1. Introduction

Building off the artifical neural network library created in Project 1, several modifications are introduced to build a convolutional neural network. Specifically, several classes are introduced, namely

`ConvolutionalLayer`

`MaxPoolingLayer`

`FlattenLayer`

An additional method `addLayer` is introduced in the `NeuralNetwork` class to allow for a variable number of layers in the neural network with different layer types.

In this project, the convolution layer is restricted to 2-dimensional convolution with a stride of 1 and padding is set to 'valid'. The max pooling layer is also restricted to 2-dimensional with the stride is always the same as the filter size and no padding is needed.

The options for activation function are logistic function (sigmoid) and linear function. The options for loss function are mean squared error and binary cross entropy.

Code examples are run using the TensorFlow with the keras API (tf.keras) to validate the correctness of the convolutional neural network library. The input, output, weights and bias are generated using `parameter.py` with a random seed specified so the results are reproducible.

2. Assumptions/choices made

- The input for each layer type (except for the fully connected layer) has a dimension of (w, h, c) where w is the width of the input, h is the height of the matrix and c is the number of input channels.
- The fully connected layer accepts an input with a dimension of (n, m), where n is the number of neurons in the fully connected layer and m is the size of the flattened input.
- The kernel for each convolutional layer has a dimension of (k, kw, kh) where k is the number of kernel, kw is the width of the kernel and kh is the height of the kernel.
- The argument of the `addLayer` method accepts 4 layer types, namely:
  - "Conv2D" : 2-dimensional convolutional layer
  - "Dense" : fully connected layer
  - "MaxPooling2D" : 2-dimensional max pooling layer
  - "Flatten" : flatten layer

3. Problems/Issues

There is a slight discrepancy between the solution of the project code and the solution of the TensorFlow keras API. This may be due to round-off errors.

4. How to run the code

You can run the project code with examples using the following command:

```
$ python project2.py [example1|example2|example3]
```

You can choose one example to run with either example1, example2 or example3.

To run the code examples with the TensorFlow keras API, use the following commands:

```
$ python tensorflowtest_example1.py
$ python tensorflowtest_example2.py
$ python tensorflowtest_example3.py
```

5. Results

The results from each run of one-step backpropagation using the project code are compared to the results from one run of Tensorflow keras API with the same input paramters.

The code examples are:

- Example 1: Run a network with a 5×5 input, one 3×3 convolutional layer with a single kernel, a flatten layer and a single neuron for the output.
- Example 2: Run a network with a 7×7 input, one 3×3 convolutional layer with two kernels, another 3×3 convolutional layer with a single kernel, a flatten layer and a single neuron for the output.
- Example 3: Run a network with an 8×8 input, one 3×3 convolutional layer with two kernels, a 2×2 max pooling layer, a flatten layer and a single neuron for the output.

