Imitation Learning with Keras

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

***Abstract*—This paper evaluates two Keras' Neural Network by test it in different scenarios with two simple tests and an imitation learning problem.**

It was observed that the Neural Network worked fine for those purposes, and in some case, the result was really good, in the imitation learning case, for example.

***Index Terms*—Simple Evolution Strategy, SES, Covariance Matrix Adaptation Evolution Strategy, CMA-ES, optimization**

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 2017, Google's TensorFlow team decided to support Keras in TensorFlow's core library. Chollet explained that Keras was conceived to be an interface rather than a standalone machine-learning framework. It offers a higher-level, more intuitive set of abstractions that make it easy to develop deep learning models regardless of the computational backend used. Microsoft added a CNTK backend to Keras as well, available as of CNTK v2.0.

Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier. The code is hosted on GitHub, and community support forums include the GitHub issues page and a Slack channel.

In addition to standard neural networks, Keras has support for convolutional and recurrent neural networks. It supports other common utility layers like dropout, batch normalization, and pooling.

Keras allows users to productize deep models on smartphones (iOS and Android), on the web, or on the Java Virtual Machine. It also allows the use of distributed training of deep-learning models on clusters of Graphics Processing Units (GPU) and Tensor processing units (TPU).

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(
```

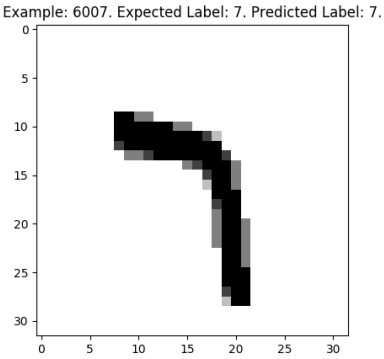

Fig. 1. Example of a correctly classified test case, was expected 7 and was predicted as 7.

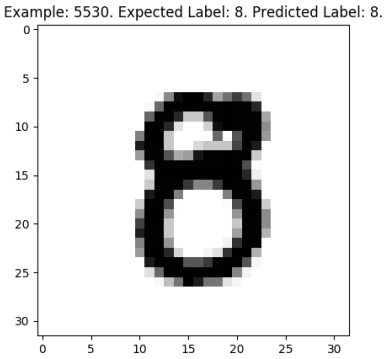

Fig. 2. Example of a misclassified test case, was expected 8 and was predicted as 8.

```
        filters=nf,
        kernel_size=(fx, fy),
        strides=(sx, sy),
        activation=activations.tanh,
        input_shape=(5, 5, 1)
    )
    )

    # 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*

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*

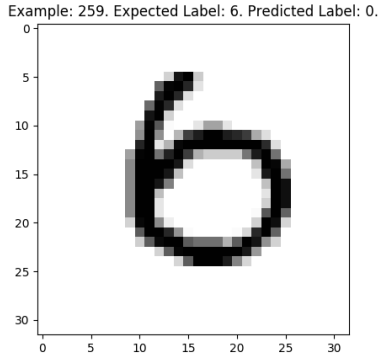

Fig. 3. Example of a misclassified test case, was expected 6 and was predicted as 0.

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

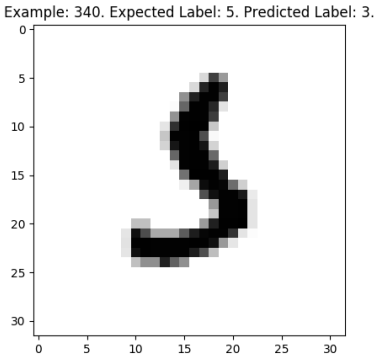

Fig. 4. Example of a misclassified test case, was expected 5 and was predicted as 3.

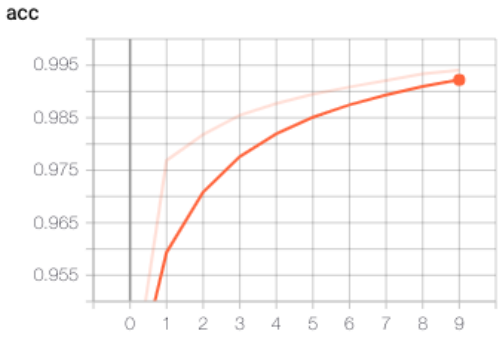

Fig. 5. Accuracy Graph generated by TensorBoard

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.

IV. Conclusion

It was clear, therefore, that the Keras' Neural Network worked as expected. Both test cases (greater than function and xor function) the Neural Network worked as well, with

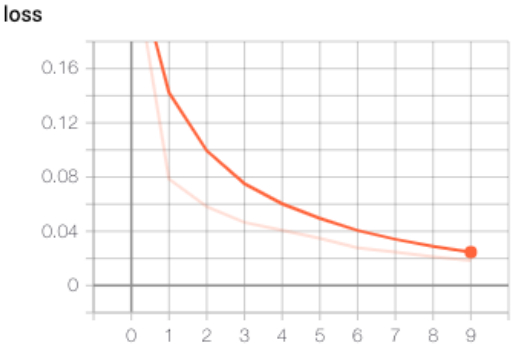

Fig. 6. Loss Function Graph generated by TensorBoard

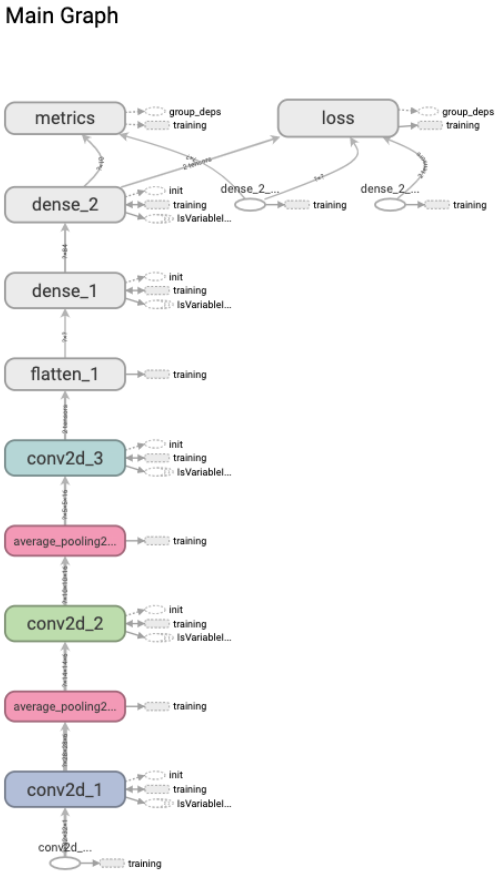

Fig. 7. TensorBoard Main Graph.

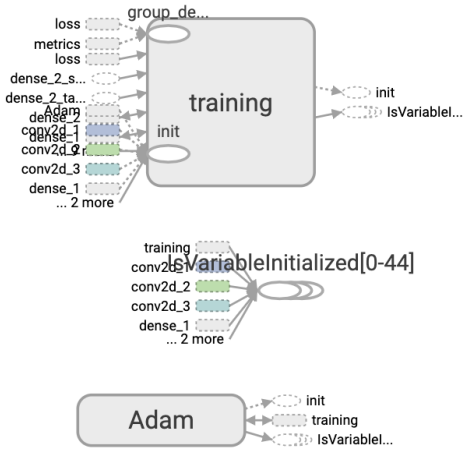

Fig. 8. TensorBoard Auxiliary Nodes.

a much better result with regularization, because without regularization the results were overfitted.

For the Imitation Learning with Keras, the results were good in some cases and very precise in most some cases.