Project 2: Convolutional Neural Networks

Ian Cox and Tanner Mengel

March 5, 2023

1 Introduction

Convolutional neural networks implement multi-dimensional, locally connected neurons with shared weights to detect features in input data. The weights of these neurons are updated via back-propagation. Along with feature detection layers, max pooling and flattening layers are used to down sample output from kernels and flatten 2 dimensional output to be feed into a fully connected layer. The goal of this project is to implement classes for flattening, max pooling, and 2D convolution layers. These classes will be added onto the project 1 frame work which and will need to be integrated into the previously developed neuron and neural network. Several examples with different kernel size and input layers are compared to the output of TensorFlow to show that the new layers are working as expected.

2 Assumptions

The only assumptions made in this code is that the custom models must be sequential, meaning that there are no neurons which skip layers, and that for the 2DConvolutional layer, the kernel must be square. This helps to make calculations a little more simple. Also, the stride in the Max Pooling Layer is assumed to be the same as the dimension of the pooling, creating no overlap between pools.

3 Issues

There were no issues that remain unsolved with the final version of the code.

4 How to run the code

Unzip the tar-ball:

```
tar -xzf project2_code.tar.gz
```

Run the examples:

```
python3 project2.py <learning rate> <exampleX>
```

Where the learning rate is the first argument and exampleX is a string where X is 1, 2, or 3.

5 Results

The results for each example are shown below. The results include comparisons for the weights (bias) after 1 epoch of training and the prediction for both our custom CNN and an identical model implemented in TensorFlow.

Figures 1, 2, and 3 show the comparison for example 1, example 2, and example 3, respectively. This was executed with a learning rate of 0.1 and the following command:

```
python3 project2.py 0.1 exampleX
```

```
Value Comparison
1st convolutional layer, 1st kernel weights (bias):
Custom CNN: [[0.77128 0.02072 0.63361]
 [0.74876 0.49847 0.22476]
 [0.19803 0.76049 0.16907]]      (0.08827)
Keras CNN: [[0.77128 0.02072 0.63361]
 [0.74876 0.49847 0.22476]
 [0.19803 0.76049 0.16907]]      (0.08827)
Fully connected layer weights (bias):
Custom CNN: [0.68522 0.95325 0.00381 0.51205 0.81247 0.61239 0.72161 0.29174 0.91763]    (0.71442)
Keras CNN: [0.68522 0.95325 0.00381 0.51205 0.81247 0.61239 0.72161 0.29174 0.91763]     (0.71442)
Output Comparison
Custom CNN: 0.99671
Keras CNN: 0.99671
```

Figure 1: Results from example 1.

where X is 1, 2, or 3. These screen shots are taken from the output of running the code.

As seen in the results, the Custom CNN is identical to the TensorFlow implemented model. Therefore the Flattening, MaxPooling, and ConvLayer classes are working and the back-propagation is implemented correctly.

```
Value Comparison
1st convolutional layer, 1st kernel weights (bias):
Custom CNN: [[0.77132 0.02075 0.63365]
 [0.7488  0.49851 0.2248 ]
 [0.19806 0.76053 0.16911]]      (0.91777)
Keras CNN: [[0.77132 0.02075 0.63365]
 [0.7488  0.49851 0.2248 ]
 [0.19806 0.76053 0.16911]]      (0.91777)
1st convolutional layer, 2nd kernel weights (bias):
Custom CNN: [[0.08834 0.68536 0.95339]
 [0.00395 0.51219 0.81262]
 [0.61253 0.72176 0.29188]]      (0.71458)
Keras CNN: [[0.08834 0.68536 0.95339]
 [0.00395 0.51219 0.81262]
 [0.61253 0.72176 0.29188]]      (0.71458)
2nd convolutional layer weights (bias):
Custom CNN: [[[0.54254 0.14217]
  [0.37334 0.67413]
  [0.44183 0.43401]]

 [[0.61777 0.51314]
  [0.6504  0.60104]
  [0.80522 0.52165]]

 [[0.90865 0.31924]
  [0.09046 0.3007 ]
  [0.11398 0.82868]]]    (0.04690)
Keras CNN: [[[0.54254 0.14217]
  [0.37334 0.67413]
  [0.44183 0.43401]]

 [[0.61777 0.51314]
  [0.6504  0.60104]
  [0.80522 0.52165]]

 [[0.90865 0.31924]
  [0.09046 0.3007 ]
  [0.11398 0.82868]]]    (0.04690)
Fully connected layer weights (bias):
Custom CNN: [0.62582 0.54712 0.81882 0.19848 0.85638 0.35118 0.75418 0.29549 0.88347]    (0.32504)
Keras CNN: [0.62582 0.54712 0.81882 0.19848 0.85638 0.35118 0.75418 0.29549 0.88347]    (0.32504)
Output Comparison
Custom CNN: 0.9965111680856696
Keras CNN: 0.9965111613273621
```

Figure 2: Results from example 2.

```
Value Comparison
1st convolutional layer, 1st kernel weights (bias):
Custom CNN: [[0.77132 0.02075 0.63365]
 [0.7488  0.49851 0.2248 ]
 [0.19806 0.76053 0.16911]]      (0.91777)
Keras CNN: [[0.77132 0.02075 0.63365]
 [0.7488  0.49851 0.2248 ]
 [0.19806 0.76053 0.16911]]      (0.91777)
1st convolutional layer, 2nd kernel weights (bias):
Custom CNN: [[0.08834 0.68536 0.95339]
 [0.00395 0.51219 0.81262]
 [0.61252 0.72175 0.29187]]      (0.71457)
Keras CNN: [[0.08834 0.68536 0.95339]
 [0.00395 0.51219 0.81262]
 [0.61252 0.72175 0.29187]]      (0.71457)
Fully connected layer weights (bias):
Custom CNN: [0.54252 0.14215 0.37332 0.67411 0.44181 0.43399 0.61775 0.51312 0.65038
 0.60102 0.8052  0.52163 0.90863 0.31922 0.09044 0.30068 0.11397 0.82866]        (0.04688)
Keras CNN: [0.54252 0.14215 0.37332 0.67411 0.44181 0.43399 0.61775 0.51312 0.65038
 0.60102 0.8052  0.52163 0.90863 0.31922 0.09044 0.30068 0.11396 0.82866]        (0.04688)
Output Comparison
Custom CNN: 0.99981
Keras CNN: 0.99981
```

Figure 3: Results from example 3.