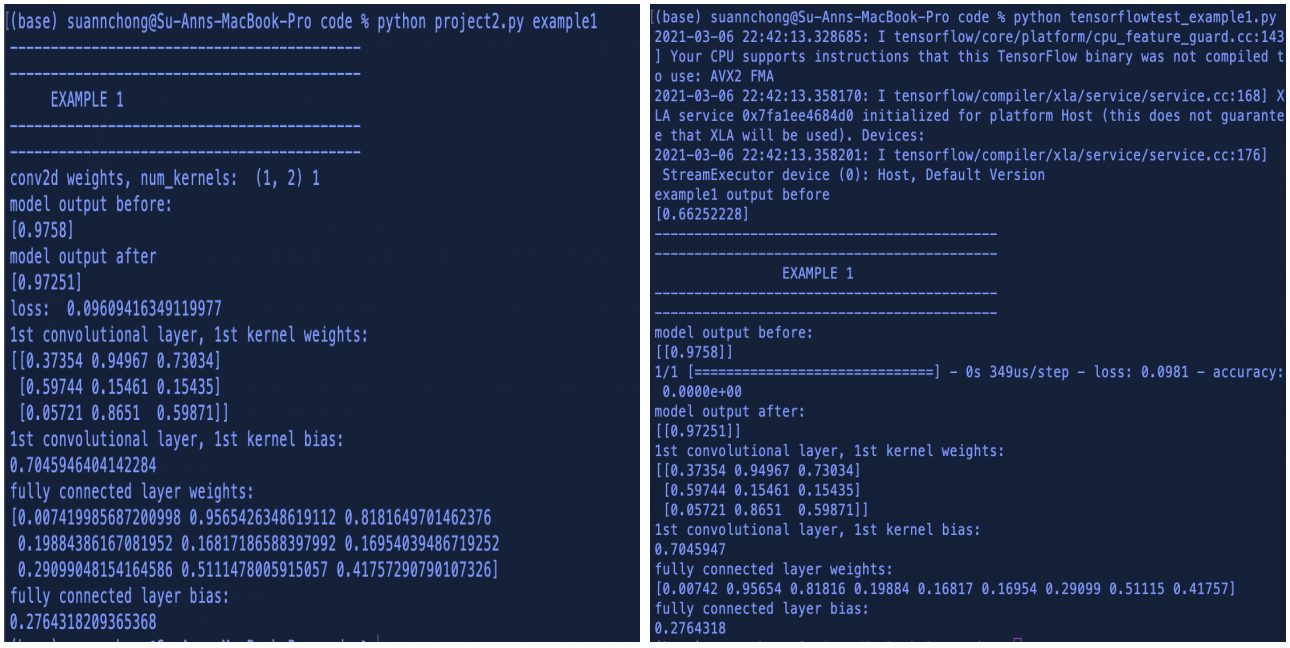

Figure 1. Screenshots of the project solution (on left) and the TensorFlow keras API solution (on right) for Example 1.

