Convolutional Neural Network

Carlos Matheus Barros da Silva, *Computer Engineering Bachelor Student of ITA*
Prof. Marcos Ricardo Omena de Albuquerque Máximo

***Abstract*—This paper evaluate a Convolutional Neural Network created using Keras. It was evaluated in a data set of handwritten decimals numbers, in which the Neural Network had to predict correctly the numbers.**

It was observed that the Convolutional Neural Network worked as expected. The number prediction was very accurate having 98.7% of accuracy on the data set.

***Index Terms*—Keras, Neural Network, Convolutional Neural Network**

I. Introduction

Neural networks (NN) are computing systems vaguely inspired by the biological neural networks and astrocytes that constitute animal brains. The neural network itself is not an algorithm, but rather a framework for many different machine learning algorithms to work together and process complex data inputs. Such systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules. For example, an image recognition, they might learn to identify images that contain cats by analyzing example images that have been manually labeled as "cat" or "no cat" and using the results to identify cats in other images. They do this without any prior knowledge about cats, for example, that they have fur, tails, whiskers and cat-like faces. Instead, they automatically generate identifying characteristics from the learning material that they process.

Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, Theano, or PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), and its primary author and maintainer is François Chollet, a Google engineer. Chollet also is the author of the XCeption deep neural network model.

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery.

CNNs are regularized versions of multilayer perceptrons. Multilayer perceptrons usually refer to fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "fully-connectedness" of these networks make them prone to overfitting data. Typical ways of regularization includes adding some form of magnitude measurement of weights to the loss function. However, CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble more complex patterns using smaller and simpler patterns. Therefore, on the scale of connectedness and complexity, CNNs are on the lower extreme.

They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics.

II. Neural Network Implementation

The implementation was based on the file *lenet5*. The essence of the implementation can be seen on the Code 1

```
def make_lenet5():
    model = Sequential()

    # 1 layer:
    nf = 6
    sx = sy = 1
    fx = fy = 5
    model.add(layers.Conv2D(
        filters=nf,
        kernel_size=(fx, fy),
        strides=(sx, sy),
        activation=activations.tanh,
        input_shape=(32, 32, 1)
        )
    )

    # 2 layer:
    nf = 6
    sx = sy = 2
    # fx = fy = 2
    px = py = 2
    model.add(layers.AveragePooling2D(
        pool_size=(px, py),
        strides=(sx, sy))
    )

    # 3 layer:
    nf = 16
    sx = sy = 1
    fx = fy = 5
    model.add(layers.Conv2D(
        filters=nf,
        kernel_size=(fx, fy),
        strides=(sx, sy),
        activation=activations.tanh,
        input_shape=(14, 14, 1)
        )
    )

    # 4 layer:
    nf = 16
    sx = sy = 2
    # fx = fy = 2
    px = py = 2
    model.add(layers.AveragePooling2D(
        pool_size=(px, py),
        strides=(sx, sy))
    )

    # 5 layer:
    nf = 120
    sx = sy = 1
    fx = fy = 5
    model.add(layers.Conv2D(
        filters=nf,
        kernel_size=(fx, fy),
        strides=(sx, sy),
        activation=activations.tanh,
        input_shape=(5, 5, 1)
        )
    )
```

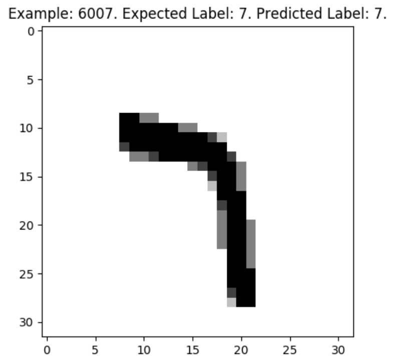

Fig. 1. Example of a correctly classified test case, was expected 7 and was predicted as 7.

