The COSMIC Functional Size Measurement Method
Version 4.0.2

Sizing software in a Machine Learning context: A COSMIC Case study

Version 1.0

October 2019

Table of Contents

1

SIZING GENERIC SOFTWARE SUPPORTING MACHINE LEARNING

1.1 Introduction

COSMIC Function Points [1] quantify ('measure') the functionality of the requirements of software from many domains. This Case study presents an example of its applicability to measure the requirements of generic software that is a component of the Image Classifier software using a Machine Learning (ML) algorithm, as shown in Figure 2.1. It is based on [2].

This document segregates the functionality of the generic ('classical') software from the functionality specific to Machine Learning (ML) software [3]. This segregation will facilitate delegating tasks to staff with programming expertise, thereby freeing up time of ML data analysts. It also enables to collect data in a standardized fashion to develop estimation models for planning purposes and for on-going monitoring of the software tasks within a ML development project.

In ML, a neural network is a software application that can 'learn' to classify input data with the help of 'training examples' of that input data.

A. For example, a neural network can 'learn' to assign a handwritten digit the desired digit with – depending on the network – an accuracy of over 97%.
B. A training example is an example of an input together with its desired output.

There are many neural network architectures (and variants of these), and this Case study presents an example with a feedforward architecture (see Appendix A).

1.2 Objective of the software in this Case study

The generic software component of a feedforward neural network must be developed, the latter assigns and records the correct digit of images of separate individual handwritten digits, with a classification accuracy equal or above 98%. This software corresponds to an image classifier.

1.3 Context and facts

Starting points for the Case study are:

A. Images are available for training, each including a hand-written digit and its correct digit.
    - An image of a hand-written digit consists of 28x28 pixels.
    - The pixels are greyscale, with a value of 0.0 representing white, a value of 1.0 representing black, and in between values representing shades of grey.
    - All images are stored in a file.
B. Functional reuse of a pre-programmed feedforward network algorithm, including its Cost function. This feedforward algorithm software:
    1. assigns random values to the weights and biases on the basis of a mean and standard deviation;
    2. 'forward-propagates' the training images of a 'mini-batch' (a fixed number of training examples) and determines the cost (average deviation or 'error') of the actual and the desired output values of the training images in the mini-batch;

3. ‘backpropagates the changes to all weights and biases backwards through the layers in the network and stores the updated values; the algorithm determines the changes on the basis of the ‘cost’ in the output layer of the mini-batch just processed -;
4. processes all mini-batches of training examples, finishing an ‘epoch’ of training;
5. Repeats steps 2) to 4) for a specified number epochs of training.

C. The feedforward algorithm software stores the training parameters, so that it is possible to train anew with one or more changed parameters, the other parameters remaining the same.

D. The feedforward algorithm provides the storage of data it processed to meet the requirements of the generic software component of the Image Classifier software.

1.4 System requirements

This set of requirements is at the system level, it includes the data analyst requirements. The three graphs below are used to determine appropriate values of the hyper-parameters learning rate η, number of epochs and mini-batch size.

System Requirement 0: Images must be pre-processed.

Note. Since pre-processing is specific to each context, pre-processing is not documented here, its description and related measurement are not included within this case study. If such requirements were to be specified, the software functionality involved could be measured separately.

System Requirement 1: Initialize the feedforward network architecture

To initialize the feedforward network architecture, the data analyst must provide to the software application the following inputs: the number of layers, the number of units (‘neurons’) of each layer and, in the hidden layers, one weight per input and one bias.

The data analyst must be able to tune the network by varying these parameters of the network architecture.

System Requirement 2: Expand the number of images

The data analyst may ask the software to expand the number of images. The data analyst must then provide to the software ‘expansion instructions’. On the basis of an expansion instruction, the software must expand the number of images available for training by adding one distorted copy of each image.

Note. Expand instructions are not documented here, assume that the instructions will consist of a single data group.

System Requirement 3: Divide the images into three sub-sets

The software must divide the images into the three sub-sets of training, test and validation images, the members of which are randomly chosen. The data analyst inputs the sub-set names and the number of images of each sub-set. All images must be stored with their sub-set name.

System Requirement 4: Training step

After receiving from the data analyst the training parameters the software must start a training.

The data analyst provides to the software (a sub-set of) the training parameters: the mean and standard deviation for the weights and biases, sub-set name and the hyper parameters; in this case study the hyper-parameters are the learning rate η, the number of epochs and the mini-batch size.

During training the software calculates the classification accuracy of each epoch for each elapsed training time, and print these next to monitor the performance of learning.

System Requirement 5: Produce the graph 'Cost per epoch'

With help of this graph the data analyst can identify a suitable value of the learning rate η. After receiving the training parameters, among which the validation set, the neural network is executed and the software must display the graph 'Cost per epoch'.