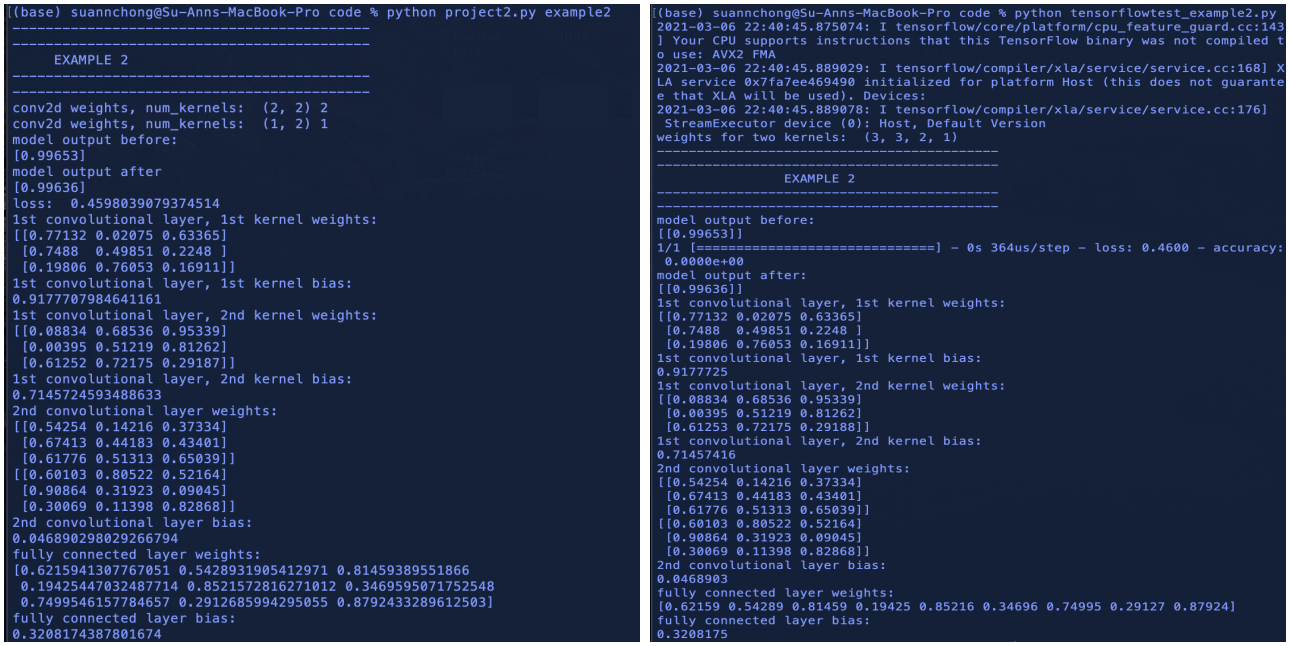

FIGURE 2. Screenshots of the project solution (on left) and the TensorFlow keras API solution (on right) for Example 2.

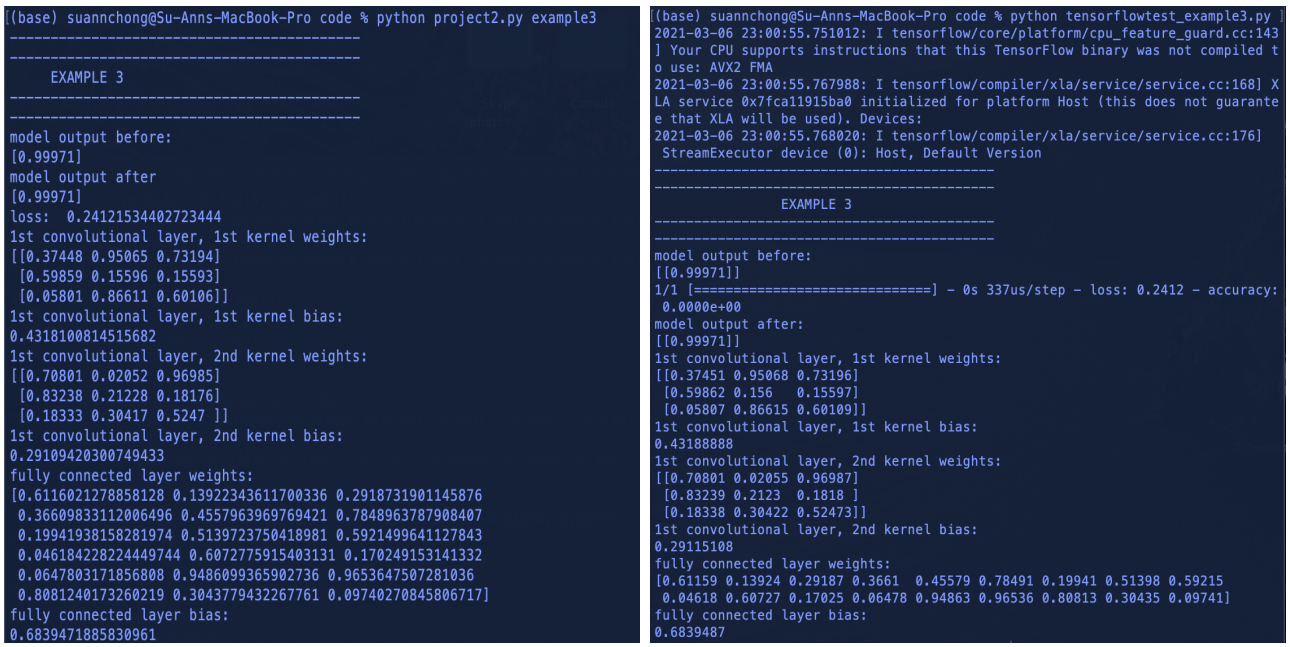

FIGURE 3. Screenshots of the project solution (on left) and the TensorFlow keras API solution (on right) for Example 3.