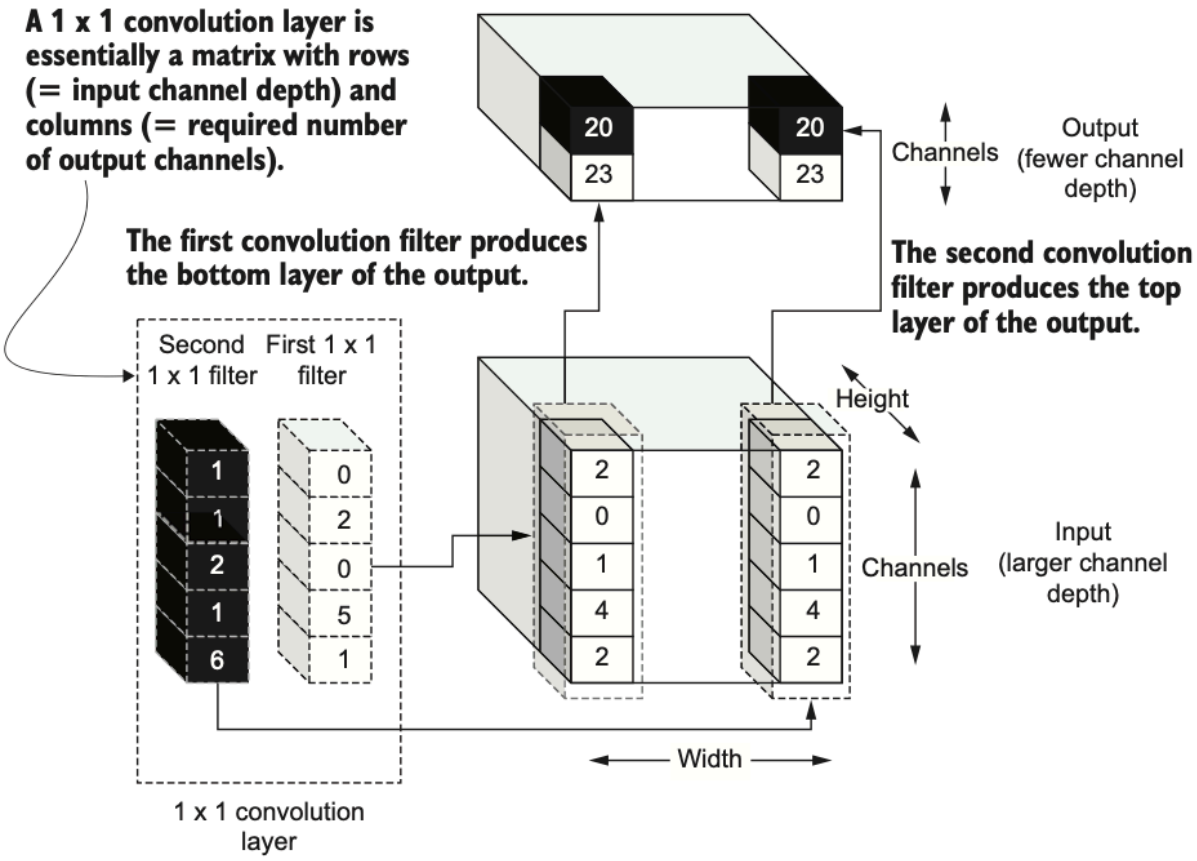

Figure 3 The computations of a 1x1 convolution and how it enables reduction of a channel dimension of an input

Inception Net v1 also includes auxiliary classifiers—two extra softmax outputs inserted mid-network to stabilize training and mitigate vanishing gradients. These consist of average pooling, 1×1 convolutions, dense layers, and softmax outputs. Together with the final prediction layer, they make training of deeper CNNs feasible. The full Inception model, adapted for tiny-ImageNet (56×56 inputs, 200 classes), is implemented in TensorFlow/Keras using the functional API.