PUC

Programming Final Project

Student: Guilherme Machado Goehringer
Advisor: Prof. Dr. Marcos Kalinowski

2020-1
Informatics Department

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900
RIO DE JANEIRO – BRASIL

Rio de Janeiro – June 13th, 2020

Sumário

1. Objective

The objective of this report, for the subject Programming Final Project, is to specify, design, develop, test and document a software product with the following requirements:

- The software product should be able to perform image classifications for a large amount of data;
- The images to be used in the classification task should be from common datasets like MNIST [1];
- The software product should give to the system's user the possibility to change basic parameters to fine tune the classification results;
- The software product should present key performance indicators so the user can evaluate the results.
- The software should be developed using modern programing language, and the respective libraries to accelerate development.

2. Introduction

Machine learning is a field of artificial intelligence and by its definition [2] is a science of programming computers with the goal to learn from data, being able then to makes data-driven predictions or choices rather than following firm static program instructions.

Machine learning involves three types of tasks:

- Supervised machine learning: a task of inferring (or learning) a function that maps an input to an output based on a pre-defined set of training examples with labeled training data.
- Unsupervised machine learning: a task to find patterns from data but without using a pre-existing labeled data set.
- Semi-supervised machine learning: an approach that combines the use of a small amount of labeled data with a large amount of unlabeled data during training. Semi-supervised learning falls between supervised and unsupervised learning.
- Reinforcement learning: is an area of machine learning where the learning system is an agent. This area studies how these agents should observe the environment, chose and perform actions, in order to get rewards.

A new area of supervised machine learning research is Deep learning. The most popular Deep Learning models are: deep neural networks (DNN), deep belief networks (DBN), recurrent neural networks (RNN) and convolutional neural networks (CNN), that have been applied to several fields.

This report, for the programming final project subject, has focus on CNNs, for the application of image classification, using a common dataset called MNIST (that contains 70,000 handwritten digits).

3. Background of Convolutional Neural Networks

Convolutional neural networks (CNNs) emerged from the study of the brain's visual cortex, and they have been used in image recognition since the 1980s. But due to the recent increase of computing power, and the availability of large datasets to be used for model training, CNNs have been adopted to perform complex visual tasks in several applications.

A CNN is composed by different layers (one input layer, one output layer and some hidden layers that lie in between), and the most important layer that characterizes a CNN is a convolution layer, that is a layer where neurons are not fully connected to every neuron to the preceding layer but only to the neurons located to a small area in the preceding layer called receptive filed (or convolutional kernel).

The below figure 1 [3] explains the main characteristics of a convolutional layer:

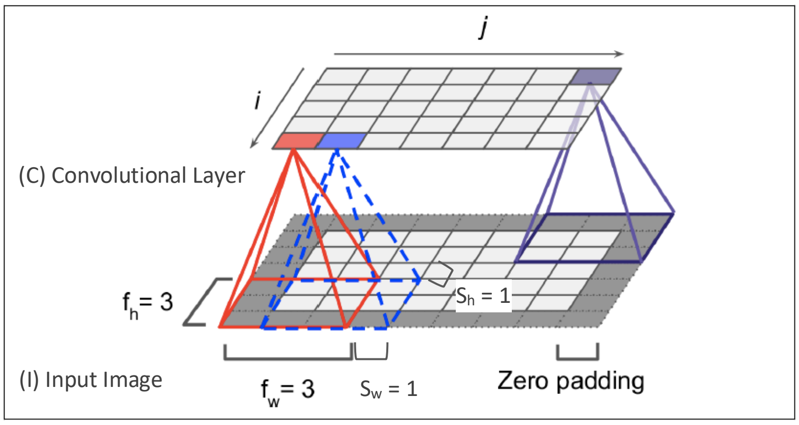


Figure 1: Convolutional layer, its connections and basic parameters

To explain the above figure 1:

- (I) Input Image: the input imagem that in this example has 5 pixels of height and 7 pixels of width.
- Receptive Field: the red or dotted blue rectangles on the input image.
- F_h and F_w: the dimensions of the receptive field, that in this example has 3 pixels of height and 3 pixels of width.
- S_h and S_w: the stride that represents the shift from one receptive field to the next, e.g., from the red receptive field to the dotted blue.
- Zero padding: in order to have the next layer after the input image with the same dimensions (in the case the architecture demands) of the input image it is needed to add zeros around the input image.
- (C) Convolutional Layer: the layer that it is connected to the input image through the receptive fields.
- Neurons on the convolutional layer: the red (or blue) squares on the convolutional layer that are the results of the convolution operation applied in the receptive field through filters.
- Filters (or convolutional kernel): it can be a black square with a vertical line in the middle, i.e., a matrix of 0s (zeros) with 1s (ones) in the middle. Neurons using this filter will ignore everything in the receptive field except for the central vertical line.
- Feature map: a layer full of neurons using the same filter outputs a feature map, which highlights the areas in an image that activate the filter the most.