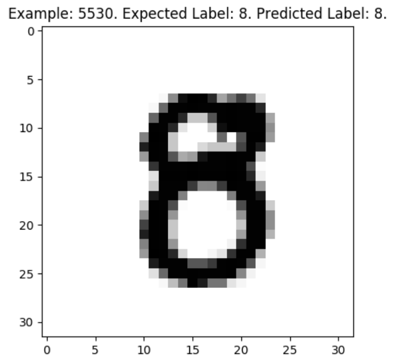

Fig. 2. Example of a misclassified test case, was expected 8 and was predicted as 8.

```
    )

    # 6 layer
    num_neurons = 84
    model.add(layers.Flatten())
    model.add(layers.Dense(
        units=num_neurons,
        activation=activations.tanh)
    )

    # 7 layer
    num_neurons = 10
    # model.add(layers.Flatten())
    model.add(layers.Dense(
        units=num_neurons,
        activation=activations.softmax)
    )

    return model
```

Code 1. Code of Convolutional Neural Network *lenet5*

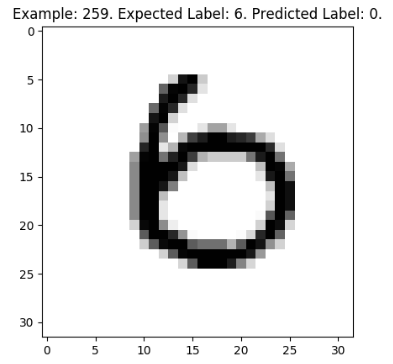

Fig. 3. Example of a misclassified test case, was expected 6 and was predicted as 0.

III. Neural Network Analysis

A. LeNet-5 Convolutional Neural Network Analysis

In order to evaluate the Neural Network, it was trained to identify the MNIST dataset. This dataset is a big set of images that represents decimal numbers.

The LeNet-5 performed very well on test cases having an acuracy of 98.7% on a set of 10000 test cases. The output for the file *evaluate lenet5* can be seen on Code 2.

On Image 1 and 2 it is possible to see two examples of correctly classified test cases.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 6)         156
_________________________________________________________________
average_pooling2d_1 (Average (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 10, 10, 16)        2416
_________________________________________________________________
average_pooling2d_2 (Average (None, 5, 5, 16)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 1, 1, 120)         48120
_________________________________________________________________
flatten_1 (Flatten)          (None, 120)               0
_________________________________________________________________
dense_1 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_2 (Dense)              (None, 10)                850
=================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
_________________________________________________________________

