Project 2

COSC 525: Deep Learning

Harshvardhan, Yu Jiang*

March 6, 2023

1 Introduction

Convolutional Neural Network (CNN) is a type of neural network designed specifically for image and video processing. It is inspired by the way in which the visual cortex of the brain processes information. CNN is effective for image classification, object detection, and image segmentation.

The most important component of a CNN is the convolutional layer, which applies a set of filters or kernels to the input image to extract task-relevant features. Typically, these filters are learned via backpropagation during training, where the network is trained on a large labeled image dataset.

A typical CNN architecture also includes pooling layers, which downsample the feature maps generated by the convolutional layers, and fully connected layers, which use the extracted features to make an image prediction.

In our project, we implement CNN with three layers, a Convolotional layer, a MaxPooling layer and a Flattern layer. For the Convolutional layer, it is restricted the layer to 2d convolutions and is initialized with the number of kernels in the layer, the size of the kernel, the activation function for all the neurons in the layer, the dimension of the inputs, the learning rate. For the MaxPooling layer, it is restricted the layer to 2d max pooling and is initialized with the size of the kernel, and the dimension of the inputs. For the Flattern layer, it is only initialized with the input size. In the end, we compare three different examples with results obtained from using keras with our own model results in order to verify accuracy.

This report is organized as follows: in Section 2, we will introduce our assumptions and choices; in Section 3, we will reflect some problems and limitations; in Section 4, we will demonstrate code processing and in Section 5 we will discuss our results and findings.

2 Assumptions and Choices

2.1 Assumptions

In this project, we have different assumptions for each layer.

For **ConvolutionalLayer** class, we have three assumptions as follows:

- Kernel is a square,
- The stride is always 1,

*Business Analytics and Statistics, Haslam College of Business, University of Tennessee, Knoxville.

- Padding is set to valid.

Similarly, for **MaxPoolingLayer** class, we have three assumptions as follows:

- Kernel is a square,
- The stride is always the same as the filter size,
- No padding is needed.

2.2 Activation Functions

In this project, we choose the Sigmoid function as our activation function.

The the Sigmoid activation function, also known as logistic activation function, is a common nonlinear activation function. It is defined as follows:

$$\varphi(x) = \frac{1}{1 + e^{-x}}$$

The logistic function maps any real-valued input to a value between 0 and 1, making it useful for binary classification problems. The objective is to predict a binary output, 0 or 1.

2.3 Loss Functions

In this project, we choose the squared error as our loss function.

The squared error loss function, also known as mean squared error (MSE) loss, is a popular loss function for regression problems. It is defined as the average squared difference between the predicted output of a model and the actual output.

$$E = \frac{1}{N} \sum_{i} (\hat{y}_i - y_i)^2$$

2.4 Weight Update

The equations of backpropagation for weight update can be defined as follows.

$$\delta^{L_j} = \frac{\partial E_{total}}{\partial out_{L_j}} \times \frac{\partial out_{L_j}}{\partial net_{L_j}}$$

$$\delta^{l_j} = \sum_{k} w_{l_j l+1_k} \times \delta^{l+1_k} \times \frac{\partial out_{l_j}}{\partial net_{l_j}}$$

$$\frac{\partial E_{total}}{\partial w_{l_j l+1_k}} = \delta^{l+1_k} \times out_{l_j}$$

$$w_i = w_i - \alpha \frac{\partial E}{\partial w_i}$$

To update the weights in a neural network, each neuron computes its own δ by multiplying the derivative of the activation function with the error signal. The neuron then sends the vector of $w\delta$ to the `FullyConnectedLayer`. The layer collects these vectors from all the neurons, sums them up, and returns the resulting vector to the `NeuralNetwork`. The `NeuralNetwork` uses this vector to update the weights of the network.

3 Problems and Limitations

Our code had certain minor difficulties that we couldn't fully resolve. In Examples 2 and 3, our weights are slightly different than the ones obtained using Keras. The differences are in fourth and third decimal places, respectively.