4. Methodology

In order to perform the proposed software product for this work, the architecture chosen is a resnet-34 due to its ability to handle complex image classification tasks. The following methodology was applied:

- Load the MNIST dataset with handwritten digits (70,000 images with handwritten digits from 0 to 9);
- Evaluate the MNIST dataset to define the best split into training, validation and test;

- Split the MNIST dataset in to training (55,000 images of handwritten digits), validation (5,000) and test (10,000);
- Develop the resnet-34 convolutional neural network architecture model;
- Train the model using MNIST training examples;
- Validate the model using MNIST validation examples;
- Test the model using MNIST test examples;
- Test the model using digitized handwritten digits from the author of this report;
- Document the software product: use case diagram, architecture, source code and test (process, test cases, results).

5. Technology

The technology used considered tools that are free, open source, and with proved performance when handling machine learning tasks, like the one it is proposed on this report.

5.1 Python

It is an interpreted, high-level, general purpose programming language, with support to object-oriented programming, comparable to others languages like Java. Python has been highly used in the field of machine learning because it brings several useful libraries for the development of machine learning applications.

5.2 Anaconda

Anaconda is an open source distribution of Python programming language that simplifies package management and deployment.

5.3 Numpy

It is a Python library that adds support to handle large arrays and matrices, as well as the needed mathematical functions to operate these arrays and matrices. Since image classification is a task done basically through matrices' operations, Numpy turns into a must have library.

5.4 Scikit-learn

It is a Python library that brings a set of machine learning algorithms, separated into classification, regression, clustering, and is designed to interoperate with the Python numerical library NumPy. In this report we have only used to import the

MNIST dataset, since the neural network algorithms will come from keras-tensorflow platform.

5.5 Matplotlib

It is a Python plotting library that offers the Pyplot module, that is a module of Matplotlib which provides a MATLAB-like interface, with the exception that it is also free and open source.

5.6 Tensorflow

It is an end-to-end machine platform, especially helpful for applications using neural networks. It makes possible to train and run very large neural networks efficiently by distributing the computations across potentially hundreds of multi-GPU (graphics processing unit) servers.

5.7 Keras

It is a high-level Deep Learning API that makes it very simple to train and run neural networks. It can run on top of either TensorFlow, Theano, or Microsoft Cognitive Toolkit (formerly known as CNTK).

5.8 Jupyter Notebook

It is a Python execution environment.

6. Prototype

The software product (.ipynb) developed and this report (.pdf) are available at:

https://github.com/ggoehringer/PFP_GG

7. Documentation

This section presents and describes the documentation generated for this project, and included: a use case diagram, the architecture of the CNN implemented, and the quality control (process, test cases and results).

7.1 Use case diagram

Figure 2 presents the system use case diagram with the related scenario, actors involved and respective relationships. In this system there are two actors: th user and the system. Note: the user in this case is also a developer and/or a software engineering with some knowledge on deep learning.

- **The user:** this actor is responsible to interact with the system by entering needed inputs to the system and related model parameters (e.g., image input size).

- **The system:** this actor is responsible to split the dataset, train the model, predict the image classifications based on supervised learning, and report the results.