The execution of the training is repeated with different learning rates provided by the data analyst.

The data analysts stops the training repetition when he identifies a suitable value of the learning rate η with the help of these graphs by comparing the rates of decrease of cost (i.e. average error between the desired and the actual output of training examples) – Figure 1.1.

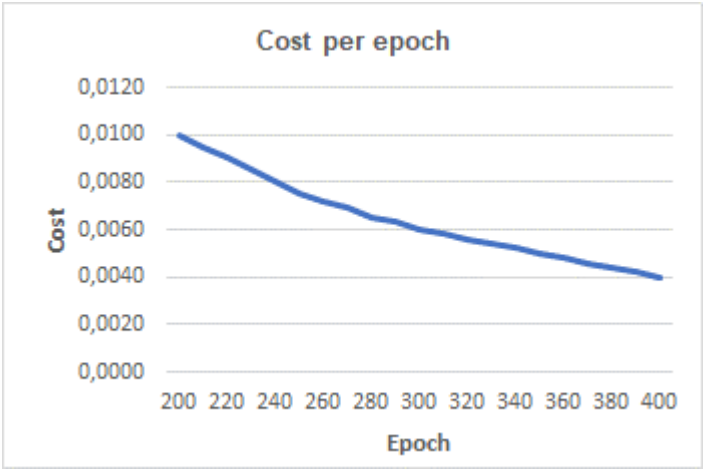

Figure 1.1 - Cost per epoch

System Requirement 6: Produce the graph 'Accuracy per epoch'

After a training step with the validation images, the software must produce the graph 'Accuracy per epoch'. The data analyst determines a suitable number of epochs for the training step with the help of this graph.

The data analyst will select the smallest number of epochs with which the required accuracy can be reached - Figure 1.2.

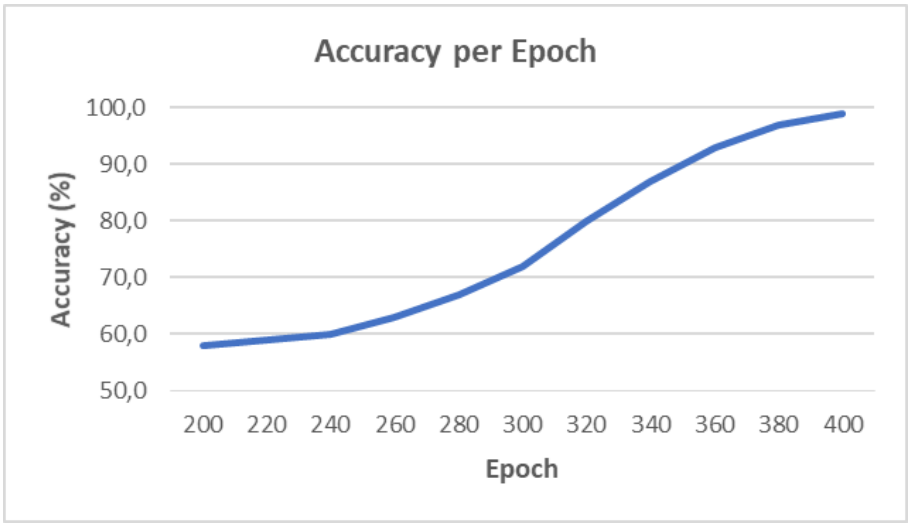

Figure 1.2 - Accuracy per epoch of training

System Requirement 7: Produce the graph 'Speed per mini-batch size'

The data analyst determines a suitable mini-batch size using this graph. The data analyst specifies the number of epochs and the number of intended mini-batch sizes, then the software must print the mini-batch size, the time and accuracy data on the graph in Figure 3:

