FP16 and INT8 convolution in OpenCV DNN

Tianyou Song 12112519
Jinlong Zhang 12112528
Shiwen Cao 12113024

October 30, 2023

Abstract

Deep learning has achieved great success in computer vision and artificial intelligence applications. In order to achieve better performance, a large number of floating-point operations are performed during model training and inference processes. Traditionally, 32-bit float32 data types are used, but this consumes a significant amount of memory and computing resources. In modern times, with the introduction of semi precision floating point numbers (float16), this type of data with small memory usage and fast computation speed has gradually been applied to model training and inference, and has shown good performance. As an module that implements loading and reasoning for multiple models, OpenCV DNN currently does not support the float16 data type. In order to improve the inference efficiency of the DNN module, this article attempts to introduce the float16 data type into OpenCV DNN.

1 Background Introduction

1.1 OpenCV

OpenCV is an open source computer vision library that provides a wealth of tools and functions for processing image and video data. Currently, OpenCV supports multiple operating systems, including Windows, Linux, and Android, and can be developed and run on different platforms; And it supports multiple programming languages and provides interfaces for languages such as C++, Python, and Java.

1.2 OpenCV DNN

OpenCV DNN is one of the modules in the OpenCV library, specifically designed for deep learning and neural network inference. The DNN module provides functionality for loading, configuring, and running deep learning models for inference on images and data. As a relatively lightweight component, the DNN module has not undergone model training, so it can achieve better performance at runtime; At the same time, it provides built-in CPU and GPU acceleration without relying on third-party libraries.

1.3 Half precision floating-point number

Half precision float16 is a 16 bit floating-point data type that has advantages such as low memory usage and fast running speed. Compared to single precision floating-point numbers (float32), float16 only has 5-bit Exponent width and 10-bit Significant precision, which differ greatly from float32 in terms of precision and representation range. Float16 is a data type commonly used for deep learning training and inference. Due to the large amount of memory and computational resources occupied by floating-point operations in deep learning, the introduction of float16 will reduce the memory bandwidth to half of its original size.

In the current version of the OpenCV DNN module, support for the float16 data type is not implemented. In order to improve the efficiency of the DNN module and enable it to support more models, we attempted to introduce the float16 data type into the module.

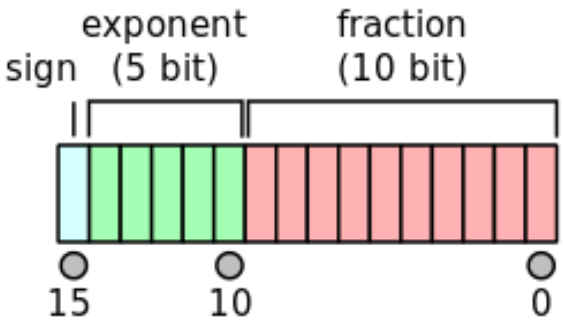

Figure 1: Storage structure of float16

1.4 Ficus

Ficus is a new functional language with first-class array support, and the Ficus NN library has implemented efficient float16 convolution operations. Therefore, we can try to learn from the implementation method in Ficus NN and port float16 convolution kernels from Ficus NN to OpenCV DNN.

2 Goal and Plan

2.1 Goal

This semester, we are first planning to introduce float16 into OpenCV DNN, enable it to support convolution operations for float16. In order to complete this goal, we divide it into the following tasks:

1. Understand the storage format of float16

2. Compare the performance of float16 and float32 in operations using the float16 data in *arm_neon.h*

3. Introducing float16 into the OpenCV DNN module to enable it to support convolutional operations of float16

4. Using some models to test the performance of float16 and the accuracy of the results

2.2 Timeline

We divide the task into roughly four stages:

1. Complete understanding and testing of float16 before October 29th
2. From October 30th to November 26th, read the codes for the Ficus NN and OpenCV DNN modules
3. Implement the introduction of float16 data types in DNN before December 24th
4. Complete basic testing and report on January 7, 2024

Figure 2: Timeline

2.3 Current Progress

So far, we have completed a basic understanding of the types of float16 and tested its computational ability.