We also want to highlight that our code must be executed via the Notebook, not the Terminal. We preferred this approach due to our limited understanding of the Terminal commands and the beauty of Jupyter notebooks that allowed us to interactively see the results while executing the program.

4 Process of Running Code

In our case, we have written everything together in a single Jupyter Notebook which can be executed sequentially. First, we load the libraries. Then, we create the classes for each functionality as was required in Projects 1 and 2. Finally, we compare the results between Keras and our model.

The code with the output can be seen in the Jupyter Notebooks (`Project 2.ipynb`) and the HTML file (`Project 2.html`), produced via Quarto.

5 Results Discussion

This section will compare the results found using our code and Keras library.

5.1 Example 1

This example has a 5x5 input, one 3x3 convolution layer with a single kernel, a flattened layer, and a single neuron for the output. It can be noted from Figure 1 and Figure 2, our results match exactly.

```
Output before backprop: [0.63992102]
----- Keras Model after Backpropagation (Example 1)-----

2023-03-06 15:18:00.871089: I tensorflow/core/platform/cpu_feature_guard.cc:193] Th:
To enable them in other operations, rebuild TensorFlow with the appropriate compile

1/1 [==============================] - 0s 194ms/step - loss: 0.1269 - accuracy: 0.0
1/1 [==============================] - 0s 53ms/step
Output from Keras model after backpropagation is [[0.94939]]
1st conv layer weight is
[[[[ 0.10745  0.91042  0.49045]
   [ 0.38067  0.23018  0.73869]
   [ 0.42771  0.53635 -0.01125]]]]
1st conv layer weight is
[0.56307]
FC layer weight is
[ 0.34338  0.34923  0.67778  0.42426  0.08943  0.1702   0.43194 -0.20615
  0.39757]
FC layer bias is
[0.39325]
```

Figure 1: Output for Example 1 from Keras.

```
----- Our Model after Backpropagation (Example 1) -----
Output from our model after backpropagation is: [[0.94972]]
layer 0 is a Conv layer, weights shape = (1, 1, 3, 3) and bias shape = (1,). Weights
[[[[ 0.10745  0.91042  0.49045]
   [ 0.38067  0.23018  0.73869]
   [ 0.42771  0.53635 -0.01125]]]]
[0.56307]
layer 1 is a Flatten layer without weights
layer 2 is a FC layer, weights shape = (9,) and bias shape = (1,). Weights and bias
[ 0.34338  0.34923  0.67778  0.42426  0.08943  0.1702   0.43194 -0.20615
  0.39757]
[0.39325]
```

Figure 2: Output for Example 1 from our model.

5.2 Example 2

This example has a 7x7 input, one 3x3 convolution layer with two kernels, another 3x3 convolution layer with a single kernel, a flattened layer and a single neuron for the output. Our outputs match up to the fourth decimal place, with slight discrepancies at the fifth decimal place. See Figure 3 and Figure 4.

5.3 Example 3

This example has 8x8 input, one 3x3 convolution layer with two kernels, a 2x2 max pooling layer, a flattened layer, and a single neuron for the output. See Figure 5 and Figure 6. In this case, our results match up to

```
label: [0.36371]
-----Keras Model after Backpropagation (Example 2)-----
1/1 [==============================] - 0s 172ms/step - loss: 0.3931 - accuracy: 0.0
1/1 [==============================] - 0s 43ms/step
keras model output after backprop is [[0.00104]]
1st conv layer weight is
[[[[0.56994 0.43787 0.98811]
   [0.10116 0.20855 0.16035]
   [0.65284 0.25239 0.46602]]]


 [[[0.24351 0.15868 0.10935]
   [0.65611 0.1374  0.19631]
   [0.36792 0.82073 0.09616]]]]
1st conv layer bias is
[0.83742 0.09441]
2nd conv layer weight is
[[[[0.97344 0.46631 0.9737 ]
   [0.60247 0.73619 0.03673]
   [0.27976 0.11789 0.29311]]

  [[0.11632 0.3149  0.41187]
   [0.06113 0.69016 0.56356]
   [0.26302 0.52019 0.09156]]]]
2nd conv layer bias is
[0.57278]
FC layer weight is
[-0.2247  -0.83551 -0.48674 -1.02226 -0.43772 -0.86473 -0.97078 -0.56753
 -1.13398]
FC layer bias is
[-0.32594]
```