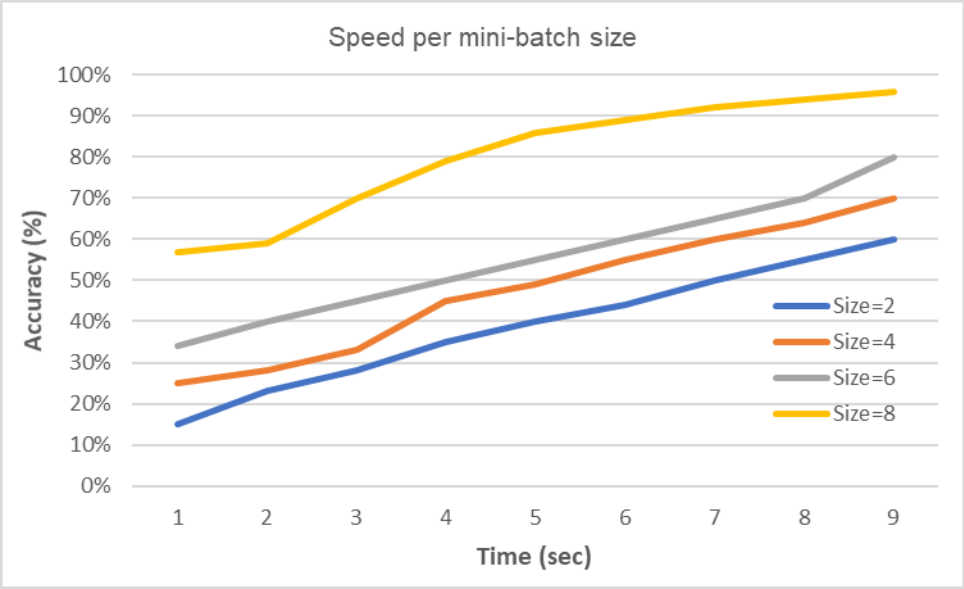

Figure 1.3 - Speed per mini-batch size

System Requirement 8: Error and confirmation messages

Each output must inform that an error has or has not occurred (but does not provide details - , in this case study the requirements for finding the errors are not specified).

System Requirement 9: Display of the x-axis and y-axis

The software must display a number of multiples on the x-axis and y-axis of graphs.

1.5 Software Requirements

The list of requirements of the generic software that follows includes the main data group names and corresponding attributes.

Software Requirement 1: Initialize the feedforward network architecture

The software receives a sequence of numbers to initialize the feedforward neural network architecture. The *length* of the sequence indicates the number of layers, each *number* in this sequence indicates the number of units of its layer, and each unit in the hidden layers has one weight per input and one bias.

Input. Data group for initializing the feedforward architecture: Number of units.

Result: Initialized feedforward architecture.

Software Requirement 2: Expand the number of images

The software receives the expansion instruction, then

1. copies each image,
2. changes it, and
3. adds the result to the images.

Input. Data group Expansion instruction (data attributes: not specified in this case study)

Output: Changed images, added as additional new images

Note. In the absence of details on 'expansion instruction' an assumption is made that this will consist of a single data group. If in other cases there is more in the 'expansion instructions', including more than one data group, this could then lead to additional data movements since there would be more data groups,

Software Requirement 3: Divide the images into three sub-sets

The images must be divided into the three sub-sets of 'training images', 'test images' and 'validation image'. The software receives the numbers of their members. The members of the sets must be randomly chosen. All images must be stored with their sub-set name.

Input. Data group Sub-set of images (attributes: sub-set name, number of images).

Output: The images with the sub-set names.

Software Requirement 4: Training step

The software receives the training parameters to start a training. The training parameters are mean and standard deviation, the name of the sub-set to be used and the hyper parameters (learning rate η, the number of epochs and the mini-batch size), or a sub-set of these training parameters. During training the classification accuracy per epoch and elapsed training time must be printed.

Input: Data group of training parameters

Results: A trained network and the data needed for the graphs stored. Data group: epoch ID, number of correctly classified images, training time.

Software Requirement 5: Produce the graph 'Cost per epoch' – Figure 1.1

The software receives the training parameters, on basis of which the graph 'Cost per epoch' must be produced.

Input. Data group of desired training parameters

Output: Data groups Epoch range (x-axis), Cost range (y-axis) and Cost per epoch

Software Requirement 6: Produce the graph 'Accuracy per epoch' – Figure 1.2

The software receives from the data analyst the first and the last epoch ID (epoch number) to be shown after a training step (during which the required data has been stored) and produces the graph.

Input: Data group epoch ID

Output: Data groups Epoch range (x-axis), Accuracy range (y-axis) and Accuracy per epoch

Software Requirement 7: Produce the graph 'Speed per mini-batch size' – Figure 1.3

The software receives from the data analyst a number of epochs and a number of intended mini-batch sizes and produces the graph 'Speed per mini-batch size'.

Input: Data groups the number of epochs and the number of intended mini-batch sizes

Output: Data groups Time (x-axis), Accuracy range (y-axis), Accuracy of mini-batch at point of time and mini-batch sizes legend for the graph.

The system requirements 8 and 9 of the error and confirmation messages and on the display of the x-axis and y-axis are taken into account in the measurement section 3.2.

1.6 The data model of the Image Classifier software

The entities and their attributes are as follows:

Feedforward architecture. Feedforward architecture ID, sequence of numbers (the length of which indicates the number of layers and each number of the sequence indicates the number of units of its layer)

Training step. Training step ID, mean, standard deviation, sub-set name, number of epochs, number of images per mini-batch, learning rate (η), values of the weights and biases.