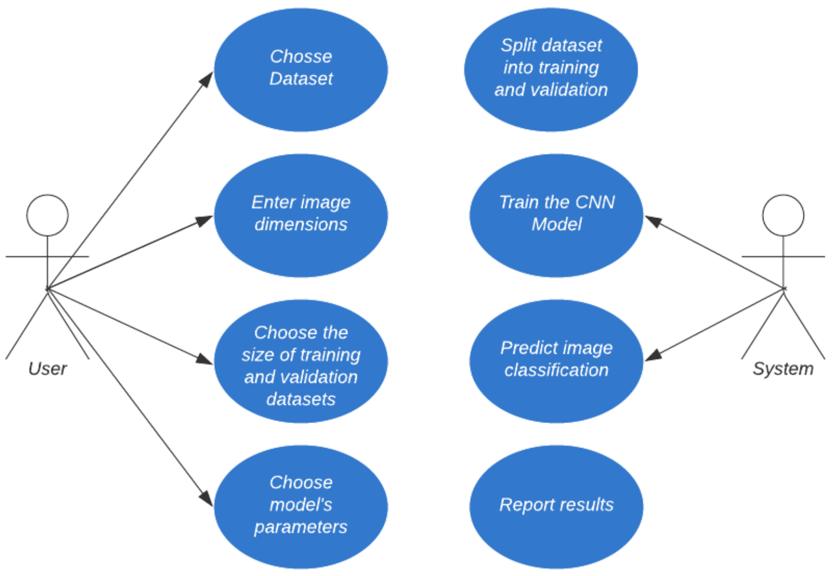

Chosse Dataset
Enter image dimensions
Choose the size of training and validation datasets
Choose model's parameters
Split dataset into training and validation
Train the CNN Model
Predict image classification
Report results
User
System

Figure 2: Use case diagram

The specification of the use case diagram for the two actors, namely user and system are described below on table 1 and 2:

Table 1: Use case specifications for the User actor

Name:	Choose Dataset
Actor:	User
Objective:	Choose, among the available options, the dataset to be used for the image classification task, e.g., MNIST, Fashion_MNIST, CIFAR-10, etc.
Main Flow:	1. On the keras.dataset click tab; 2. Select the dataset.

Name:	Enter image dimensions
Actor:	User
Objective:	Choose the image dimensions (height and width) according to the dataset chosen, as well as the number of channels, e.g., one channels for gray images and three for colored (RGB) images.
Main Flow:	1. On the first layer of the CNN, added by the script model.add (DefaultConv2D), add the input shape on input_shape = [height, width, channels], e.g., input_shape = [28, 28, 1]

Name:	Choose the size of training and validation datasets
Actor:	User
Objective:	From the chosen dataset, define the amount of data that will be used for training and validation of the model, as well as testing the model.
Main Flow:	1. On the sintaxe “X_train, X_valid = X_train_full[:-5000], X_train_full[-5000:]” choose how to split the dataset into training and validation. In this example, since MNIST dataset has 70,000 data, being 60,000 for training and validation and 10,000 for testing, we are choosing 55,000 for training and 5,000 for validation. 2. And on the sintaxe “y_train, y_valid = y_train_full[:-5000], y_train_full[-5000:]” choose the same size for the labels.

Name:	Choose model's parameters
Actor:	User
Objective:	Choose the basic parameters of the convolutional layers: receptive field, stride, and padding.
Main Flow:	1. Choose the receptive field size, by looking into the code for “kernel_size”, e.g., kernel_sizel = 3, that would mean a receptive field of 3 x 3. 2. Choose the stride, by looking into the code for “strides”, e.g., strides=2, that would mean the receptive fields shifting two pixels in the height direction and two pixels in the width direction. 3. Choose the padding, by looking into the code for “padding”, e.g., padding=”same” that would mean adding 0s (zeros) into the input image in order to keep the output image with the same dimensions or padding=”valid” in order to have different sizes between input and output layers, in this case not adding 0s (zeros).

Table 2: Use case specifications for the System actor

Name:	Split dataset into training and validation
Actor:	System
Objective:	Split the MNIST dataset into training and validation data.
Main Flow:	1. The system loads the dataset;

	2. The system split the dataset into training and validation, according to the size defined by the user.

Name:	Train the CNN Model
Actor:	System
Objective:	Train the model.
Main Flow:	1. The system runs model.compile; 2. The system runs model.fit.

Name:	Predict image classification
Actor:	System
Objective:	After training the model, the system can predict image classification from new input images.
Main Flow:	1. The user selects a new input image; 2. The system runs model.predict.

Name:	Report results
Actor:	System
Objective:	After training the model, the system can report an evaluation of the training using key performance indicators as accuracy and loss.
Main Flow:	1. The user runs model.evaluate; 2. The system reports the accuracy.