   32/10000 [..............................] - ETA: 22s
  256/10000 [..............................] - ETA: 4s
  480/10000 [>.............................] - ETA: 3s
...
 9664/10000 [===========================>..] - ETA: 0s
 9856/10000 [============================>.] - ETA: 0s
10000/10000 [==============================] - 3s 320us/
    step
Test loss: 0.04253822890962474
Test accuracy: 0.987
```

Code 2. Output for *evaluate lenet5*

As said before, in 1.3% of the test cases ware misclassified. In Image 3 and Image 4 is possible to see two examples of test cases of misclassification.

B. LeNet-5 Convolutional Neural Network Analysis on TensorBoard

In *Tensor Board* it was possible to see that occured a fast decrase on loss function and a rapidly accuracy to converge in more than 98% of acuracy. On Image 5 and Image 6 it is possible to see the graphs representing the converge of *accuracy* and *loss function* respectively.

TensorBoard also provided an overview of the Neural Network format, this is denoted on Image 7 and Image 8.

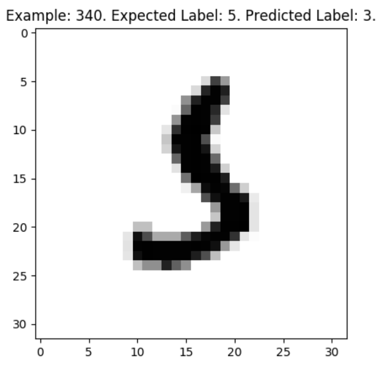

Fig. 4. Example of a misclassified test case, was expected 5 and was predicted as 3.

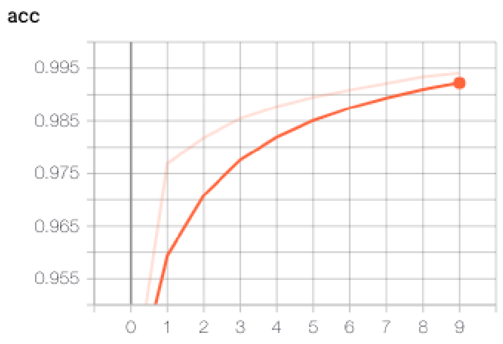

Fig. 5. Accuracy Graph generated by TensorBoard

IV. Conclusion

It was clear, therefore, that the Convolutional Neural Network worked as expected. The number prediction was very accurate having 98.7% of accuracy on the data set.

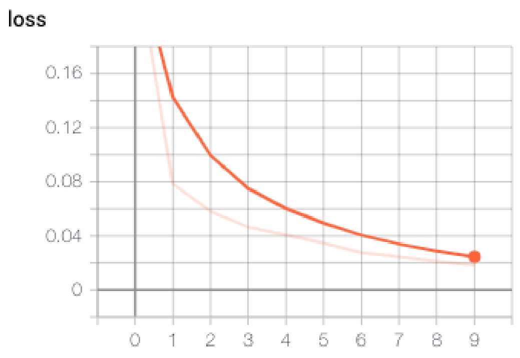

Fig. 6. Loss Function Graph generated by TensorBoard

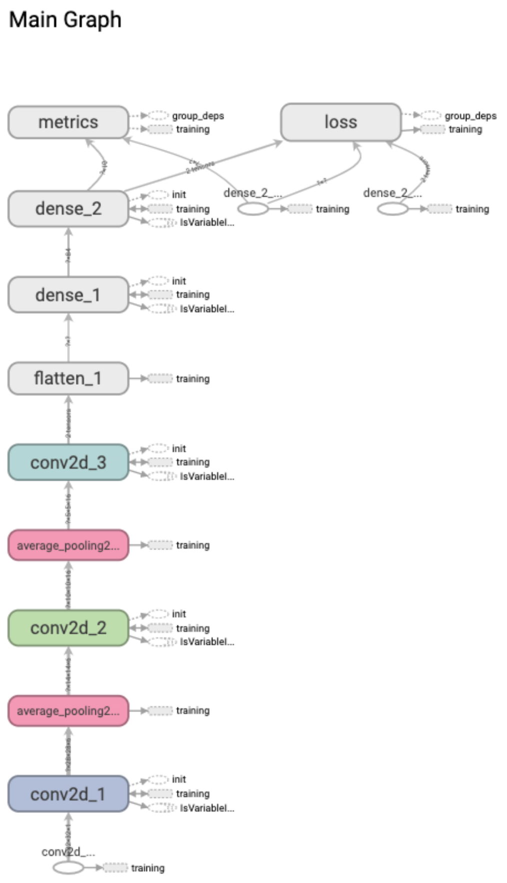

Fig. 7. TensorBoard Main Graph.

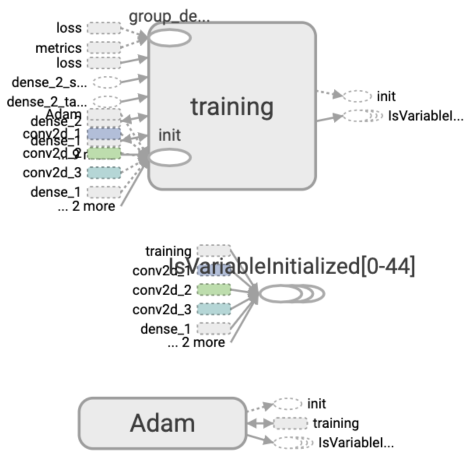

Fig. 8. TensorBoard Auxiliary Nodes.