NVIDIA. | DEEP LEARNING INSTITUTE

LEARNING DEEP LEARNING

Theory and Practice of Neural Networks, Computer Vision, Natural Language Processing, and Transformers Using TensorFlow

MAGNUS EKMAN

Learning Deep Learning

Experiment: Deeper and Regularized Models for House Price Prediction

We now present the results of some experiments where regularization techniques are applied to the model. As previously stated, we saw that the three-layer model was significantly better than the linear model but suffered from overfitting. Those results are shown in the first two rows in Table 6-2, where the columns show the network topology (each number represents the number of neurons in a layer), which regularization technique is used, as well as the training and test errors.

The third row in the table (Configuration 3) shows what happens when we add L2 regularization to the model. We use a lambda of 0.1, and we can see that the training error increases, but unfortunately, the test error increases slightly as well.

The next row (Configuration 4) shows what happens if we use dropout (factor 0.2) instead of L2 regularization. This is more effective and almost closes the gap between the training and test errors. This indicates that overfitting is no longer a big problem, and it makes sense to try a more complex model.

This is shown in the next row (Configuration 5), where we add another layer and increase the number of neurons to 128 in the first two layers. This improves the test error, but we see that the training error is reduced even more, so we now have problems with overfitting again.

Table 6-2 Experiments with Deeper Models and Regularization

CONFIGURATION	TOPOLOGY	REGULARIZATION	TRAINING ERROR	TEST ERROR
Conf1	1	None	10.15	10.24
Conf2	64/64/1	None	0.647	2.54
Conf3	64/64/1	L2=0.1	1.50	2.61
Conf4	64/64/1	Dropout=0.2	2.30	2.56
Conf5	128/128/64/1	Dropout=0.2	2.04	2.36
Conf6	128/128/64/1	Dropout=0.3	2.38	2.31

In the final row of the table (Configuration 6), we increase the dropout factor to 0.3, which both increases the training error and decreases the test error, and we have arrived at a model that generalizes well.

Concluding Remarks on Output Units and Regression Problems

In this chapter, we described the three most common types of output units and their associated loss functions. Whereas the types of hidden units typically are chosen in the process of tuning hyperparameters, the type of output unit is tightly coupled to the problem type.

When training DL models, it is common to run into overfitting. This can be addressed by regularizing the model. In this chapter, we described a number of regularization techniques and applied them to our programming example.

The programming example showed that it is often necessary to tweak parameters iteratively to get to a model that performs well on the test set. One thing to note is that our best configuration has more than 26,000 parameters. This can be compared to the linear regression case, which has one weight for each input feature plus a bias weight—14 in total in our example. From the perspective of predicting well, it clearly pays to have all of these parameters that the model learns by itself. However, it is much harder to understand a model with 26,000 parameters than a model with 14 parameters, which illustrates a common problem with DL. We end up with a model that works well, but we do not know how it works.

Overall, our impression is that, as the current DL boom started, the field transformed from being theoretical to being more empirical. In other words, focus has shifted from how something works to how well it works. Before the field could demonstrate impressive results, perhaps it had to produce elaborate mathematical analysis to justify its existence, whereas the more recent results are so impressive that people are happy to skip the math?

Chapter 7

Convolutional Neural Networks Applied to Image Classification

Training of deep models with backpropagation has been demonstrated in various forms since at least 1990 (Hinton, Osindero, and Teh, 2006; Hinton and Salakhutdinov, 2006; LeCun et al., 1990; LeCun, Bottou, Bengio, et al., 1998). Still, a pivotal point for deep learning (DL) was in 2012 when AlexNet was published (Krizhevsky, Sutskever, and Hinton, 2012). It scored significantly better than any other contestant in the ImageNet classification challenge (Russakovsky et al., 2015) and greatly contributed to popularizing DL. AlexNet is an eight-layer network and uses convolutional layers, which were introduced by Fukushima (1980) and later used in LeNet (LeCun et al., 1990). Convolutional layers, and the resulting convolutional neural networks (CNNs), are important building blocks in DL. This chapter describes how they work. We start by introducing the overall AlexNet architecture to highlight a number of concepts that we then explain in more detail.

AlexNet is a **convolutional neural network** for image classification. It scored well on the ImageNet challenge in 2012 and has been attributed as a key reason for the DL boom that evolved over the next few years.

The topology of the AlexNet CNN is shown in Figure 7-1. It consists of five convolutional layers (drawn as 3D blocks) followed by three fully connected layers (drawn as 2D rectangles). One somewhat confusing property is that the layers are split up horizontally, so each layer is represented as two blocks or rectangles. The reason for this is that, at the time, there was no graphics processing unit (GPU) that had enough memory to be able to run the entire network. The solution was to split up the network and map it to two GPUs. Although important at the time, we ignore that detail in our discussion and focus on other properties of the network.

We make the following additional observations from the figure:

- The input image is 224×224 pixels, where each pixel has a depth of 3 (represented by the *3* in the lower left corner of the figure), which represents the three color channels red, green, and blue (RGB).
- The convolutional layers have a 3D structure as opposed to the fully connected layers, which have a single dimension (vector).
- There are seemingly arbitrary mappings from sub-blocks of varying sizes in one layer to the next (marked 11×11, 5×5, 3×3), and there seems to be no method to the madness when it comes to how the dimensions of one layer relate to the dimensions of a subsequent layer.

