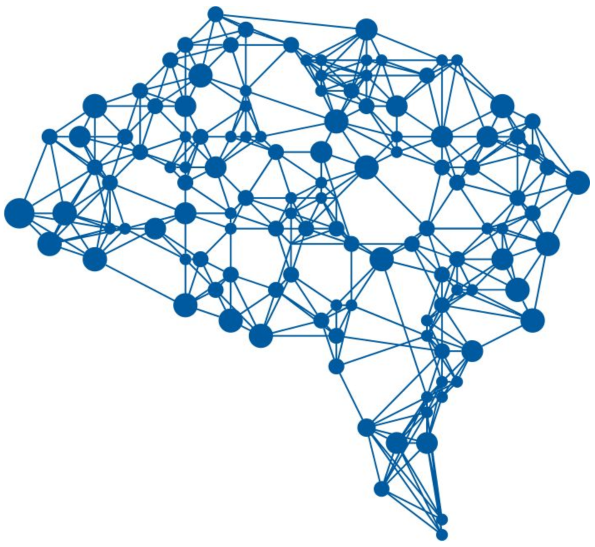


Deep Learning

Mingming Li

First Created: September 29, 2020
Last Modified: November 30, 2022

Contents

List of Figures

List of Tables

Part I

Basics

Chapter 1

Introduction

At the start of the artificial intelligence, a set of rules was written to do some computationally intensive work. At this time, it is called **knowledge base**.

Some tasks to difficult for human to write the set of rule to consider all the situations. It involved to learn from data. Human first extract features from raw data and computer extract patterns from features. At this time, it is called **machine learning**.

For some tasks, for example image classification, face recognition, it is difficult to extract features. We let computer learn extracting features and patterns from raw data. For example, in image classification, it first extract line features. From the line features, it extract shape features. From the shape features, it extract more abstract features, like object features. In patterns extracting process, it reduce the dimension gradually to the classes. Considering the whole processing units, it is very deep. At this time, it is called **deep learning**.

Chapter 2

Machine Learning Basics

2.1 Learning Algorithms

A machine learning algorithm is an algorithm that is able to learn from data. But what do we mean by learning? Mitchell provides a succinct definition: “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E” Now, how does the learn happen? The model or algorithm learns by adjusting the parameters contained in it.

2.2 Capacity, Overfitting, Underfitting and Regularization

We train model on training data but use test data (not used to train the model) to test out model. The ability to perform on test data is called **generalization**. We can use model on test data because we assume that the train data and the test has the same probability distribution (i.e. they have relationship).

The error on training data is called **training error**. The error on test data is called **test error**. Underfitting occurs when the model is not able to obtain a sufficient low error value on the training set. Overfitting occurs when the gap between the training error and test error is too large.

We can control whether a model is more likely to overfit or underfit by altering its **capacity**. The capacity is the pattern space (family of functions) we can learn from.

Regularization is any modification we make to a learnining algorithm that is intended to reduce its generalization error.

Machine learning algorithm will generally perform best when their capacity is appropriate for the true complexity of the task and the amount of training data.

2.2.1 The No Free Lunch Theorem

For any algorithms a_1 and a_2, at iteration step m

$$\sum P(d_m^y|f, m, a_1) = \sum P(d_m^y|f, m, a_2) \tag{2.1}$$

where d_m^y denotes the ordered set of size m of the cost values y associated to input values $x \in X$, $f : X \longrightarrow Y$ is the function being optimized and $P(d_m^y | f, m, a)$ is the conditional probability of obtaining a given sequence of cost values from algorithm a run m times on function f.

The no free lunch theorem implies that we must design our machine learning algorithms to perform well on a specific task but not a universal task.

2.3 Hyperparameters and validation sets

hyperparameters are parameters used to control the algorithm's behavior but can or should not be learned by the learning algorithm.

In practice, we usually split training data into two disjoint subsets: training set and validation set (generally, 8:2n). The validation set is used to adjust the hyperparameters.

2.4 Building a machine learning algorithm

Nearly all deep learning algorithms can be described as particular instances of a fairly simple recipe:

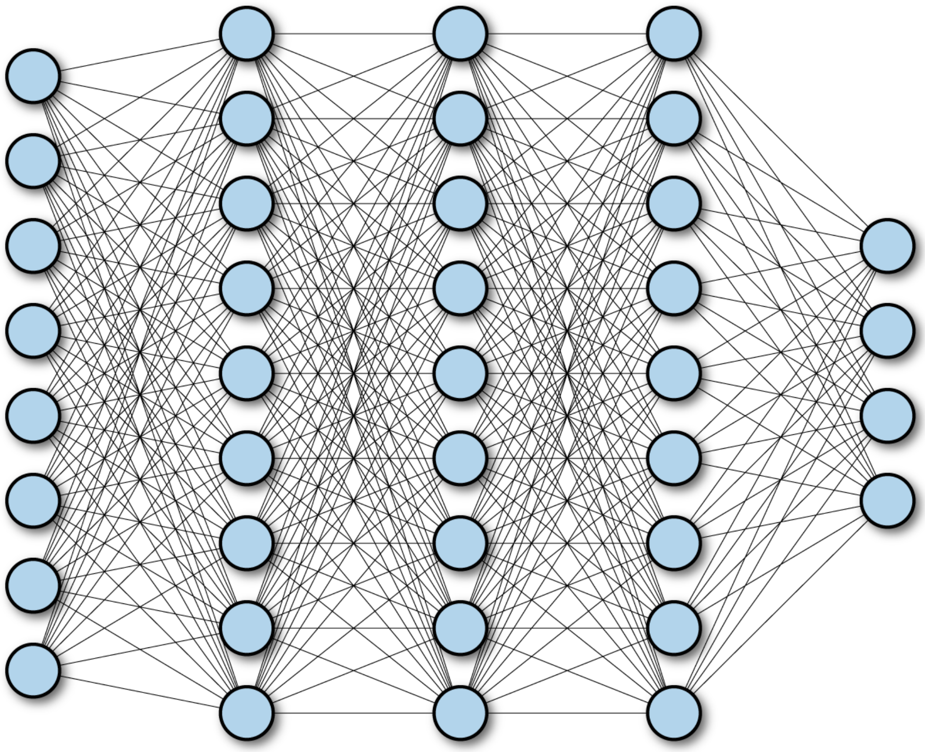
- ♥ a specification of a dataset
- ♥ a cost function
- ♥ an optimization procedure
- ♥ a model

Chapter 3

Full Connected Networks

A fully connected neural network consists of a series of fully connected layers that connect every neuron in one layer to every neuron in the other layer. Full connected neural network layers use matrix multiplication by a matrix of parameters with a separate parameter describing the interaction between each input unit and each output unit. This means that every output unit interacts with every input unit.