7.2 Architecture

A typical CNN architecture (see Figure 3) [3] is composed by some convolutional layers, intercalated with some pooling layers, and generally these layers are followed by activation function layers like RELU (Rectified Linear Unit).

The objective of the pooling layers is to subsample the input image aiming to reduce the computational effort. A pooling layer has also as basic parameters the receptive field, the stride and the padding, and its goal is to aggregate the inputs into an aggregate function, e.g., max or mean.

At the top of the CNN layers' stack a feedforward fully connected neural network is added, also followed by a RELU layer, and this final layer is responsible to present the predicted result.

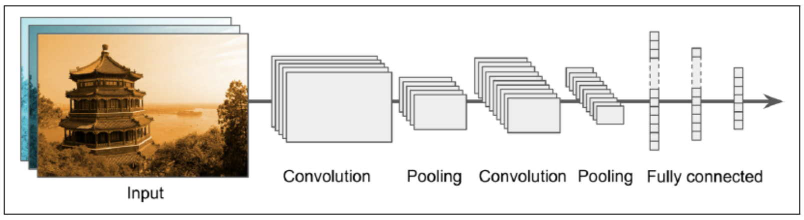
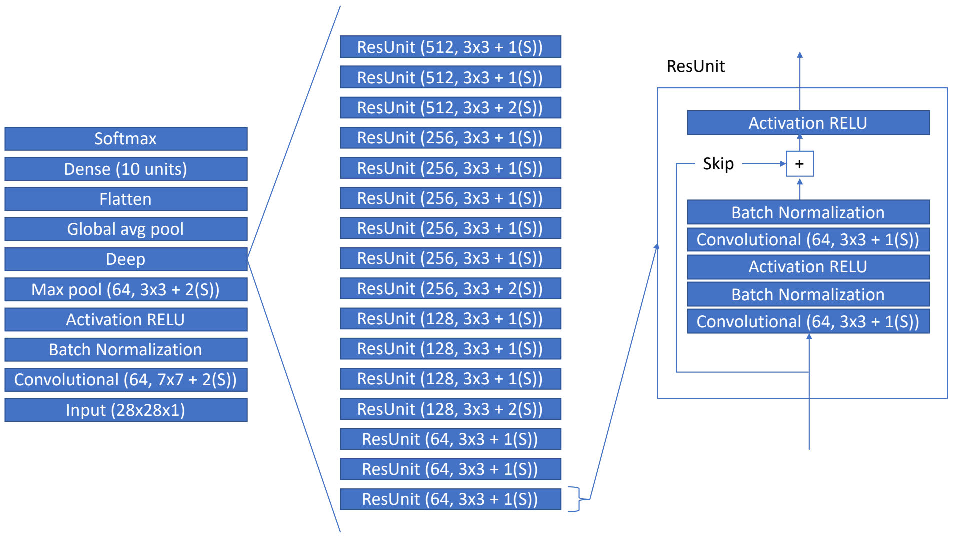


Figure 3: Typical CNN architecture

There are lots of different variants of the above CNN typical architecture, and some of the most famous architectures are: LeNet-5 architecture, AlexNet, GoogLeNet and the ResNet. In this report we have choosen to use the ResNet architecture with 34 layers.

7.2.1 ResNet-34

The following architecture was implemented for this report (figure 4):



Softmax
Dense (10 units)
Flatten
Global avg pool
Deep
Max pool (64, 3x3 + 2(S))
Activation RELU
Batch Normalization
Convolutional (64, 7x7 + 2(S))
Input (28x28x1)
ResUnit (512, 3x3 + 1(S))
ResUnit (512, 3x3 + 1(S))
ResUnit (512, 3x3 + 2(S))
ResUnit (256, 3x3 + 1(S))
ResUnit (256, 3x3 + 1(S))
ResUnit (256, 3x3 + 1(S))
ResUnit (256, 3x3 + 1(S))
ResUnit (256, 3x3 + 1(S))
ResUnit (256, 3x3 + 2(S))
ResUnit (128, 3x3 + 1(S))
ResUnit (128, 3x3 + 1(S))
ResUnit (128, 3x3 + 1(S))
ResUnit (128, 3x3 + 2(S))
ResUnit (64, 3x3 + 1(S))
ResUnit (64, 3x3 + 1(S))
ResUnit (64, 3x3 + 1(S))
ResUnit
Activation RELU
Skip
+
Batch Normalization
Convolutional (64, 3x3 + 1(S))
Activation RELU
Batch Normalization
Convolutional (64, 3x3 + 1(S))