Figure 3: Output for Example 2 from Keras.

```
----- Our Model after Backpropagation (Example 2) -----
Output from our model after backpropagation is: [[0.00104]]
layer 0 is a Conv layer, weights shape = (2, 1, 3, 3) and bias shape = (2,). Weights
[[[[0.56992 0.43826 0.98801]
   [0.1017  0.20854 0.16094]
   [0.65281 0.25293 0.46601]]]


 [[[0.24378 0.15826 0.10963]
   [0.65578 0.13753 0.19584]
   [0.3681  0.82037 0.09638]]]]
[0.83607 0.09422]
layer 1 is a Conv layer, weights shape = (1, 2, 3, 3) and bias shape = (1,). Weights
[[[[0.97157 0.46366 0.97167]
   [0.59993 0.73423 0.03417]
   [0.27788 0.11521 0.29107]]

  [[0.11384 0.31299 0.40918]
   [0.05923 0.68744 0.56159]
   [0.26046 0.51826 0.08887]]]]
[0.57312]
layer 2 is a Flatten layer without weights
layer 3 is a FC layer, weights shape = (9,) and bias shape = (1,). Weights and bias
[-0.22442 -0.83527 -0.48649 -1.02196 -0.43751 -0.86449 -0.97057 -0.56737
 -1.13374]
[-0.32562]
```

Figure 4: Output for Example 2 from our model.

the second decimal place.

```
label: [0.19658]
-----Keras Model after Backpropagation (Example 3)-----
1/1 [==============================] - 0s 154ms/step - loss: 0.6451 - accuracy: 0.0
1/1 [==============================] - 0s 39ms/step
Keras model output after backprop is [[0.99954]]
1st conv layer weight is
[[[[0.3613  0.81864 0.08867]
   [0.83532 0.08839 0.97368]
   [0.46092 0.9748  0.59834]]]


 [[[0.73691 0.03401 0.28045]
   [0.11179 0.29337 0.11395]
   [0.31612 0.40792 0.86221]]]]
1st conv layer bias is
[0.68129 0.56232]
FC layer weight is
[ 0.23038  0.48557  0.05873  0.53793  0.8948   0.28099  0.63399  0.09414
  0.68126  0.25137  0.14858  0.54896 -0.01321  0.79247 -0.02798  0.6407
  0.23616  0.69747]
FC layer bias is
[0.92381]
```

Figure 5: Output for Example 3 from Keras.

```
----- Our Model after Backpropagation (Example 3) -----
Output from our model after backpropagation is: [[0.99949]]
layer 0 is a Conv layer, weights shape = (2, 1, 3, 3) and bias shape = (2,). Weights
[[[[0.36844 0.81934 0.09561]
   [0.83743 0.09477 0.9759 ]
   [0.46767 0.97559 0.60441]]]


 [[[0.73612 0.03587 0.27964]
   [0.11721 0.2927  0.11492]
   [0.31595 0.41138 0.86064]]]]
[0.68531 0.55944]
layer 1 is a Max Pooling layer without weights
layer 2 is a Flatten layer without weights
layer 3 is a FC layer, weights shape = (18,) and bias shape = (1,). Weights and bia
[ 0.22502  0.48283  0.05397  0.53595  0.88895  0.27833  0.62816  0.09185
  0.67719  0.25242  0.14616  0.55032 -0.01551  0.79188 -0.03224  0.64272
  0.23377  0.69929]
[0.92134]
```

Figure 6: Output for Example 3 from our model.