Figure 3.1 show the full connects neural network.



Figures 3.1: Full connected neural network

Part II

Computer Vision

Chapter 4

CNN

CNN stands for convolutional neural network. Convolutional networks are neural networks that have convolutional layers. A typical convolutional layer consists of three stages:

1 convolution stage: affine transform
2 detector stage: nonlinearty
3 pooling stage

4.1 Convolution

$$s(t) = \int x(a)w(t-a)da. \tag{4.1}$$

This operation is called **convolution**. The convolution operation is typically denoted with an asterisk:

$$s(t) = (x * w)(t). \tag{4.2}$$

In convolutional network terminology, the first argument (in this example, the function x) to the convolution is often referred to as the **input**, and the second argument (int this example, the function w) as the **kernel**. The output is sometimes referred to as the **feature map**.

If we assume that x and w are defined only on integer t, we can define the discrete convolution:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a). \tag{4.3}$$

We often use convolutions over more than one axis at a time. For example, if we use a two-dimensinal image I as our input, we probably also want to use a two-dimensional kernel K:

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n)K(i-m, j-n). \tag{4.4}$$

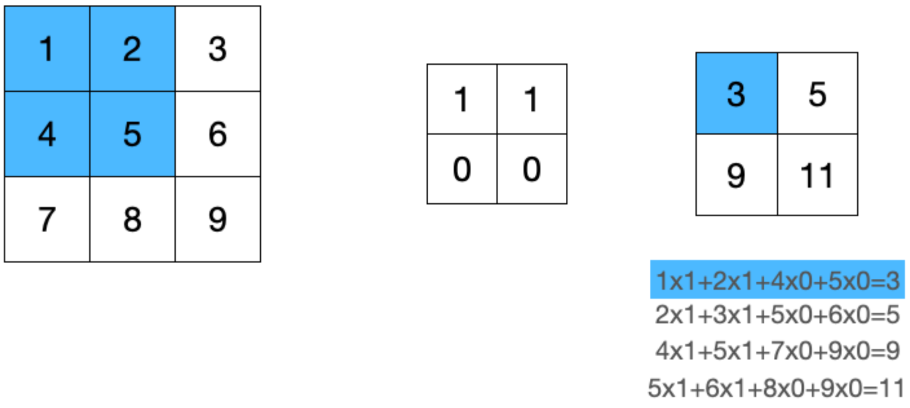
The following formula can be used to calculate the output dimension.

$$h_o = \frac{h_i - h_k}{h_s} + 1 \tag{4.5}$$

$$w_o = \frac{w_i - w_k}{w_s} + 1 \tag{4.6}$$

where h_o is the output height, h_i is the input height, h_k is the kernel height, h_s is the stride height, w_o is the output width, w_i is the input width, w_k is the kernel width, w_s is the stride width.

The convolution operation is shown in Figure 4.1.



Figures 4.1: Convoluation operation

4.2 Properties

CNN leverages three important ideas:

- sparse interaction.
- parameter sharing.
- equivariant representations.

4.2.1 Sparse interaction

This is accomplished by making the kernel smaller than the input.

4.2.2 Parameter sharing

In convolutional layers, the same parameter defined in one kernel are used at every position of the input.

4.2.3 Equivariant representations

In the case of convolution, the particular form of a parameter sharing causes the layer to have a property called **equivariance** to translation. To say a function is equivariant means that if the input changes, the output changes in the same way.

4.3 Pooling

A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs. For example, the max pooling oeration reports the maximum output within a rectangular neighborhood. Pooling helps to make the representation approximately invariant to small translations of the input. Invariant to translation means that if we translate the input by a small amount, the values of most of the pooled outputs do not change.

The following formula can be used to calculate the output dimension.

$$h_o = \frac{h_i - h_k}{h_s} + 1 \tag{4.7}$$

$$w_o = \frac{w_i - w_k}{w_s} + 1 \tag{4.8}$$

where h_o is the output height, h_i is the input height, h_k is the pooling height, h_s is the stride height, w_o is the output width, w_i is the input width, w_k is the pooling width, w_s is the stride width.

Part III

Natural Language Processing

Chapter 5

Text Preprocessing

To convert text to a data format that is easier for computer to train a model, we need text preprocessing. Here are the common preprocessing steps for text:

1. Load text as strings into memory.
2. Split strings into tokens (e.g., words or characters).
3. Build a table of vocabulary to map the split tokens to numerical indices.
4. Convert text into sequences of numerical indices.

A **token** is the basic unit in text, for example, word or character. The string type of the token is inconvenient to be used by models. We build a dictionary called vocabulary to map string tokens into numerical indices starting from 0. To do so, we first count the unique tokens in all the documents from the training set, namely a **corpus**, and then assign a numerical index to each unique token according to its frequency. Rarely appeared tokens are often removed to reduce the complexity. Any token that does not exist in the corpus or has been removed is mapped into a special unknown token "<unk>". We can also add a list of reserved tokens, such as “<pad>” for padding, “<bos>” to present the beginning for a sequence, and “<eos>” for the end of a sequence.

Chapter 6

Language Model

The goal of a language model is to estimate the joint probability of the sequence

$$P(x_1, \ldots, x_T) \tag{6.1}$$

Where T is a constant.

Generally, the probability of $(x_1, \ldots, x_T)$ is:

$$P(x_1, \ldots, x_T) = \prod_{t=1}^{T} P(x_t | x_{t-1}, \ldots, x_1) \tag{6.2}$$

6.1 Markov Model

In probability theory, a Markov model is a stochastic model used to model pseudo-randomly changing systems. It is assumed that future states depend only on the current state, not on the events that occurred before it (that is, it assumes the Markov property). Generally, this assumption enables reasoning and computation with the model that would otherwise be intractable.