Due to the low memory usage of float16, it is significantly faster than float32 when reading and writing, so float16 will use less time under the same number of basic operations. When using SIMD to accelerate floating-point operations, the number of data retrieved simultaneously from float16 is twice that of float32. For example, if SIMD can process four float32 data simultaneously, it can process eight float16 data simultaneously. Therefore, the efficiency of float16 at runtime is approximately twice that of float32.

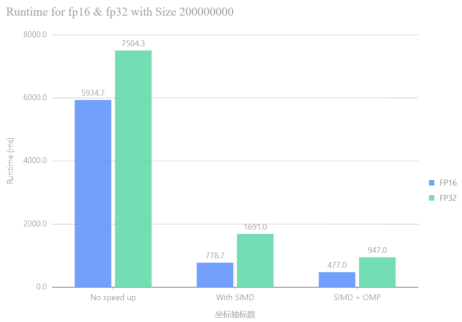

Figure 3: Compare fp16 and fp32 using SIMD + OMP

At the same time, we compared the accuracy of float16 and float32 when storing floating-point numbers. Due to the use of fewer digits, the accuracy of float16 is significantly lower than that of float32. Through testing, it was found that the smaller the data, the greater the data error of float16.

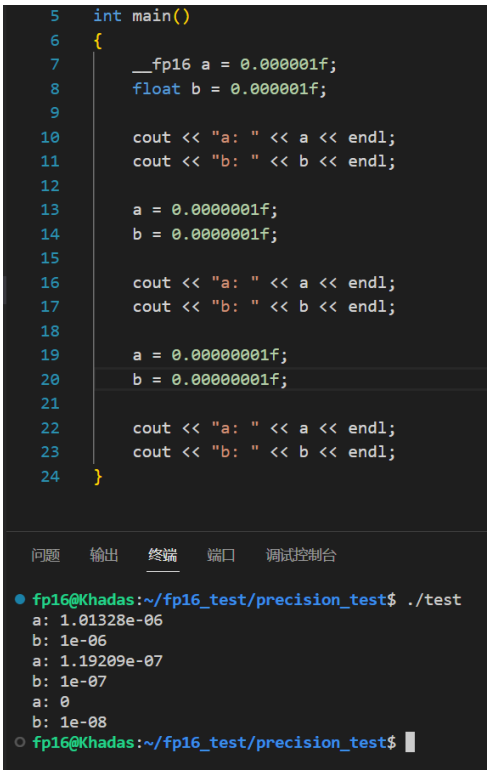

Figure 4: Compare fp16 and fp32's precision

3 Risk and Challenge

There are currently three potential issues that may arise in this project.

1. Precision problem: During the testing process, float16 is prone to errors in stored data, so the accumulation of accuracy errors during the calculation process can lead to unstable final results and errors.

2. Numeric overflow: Compared to float32, float16 uses fewer bytes for storage, making it much less data scoped and more prone to data overflow.

3. Hardware compatibility issues: At present, only some CPUs and GPUs support the float16 data type, and the definition of float16 is not completely the same for different hardware, making it difficult to achieve multi platform support for float16 operations.

To solve these problems, mixed precision operations and declaration of macro definitions can be used.

In order to reduce the accuracy error and numerical overflow caused by float16, it can be attempted to use float16 during computation to improve computational efficiency, while using float32 to maintain numerical accuracy and reduce errors. For different types of

hardware, we can use different macro definitions to introduce code that supports float16 operations, which ensures that compilation errors do not occur at runtime.

4 Future work

We plan to implement the porting of float16 convolution operations this semester, and will continue to make revisions to improve the calculation speed and accuracy of float16 convolution; After implementing float16, we will begin to improve the efficiency of int8 in the DNN module, making the functionality of OpenCV DNN more comprehensive.