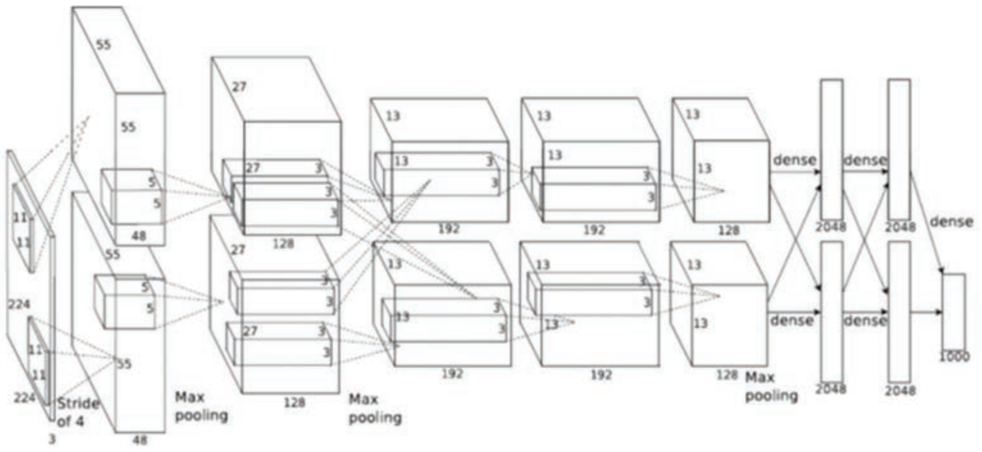

Figure 7-1 Topology of the AlexNet convolutional network. (Source: Krizhevsky, A., Sutskever, I., and Hinton, G., "ImageNet Classification with Deep Convolutional Neural Networks," Advances in Neural Information Processing Systems 25 [NIPS 2012], 2012.)

- There is something called *stride*.
- There is something called *max pooling*.
- The output layer consists of 1,000 neurons (it says "1000" in the lower right corner of the figure).

In this chapter, we describe all of the above and additionally describe terminology such as *kernel size* (refers to the 11×11, 5×5, 3×3 items in the figure) and *padding*, which are important concepts to know when designing and training a CNN. Before going into these details, we introduce the input dataset that we use in this chapter.

The CIFAR-10 Dataset

The CIFAR-10 dataset consists of 60,000 training images and 10,000 test images, each belonging to one of the ten categories *airplane, automobile, bird, cat, deer, dog, frog, horse, ship,* and *truck,* as previously shown in Figure P-1 in the preface. Each image is 32×32 pixels, so altogether it might seem like the dataset is similar to the MNIST handwritten digit dataset studied in earlier chapters. However, the CIFAR-10 dataset is more challenging in that it consists of color images of everyday objects that are much more diverse than handwritten digits. Figure 7-2

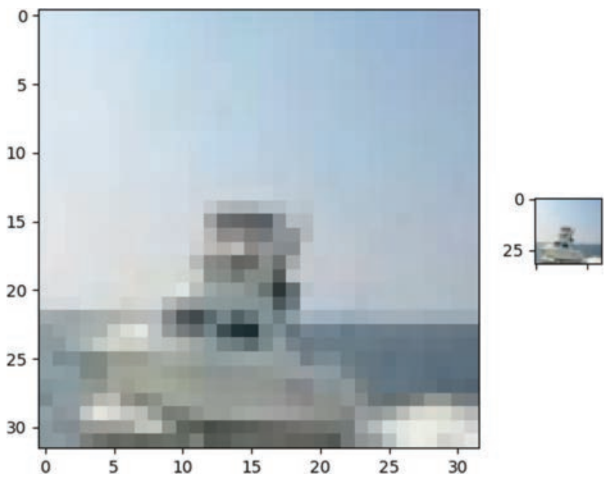

Figure 7-2 Image 100, belonging to the ship category in the CIFAR-10 dataset. (Source: Krizhevsky, A., *Learning Multiple Layers of Features from Tiny Images*, University of Toronto, 2009.)

shows image number 100 (starting counting from 0) in the CIFAR-10 dataset. The figure shows a magnified version in which each of the 32×32 pixels can be clearly seen, and next to it, a more realistically sized version given the low resolution of the image.

When working with a new dataset, it always makes sense to explore it a little bit. The CIFAR-10 dataset is included in Keras. Code Snippet 7-1 shows how to access it and display the ship image shown in Figure 7-2.

Code Snippet 7-1 Python Code to Access the CIFAR-10 Dataset and Display One of the Images

```python
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
import logging
tf.get_logger().setLevel(logging.ERROR)

cifar_dataset = keras.datasets.cifar10
(train_images, train_labels), (test_images,
    test_labels) = cifar_dataset.load_data()

print('Category: ', train_labels[100])
plt.figure(figsize=(1, 1))
plt.imshow(train_images[100])
plt.show()
```

In addition to displaying the image, the print statement should result in the following output, where 8 refers to the *ship* category:

```
Category:  [8]
```

Apparently, the `train_labels` variable is a 2D array (the 8 is enclosed within brackets, which indicates that `train_labels[100]` is still an array instead of a scalar value). We can explore this further, this time by just typing the following commands in a Python interpreter:

```
>>> import tensorflow as tf
>>> from tensorflow import keras
>>> import numpy as np
```