Where T is a constant.

The simplest Markov model is the Markov chain.

$$P(x_t | x_1, \ldots, x_{t-1}) = P(x_t | x_{t-1}) \tag{6.3}$$

In this case, we have a **first-order Markov model** and $P(x)$ is given by:

$$P(x_1, \ldots, x_T) = \prod_{t=1}^{T} P(x_t | x_{t-1}). \tag{6.4}$$

6.2 n-grams

$$P(x_1, \ldots, x_T) = \prod_{t-1}^{T} P(x_t) \tag{6.5}$$

$$P(x_1, \ldots, x_T) = \prod_{t-1}^{T} P(x_t|x_{t-1}) \tag{6.6}$$

$$P(x_1, \ldots, x_T) = \prod_{t-1}^{T} P(x_t|x_{t-1}, x_{t-2}) \tag{6.7}$$

Formula 6.5 6.6 and 6.7 refers to **unigram**, **bigram** and **trigram**.

Chapter 7

RNN

RNN stands for recurrent neural network. RNN is neural network that has recurrent layers. While CNNs can efficiently process spatial information, RNNs are designed to better handle sequential information. RNNs introduce state variables to store past information, together with the current inputs, to determine the current outputs.

7.1 Recurrent

$$s^{(t)} = f(s^{(t-1)}; \theta) \tag{7.1}$$

Where $s^{(t)}$ is the state or output at time t, f is the function, θ is the parameter. Equation 7.1 is recurrent because s at time t refer to itself at time $t-1$.

$$s^{(t)} = f(s^{(t-1)}, x^{(t)}; \theta) \tag{7.2}$$

This equation is similar to equation 7.1 but with input $x^{(t)}$.

In natural language processing (NLP), we usually need to compute

$$P(x_t | x_{t-1}, \ldots, x_1) \tag{7.3}$$

With n-gram model (Section 6.2), the number of model parameters increase exponentially as n increase. We need to store $|V|^n$ numbers for a vocabulary set V. In this case, we usually use a latent variable model:

$$P(x_t | x_{t-1}, \ldots, x_1) \approx P(x_t | h_{t-1}) \tag{7.4}$$

and

$$h_t = f(x_t, h_{t-1}) \tag{7.5}$$

7.2 Properties

RNN leverages three important ideas:

- ♥ sparse interaction.
- ♥ parameter sharing.
- ♥

7.2.1 Sparse interaction

Comparing to traditional neural network, it has sparse interaction. For example, in Equation 7.2, $s^{(t)}$ is determined by $s^{(t-1)}$ and $x^{(t)}$. It has no direct interaction with $x^{(1)}, x^{(2)}, \ldots x^{(t-1)}$ because they are contained in $s^{(t-1)}$.

7.2.2 Parameter sharing

In recurrent layers, the same parameters defined in function f are used at every position of the input.

Part IV

Computer Vision Practice

Chapter 8

Classification

Image classification comprises two major part: CNN network part and full connected network part.

CNN network is used to extract feature maps from the images. Feature maps contains the information used to classifier the image. The FC network output n-class dimension vector, each dimension for a class probability.

8.1 LeNet with Keras

The LeNet architecture is a seminal work in the deep learning community, first introduced by LeCun et al. in their 1998 paper, Gradient-Based Learning Applied to Document Recognition [4].

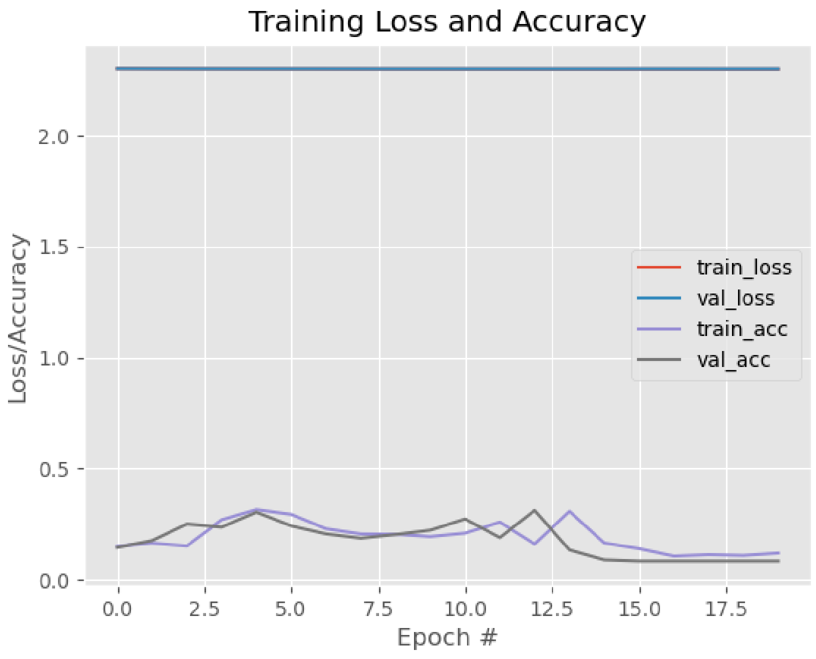
The code is on Github.