Figure 4: Resnet-34 architecture

To explain each layer on this architecture:

- This architecture expects an image of 28x28 pixels, grayscale, as input;
- The 1st layer is a convolutional layer that outputs 64 feature maps, receptive field of 7x7 pixels, stride of 2 pixels and padding as same;
- The 1st layer is followed by a batch normalization (to address the issue of the vanishing/exploding gradients problems at the beginning of training, by simply zero-centering and normalizing each input). The batch normalization

can de added just before or after the activation function of each hidden layer and a RELU activation function layer;

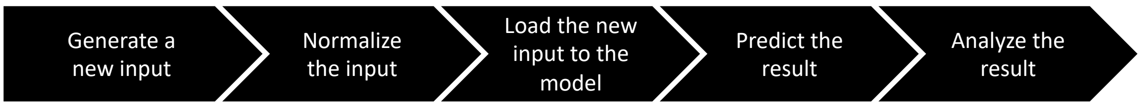
- Then we have a maxpool layer, with 64 feature maps, receptive field of 3x3, stride of 2 and same padding;
- After the maxpool we have 16 residual units (ResUnits or RUs). Each residual unit is composed of 2 convolutional layers (using 3×3 pixels for the receptive field, stride of 1 pixel and same padding), 2 batch normalization layers and 2 RELU activation layers. The stride of 1 pixel only changes to 2 pixels when the number of feature maps double from one ResUnit to another, e.g., 64 to 128. Additionally, the ResNet architecture brings an innovation called skip connections (or shortcut connections), that it is a technique that feeds the input signal of a given layer also to the output of this layer with the objective to speed up the learning process;
- After this deep stack of ResUnits, the network is composed by 1 global average pool, a layer to flatten the input to an 1D array (required to feed the next layer that is a fully connected layer), and at the top a dense fully connected layer with 10 units followed by a softmax activation function layer.

7.3 Quality Control

The quality control is made by adding new inputs to the model, after training, and by measuring the accuracy.

7.3.1 Test process and test cases

The test process consisted in the following steps:



- Write a digit in a piece of paper
- Take a picture
- Load the picture to the home project of Jupyter Notebook

- Resize the new input to 28x28 pixels
- Convert the new input to grayscale

- Load the resized and converted picture to the home project of Jupyter Notebook

- Run the model.predict using the new input

- Analyze the results through loss and accuracy indicators

7.3.2 Results

The following results were achieved by the model:

Phase	Dataset	Number of Images	Accuracy
Training & Validation	MNIST	55,000	98,59%
Test	MNIST	10,000	98,62%
Test	MNIST	10	100,00%
Test	Generated by the user	10	40%

The results using images generated by the end user reached 40% (4 digits were corrected classified out of the 10 digits), because probably the generated images carried noise that confused the predictor.

This is a clear demonstration that the software engineering community needs to increase its effort on designing architectures and testing techniques to increase the accuracy and robustness of these predictors based on neural networks.

The next step of the author of this report is to evaluate existing testing techniques, in order to propose a new technique that can help the community to increase the knowledge around deep learning testing and reduce the misclassifications after the network receives an adversarial example, e.g., a handwritten digit with noise.

8. Conclusion

In this report we showed the efficiency of a convolutional neural network, using a resnet-34 architecture, when handling image classification tasks with an accuracy higher then 98%.

But this high accuracy was reached only when using MNIST validation and testing data examples. When adding new inputs, generated by and end user for instance, the results reached only 40%.

There is a clear need to evolve architecture design and testing techniques to increase the correctness and the robustness of classifiers based on neural networks, reducing the misclassifications after the network receives an adversarial example.

This report brings the needed documentation for the user the manage the program and fine tune the model using basic parameters as the size of the receptive field, and stride and the padding.

Additionally, this report presents a quality control process, and also how to generate new inputs to the model, to test its robustness.

9. References

[1] https://www.openml.org/d/554.

[2] Mitchell, T.M., 1997. Machine learning. 1997. Burr Ridge, IL: McGraw Hill, 45(37), pp.870-877.

[3] Géron, A., 2019. Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems. O'Reilly Media.