CS/EE147

Final Project Report

Implementing cuDNN in a 2-D CNN Classifier

Team Member: Boyi Li

1. Overview

Convolutional Neural Networks (CNNs) have become a prominent tool in the fields of AI, image processing, and computer vision. These algorithms heavily rely on matrix and tensor operations to extract meaningful features from images. Traditionally, CPUs process data in a sequential manner, which can limit the efficiency of CNN computations. However, the advent of Graphics Processing Units (GPUs) has revolutionized the landscape, offering parallel processing capabilities that are ideally suited for such tasks. To evaluate and compare the performance of different implementations, I plan to construct an image classifier using two variations of TensorFlow: the regular version, which primarily utilizes CPU processing, and GPU-TensorFlow, which leverages the power of cuDNN for GPU-accelerated computations. By implementing these frameworks side by side, I aim to analyze their respective efficiencies and determine the impact of CPU and GPU utilization on the overall performance of the image classifier. This investigation will provide valuable insights into the advantages and trade-offs associated with using CPU or GPU for CNN-based tasks.

2. How is the GPU used to accelerate the application?

Each neuron in the network can be considered an independent unit of computation. Therefore, in the context of GPUs, each neuron's calculation can be thought of as a thread. For convolutional layers, each convolution operation is independent and can be performed in parallel. These operations are usually implemented as matrix multiplications which are highly parallelizable on GPU. The convolution operation can be divided into multiple smaller operations, each of which can be processed by a thread.

Also, instead of processing one training example at a time, multiple training examples can be processed simultaneously.

3. Implementation details

By implementing TensorFlow-GPU, we can automatically run our model on GPU by using some simple command:

```
tf.device('/GPU:0'):
```

And the model I built is 3 layers of Convolutional Neural Network which includes 56,320 trainable parameters, which means there are 56,320 weights. After modify the model for the CIFAR-10 dataset, I got a model with 122,570 trainable parameters and no non-trainable parameters.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896

 max_pooling2d (MaxPooling2D  (None, 15, 15, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 64)          36928

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 64)                65600

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0
```

4. **Documentation on how to run your code**

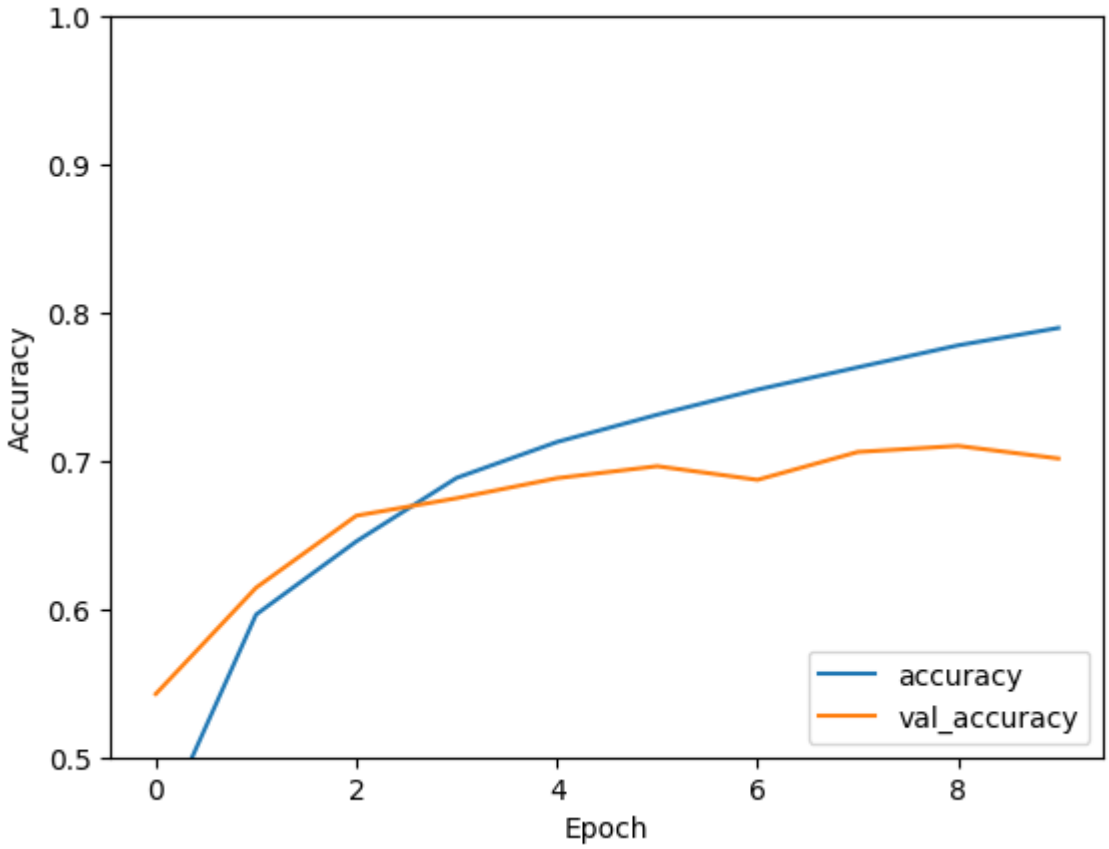
   The code is in Jupyer Notebook, so it is really easy to run. The requirement is to have a GPU which supports CUDA and cuDNN also need to install TensorFlow.

```
import tensorflow as tf
import time
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
```

5. **Evaluation/Results**

After the two different implementations with CPU and GPU respective, the CPU comes with a slightly higher accuracy, the GPU comes with a much faster run time. This is the advantage of GPU's parallelism.

```
313/313 - 1s - loss: 0.9225 - accuracy: 0.7016 - 1s/epoch - 5ms/step
```



6. **Problems faced:**
    - Because of the lack of knowledge of CUDA C, I can't implement this algorithm by implementing a kernel.cu.
    - It's hard to figure out the details of how GPU is processing the data, I just can judge everything by the run time.

7. **Percentage Breakdown**

cuDNN Implementation	**Boyi Li: 100%**
TensorFlow Implementation	**Boyi Li: 100%**
Project Report	**Boyi Li: 100%**