8.1.1 Error and Anylysis

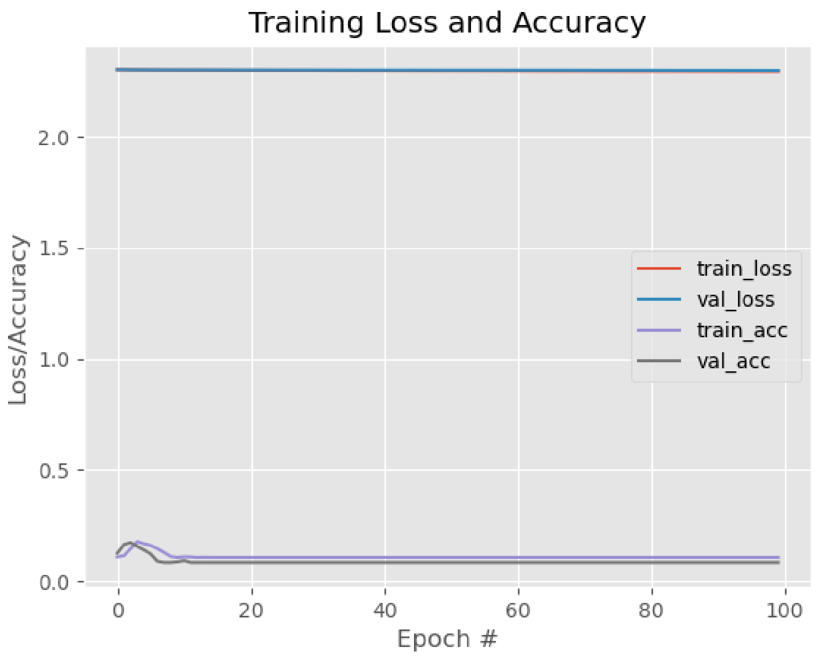
At first, the division value used is 255.0 (train.py line 35). Normally, this should make sense, becuase the value of image points lay in [0,255]. The output is as shown in Figure 8.1 (epochs=20) and 8.2 (epochs=100).

After diving into the dataset, I found that the maximum value is 16. After changing the division to 16, the result is shown in Figure 8.3.

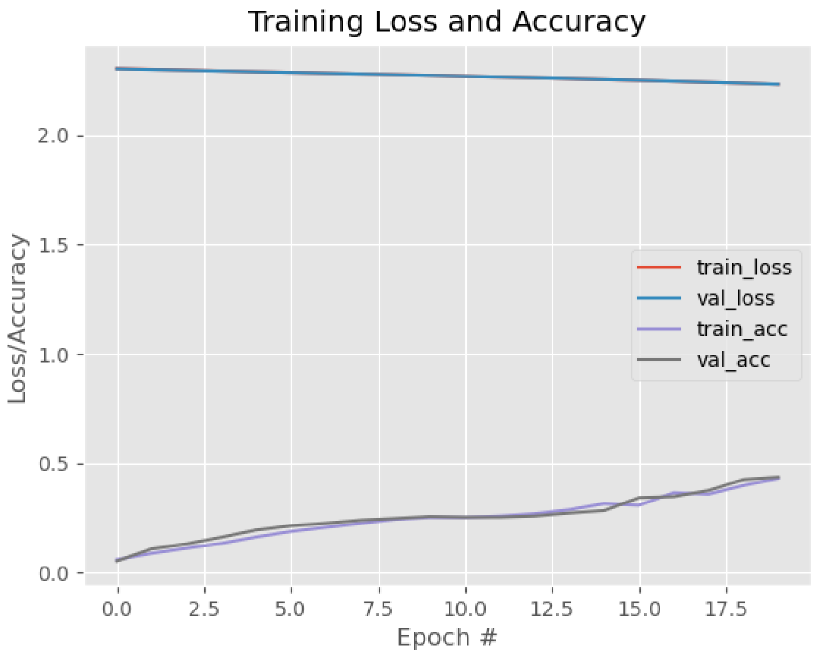
From the Figue 8.3 we can see that the epochs is too small. After chaning the epochs to 100, the result is shown inf Figure 8.4.



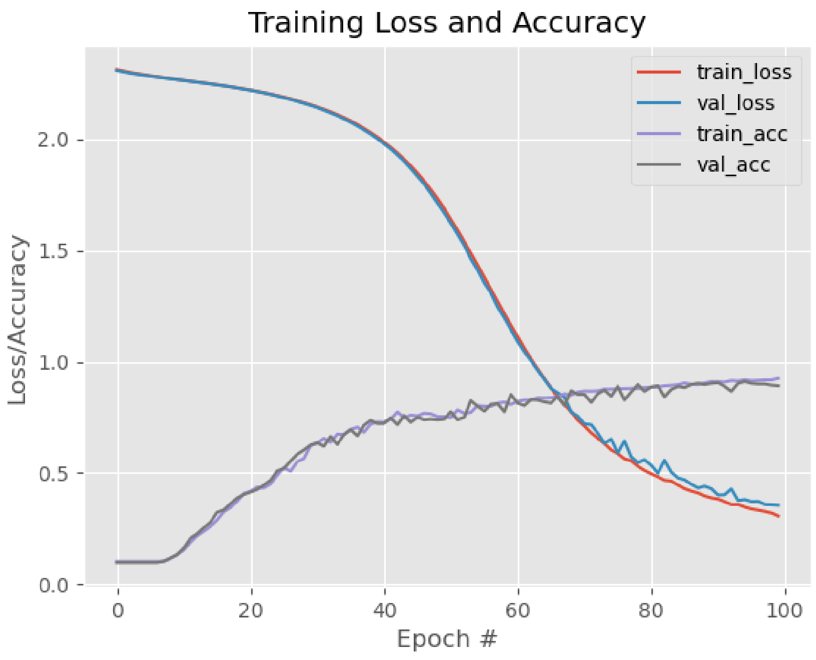
Figures 8.1: Divide 255 and epochs=20



Figures 8.2: Divide 255 and epochs=100



Figures 8.3: Divide 16 and epochs=20



Figures 8.4: Divide 16 and epochs=100

8.2 LeNet with PyTorch

The code link on Github.

Before, there is no tensorboard in PyTorch. You did the training process visualization. With the tensorboard, it becomes more easier for the visualization in training process.

Chapter 9

Object Detection

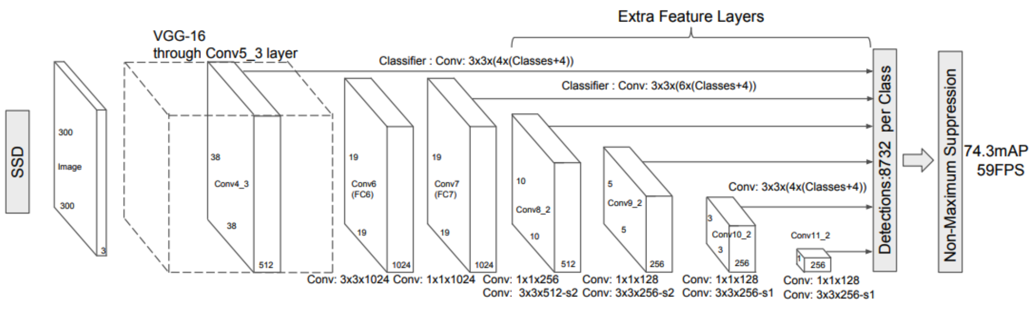
Usually, there are often multiple objects in the image of interest. We not only want to know their categories, but also their specific positions in the image. In computer vision, we refer to such tasks as object detection (or object recognition).

9.1 Single-Shot Detector

The object dection model used here is the SSD[1].

The code is on Github.

Figure **??** show the structure.



Figures 9.1: SSD

There are two parts: backbone and SSD head. The backbone is the EGG as the feature extractor. The SSD head is a set of convolution layers. The SSD head extracts features on different size. Then it do regression and classification to achieve anchor box offset and object class.

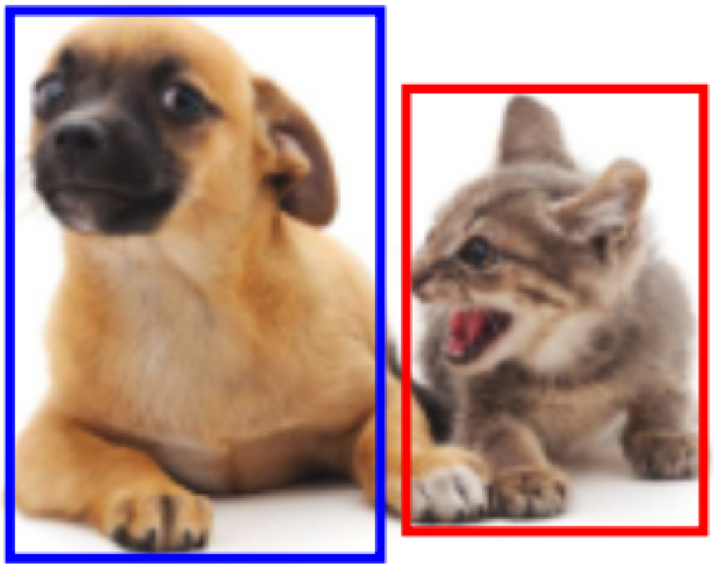
9.1.1 Bounding box

In object detection, we usually use a **bounding box** to describe the spatial location of an object. The bounding box is rectangular, which is determined by the x and y coordinates of the upper-left corner of

[1]https://arxiv.org/pdf/1512.02325.pdf

the rectangle and the such coordinates of the lower-right corner. Another commonly used bounding box representation is the (x, y)-axis coordinates of the bounding box center, and the width and height of the box.

For example in Figure 9.2.



Figures 9.2: Bounding box

9.1.2 Anchor boxes

Object detection algorithms usually sample a large number of regions in the input image, determine whether these regions contain objects of interest, and adjust the boundaries of the regions so as to predict the ground-truth bounding boxes of the objects more accurately.

Different models may adopt different region sampling schemes. One method is: it generates multiple bounding boxes with varying scales and aspect ratios centered on each pixel. These bounding boxes are called **anchor boxes**.

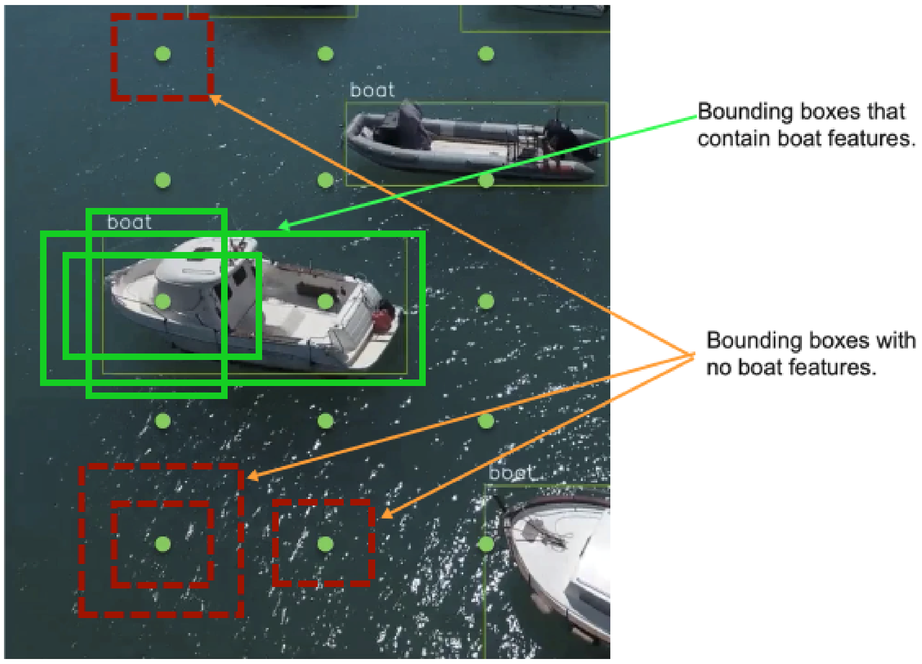
Suppose that the input image has a height of h and width of w. We generate anchor boxes with different shapes centered on each pixel of the image. Let the scale be $s \in (0, 1]$ and the aspect ratio (ratio of width to height) is $r > 0$. Left out the scale s and the anchor box area does not change. The new width and height of the anchor box are $w\sqrt{r}$ and $h/\sqrt{r}$ respectively. Counting the scale s, those are $ws\sqrt{r}$ and $h/\sqrt{r}$ respectively. Note that when the center position is given, an anchor box with known width and height is determined.

To generate multiple anchor boxes with different shapes, let us set a series of scales $s_1, \ldots, s_n$ and a series of aspect ratios $r_1, \ldots, r_m$. When using all the combinations of these scales and aspect ratios with each pixel as the center, the input image will have a total of $whnm$ anchor boxes. Although these anchor boxes may cover all the ground-truth bounding boxes, the computational complexity is easily too high. In practice, we can only consider those combinations containing s_1 and $r1$:

$$(s_1, r_1), (s_1, r_2), \ldots, (s_1, r_m), (s_2, r_1), (s_3, r_1), \ldots, (s_n, r_1) \tag{9.1}$$

That is to say, the number of anchor boxes centered on the same pixel is $n + m - 1$. For the entire input image, we will generate a total of $wh(n+m-1)$ anchor boxes.

For example in Figure


boat
boat
boat
Bounding boxes that contain boat features.
Bounding boxes with no boat features.

Figures 9.3: Anchor boxes

9.1.3 IntersectionoverUnion(IoU)

We use IoU to measure the similarity between the anchor boxes and the ground-truth bounding box.

9.1.4 Labeling Anchor Boxes in Training Data

In a training dataset, we consider each anchor box as a training example. In order to train an object detection model, we need class and offset labels for each anchor box.

To label any generated anchor box, we refer to the labeled location and class of its assigned ground-truth bounding box that is closest to the anchor box.

Given an image,suppose that the anchor boxes are $A_1, A_2, \ldots A_{n_a}$ and the ground-truth bounding boxes are $B_1, B_2, \ldots B_{n_b}$, where $n_a \geq n_b$. Let us define a matrix $X \in R_{n_a \times n_b}$, whose element x_{ij} in the i^{th} row and j^{th} column is the IoU of the anchor box A_i and the ground-truth bounding box B_j. The algorithm consists of the following steps:

1 Find the largest element in matrix X and denote its row and column indices as i_1 and j_1, respectively. Then the ground-truth bounding box B_{j_1} is assigned to the anchor box A_{i_1}. After the first assignment, discard all the elements in the i_1^{th} row and the j_1^{th} column in matrix X.

2 Repeat process 1 until all elements in n_b columns in matrix X are discarded.

3 Traverse through the remaining $n_a - n_b$ anchor boxes. For example, given any anchor box A_i, find the ground-truth bounding box B_j with the largest IoU with A_i throughout the i^{th} row of matrix X, and assign B_j to A_i only if this IoU is greater than a predefined threshold.

9.1.5 Labeling Classes and Offsets

Suppose that an anchor box A is assigned a ground-truth bounding box B. On one hand, the class of the anchor box A will be labeled as that of B. On the other hand, the offset of the anchor box A will be labeled according to the relative position between the central coordinates of B and A together with the relative size between these two boxes. Here's a command transformation. Given the central coordinates of A and B as (x_a, y_a) and (x_b, y_b), their widths as w_a and w_b, and their heights as h_aa nd h_b, respectively. We label the offset of A as:

$$\left(\frac{\frac{x_b-x_a}{w_a} - \mu_x}{\sigma_x}, \frac{\frac{y_b-y_a}{h_a} - \mu_y}{\sigma_y}, \frac{\log \frac{w_b}{w_a} - \mu_w}{\sigma_w}, \frac{\log \frac{h_b}{h_a} - \mu_h}{\sigma_h}\right) \tag{9.2}$$

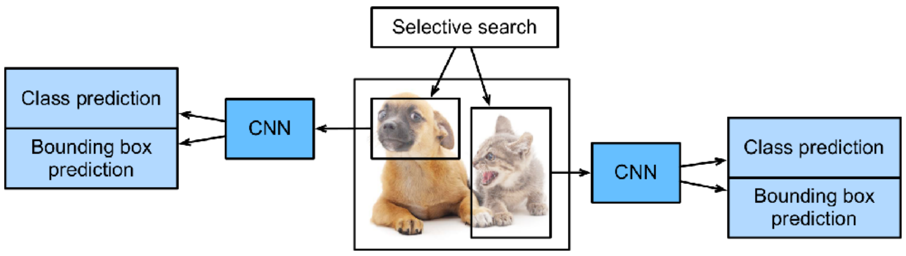
where default values of the constants are $\mu_x = \mu_y = \mu_w = \mu_h = 0, \sigma_x = \sigma_y = 0.1$ and $\mu_w = \mu_h = 0.2$.

9.2 R-CNN models

9.2.1 Region-based CNN (R-CNN)

The R-CNN consists of the following four steps:

1. Perform **selective search** to extract multiple high-quality **region proposals** on the input image. These proposed regions are usually selected at multiple scales with different shapes and sizes. Each region proposal will be labeled with a class and a ground-truth bounding box.
2. Choose a pretrained CNN and truncate it before the output layer. Resize each region proposal to the input size required by the network, and output the extracted features for the region proposal through forward propagation.
3. Take the extracted features and labeled class of each region proposal as an example. Train multiple **support vector machines** to classify objects, where each support vector machine individually determines whether the example contains a specific class.
4. Take the extracted features and labeled bounding box of each region proposal as an example. Train a **linear regression** model to predict the ground-truth bounding box.

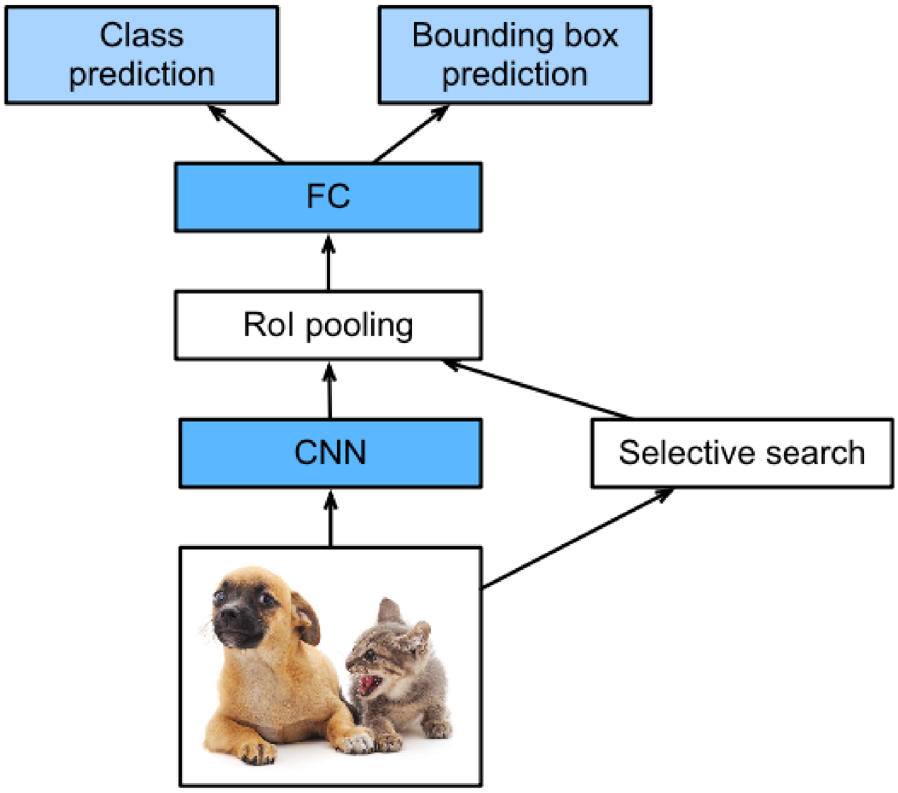


Figures 9.4: R-CNN

Although the R-CNN model uses pretrained CNNs to effectively extract image features, it is slow. Imagine that we select thousands of region proposals from a single input image: this requires thousands of CNN forward propagations to perform object detection. This massive computing load makes it infeasible to widely use R-CNNs in real-world applications.

9.2.2 Fast R-CNN

The main performance bottleneck of an R-CNN lies in the independent CNN forward propagation for each region proposal, without sharing computation. Since these regions usually have overlaps, independent feature extractions lead to much repeated computation. One of the major improvements of the fast R-CNN from the R-CNN is that the CNN forward prop- agation is only performed on the entire image.



Figures 9.5: Fast R-CNN

Its major computations are as follows:

1 Compared with the R-CNN, in the fast R-CNN the input of the CNN for feature extraction is the entire image, rather than individual region proposals. Moreover, this CNN is trainable. Given an input image, let the shape of the CNN output be $1 \times c \times h_1 \times w_1$.

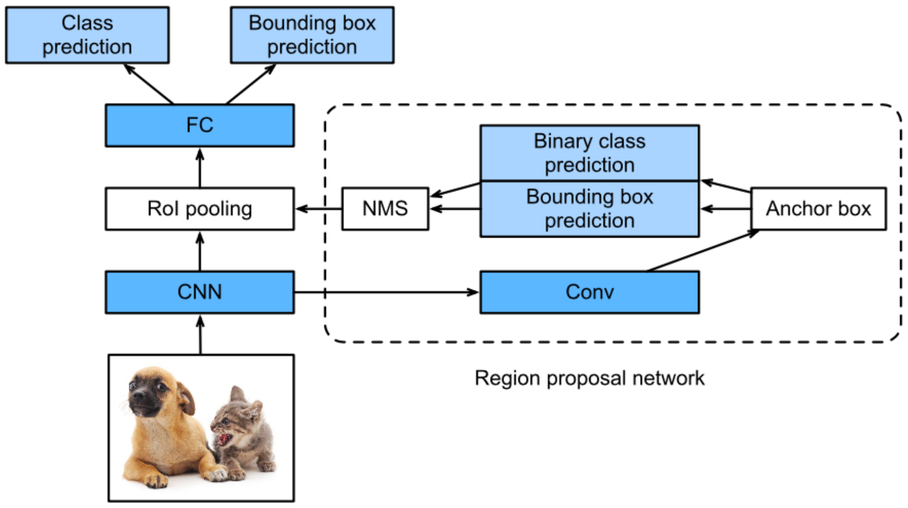
2 Suppose that selective search generates n region proposals. These region proposals (of different shapes) mark **regions of interest** (of different shapes) on the CNN output. Then these regions of interest further extract features of the same shape (say height h_2 and width w_2 are specified) in order to be easily concatenated. To achieve this, the fast R-CNN introduces the **region of interest (RoI)** pooling layer: the CNN output and region proposals are input into this layer, outputting concatenated features of shape $n \times c \times h_2 \times w_2$ that are further extracted for all the region proposals.

3 Using a fully connected layer,transform the concatenated features into an output of shape $n \times d$, where d depends on the model design.

4 Predict the class and bounding box for each of the n region proposals. More concretely, in class and bounding box prediction, transform the fully connected layer output into an output of shape $n \times q$ (q is the number of classes) and an output of shape $n \times 4$, respectively.

9.2.3 Faster R-CNN

To be more accurate in object detection, the fast R-CNN model usually has to generate a lot of region proposals in selective search. To reduce region proposals without loss of accuracy, the faster R-CNN proposes to replace selective search with a **region proposal network**.



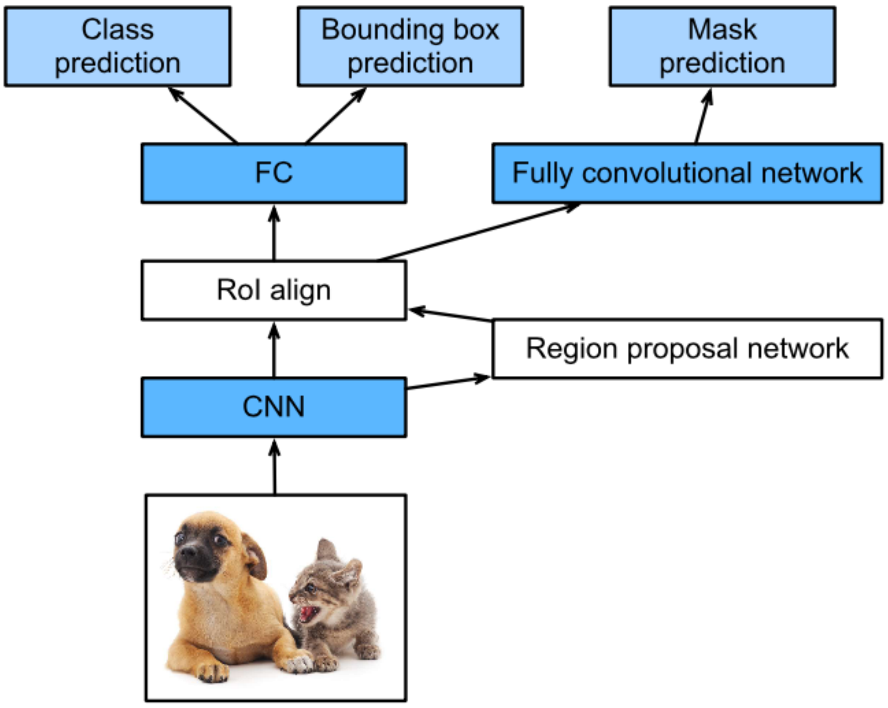
Figures 9.6: Faster R-CNN

The region proposal network works in the following steps:

1. Use a 33 convolutional layer with padding of 1 to transform the CNN output to a new output with c channels. In this way, each unit along the spatial dimensions of the CNN-extracted feature maps gets a new feature vector of length c.
2. Centered on each pixel of the feature maps, generate multiple anchor boxes of different scales and aspect ratios and label them.
3. Using the length-c feature vector at the center of each anchor box,predict the binary class (background or objects) and bounding box for this anchor box.
4. Consider those predicted bounding boxes whose predicted classes are objects. Remove overlapped results using non-maximum suppression. The remaining predicted bounding boxes for objects are the region proposals required by the region of interest pooling layer.

9.2.4 Mask R-CNN

In the training dataset, if pixel-level positions of object are also labeled on images, the mask R-CNN can effectively leverage such detailed labels to further improve the accuracy of object detection.



Figures 9.7: Mask R-CNN

The mask R-CNN replaces the region of interest pooling layer with the **region of interest (RoI) alignment** layer. This region of interest alignment layer uses bilinear interpolation to preserve the spatial information on the feature maps, which is more suitable for pixel-level prediction. The output of this layer contains feature maps of the same shape for all the regions of interest. They are used to predict not only the class and bounding box for each region of interest, but also the pixel-level position of the object through an additional fully convolutional network.

Chapter 10

Segmentation

10.1 Image Segmentation and Instance Segmentation

Image segmentation divides an image into several constituent regions. The methods for this type of problem usually make use of the correlation between pixels in the image. It does not need label information about image pixels during training, and it cannot guarantee that the segmented regions will have the semantics that we hope to obtain during prediction.

Instance segmentation is also called simultaneous detection and segmentation. It studies how to recognize the pixel-level regions of each object instance in an image. Different from semantic segmentation, instance segmentation needs to distinguish not only semantics, but also different object instances.

10.2 Full Convolutional Network

Semantic segmentation focuses on how to divide an image into regions belonging to different semantic classes. Different from object detection, semantic segmentation recognizes and understands what are in images in pixel level: its labeling and prediction of semantic regions are in pixel level.

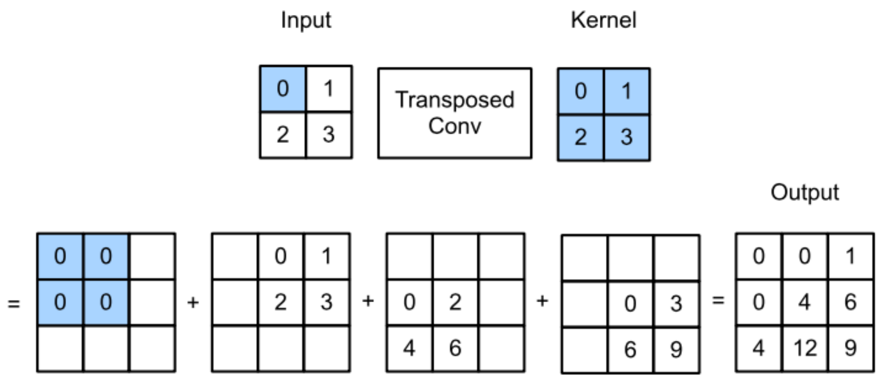
The code is on Github

10.2.1 Transposed Convolution

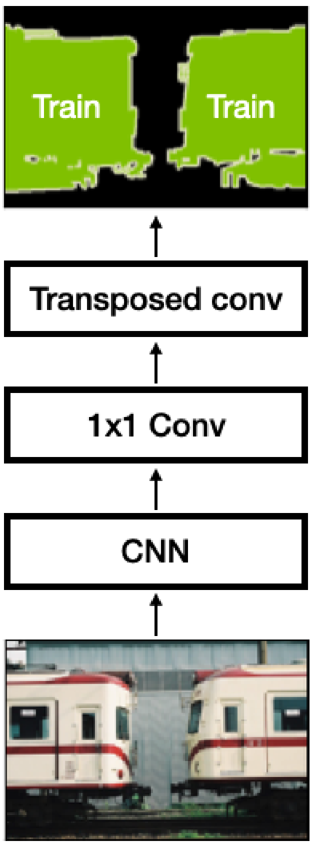
Transposed convolution is shown in Figure 10.1

10.2.2 Fully Convolutional Networks

A fully convolutional network (FCN) uses a convolutional neural network to transform image pixels to pixel classes. Figure 10.2 shows the fully convolutional network.



Figures 10.1: Transposed Convolution



Figures 10.2: FCN

Chapter 11

Generative Model

11.1 GAN

11.2 Diffusion Model

Part V

Natural Language Processing Practice

Chapter 12

Classificatrion

Chapter 13

Chat

Bibliography

[1] Aston Zhang et al. *Dive into Deep Learning*. 2021.

[2] Debra Cameron et al. *Learning GNU Emacs*. O'Reilly Media, Inc., 2004.

[3] Ian Goodfellow et al. *Deep Learning*. 2016.

[4] Yann LeCun et al. Gradientbased learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[5] D.E. Knuth. *The TEXbook*. Addison Wesley, 1986.

[6] D.E. Knuth. Typesetting concrete mathematics. *TUGboat*, 10(1):31–36, April 1989.

[7] Adrian Rosebrock. *Practical Python and OpenCV*. PyImageSearch, 2016.

[8] Adrian Rosebrock. *Deep Learning for Computer Vision with Python*. PyImageSearch, 2017.