Epoch. Epoch ID (= epoch number), cost, accuracy, elapsed time

Figure 1.5

2

MEASUREMENT STRATEGY

2.1 Measurement purpose

The purpose of the measurement *for this Case study* is to determine the functional size of the generic software, on the basis of its software functional specifications (i.e. excluding the specifications to develop the algorithms themselves that are being reused). In practice, the purpose would usually be to estimate the effort of implementing the network's generic software.

2.2 Measurement scope

The scope of the measurement consists of the software requirements of section 1.4.

2.3 Functional users of the generic software

The functional users are

- the data analysts of the neural network, i.e. those who tune the network with help of the generic software so as to meet the accuracy requirement.
- The reused feedforward neural network algorithm.

In this example, the reused software doesn't have to be measured: therefore, it is considered a functional user of the generic software component.

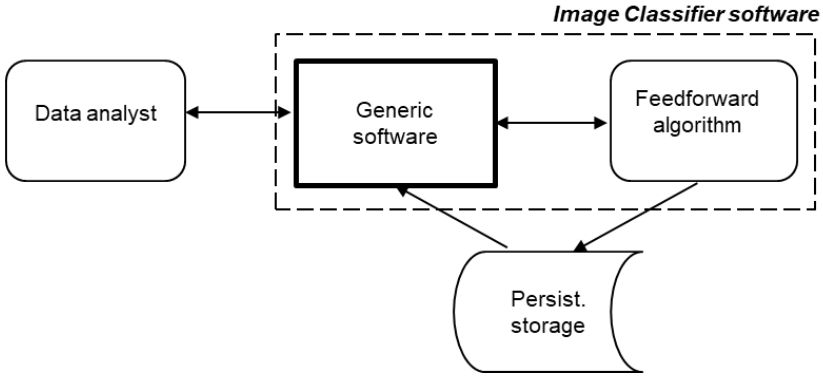

Figure 2.1 - Context diagram of the generic software

2.4 Other measurement strategy parameters

The functional users that initiate services of the generic software are individual data analysts of the neural network and hence the functional processes are individual functional processes.

3

THE MAPPING AND MEASUREMENT PHASES

3.1 The functional processes

Functional processes are the unique elementary parts of the requirements initiated by a functional user. In the requirements the following functional processes can be identified.

FP 1: Initialize of the feedforward network architecture

The software receives from the data analyst a sequence of numbers to initialize the desired neural network architecture.

FP 2: Expand the number of images

The software receives from the data analyst an expansion instruction to initiate expanding the images.

FP 3: Divide the images into three sub-sets

The software received from the data analyst the sub-set names and their number of images to initiate the division of the images into the sub-sets.

FP 4: Training step

The software receives from the data analyst the training parameters to initiate a training step.

FP 5: Produce the graph 'Cost per epoch'

The software receives from the data analyst a set of training parameters and produces the graph 'Cost per epoch' of the training step,

FP 6: Produce the graph 'Accuracy per epoch'

The software receives from the data analyst the first and the last epoch ID to produce the graph 'Accuracy per epoch' of the training step.

FP 7: Produce the graph 'Speed per mini-batch size'

The software receives from the data analyst the number of epochs and the number of mini-batch sizes to produce the graph 'Accuracy per epoch' of the training step.

3.2 Measurement of the feedforward neural network

The objects of interest of the data movements are in italics between [] after the corresponding data groups. Note that fixed text does not require movement of data.

FP 1: Initialize a feedforward network architecture

DM	Data Group/ Data attributes
Entry	Number of units [*Feedforward architecture*]
Exit	Number of units [*Feedforward architecture*], to feedforward algorithm
Entry	Error/Confirmation data [*Feedforward architecture*] from feedforward algorithm
Exit	Error/Confirmation message [*Error/Confirmation*]

Size: 4 CFP

FP 2: Expand the images

DM	Data Group/ Data attributes
Entry	Expansion instruction [*Expansion*]
Read	Image data [*Image*]
Write	Add distorted image data [*Image*]
Exit	Error/Confirmation message [*Error/Confirmation*]

Size: 4 CFP

FP 3: Divide the images into three sub-sets

DM	Data Group/ Data attributes
Entry	Sub-set of images (sub-set name, number of images [*Sub-set of images*]
Read	Image data [*Image*]
Write	Image data with sub-set name [*Image*]
Exit	Error/Confirmation message [*Error/Confirmation*]

Size: 4 CFP

FP 4: Train the network

DM	Data Group/ Data attributes
Entry	Training parameters (mean, standard deviation, number of epochs, number of images per mini-batch, learning rate (η), sub-set name) [*Training step*]
Exit	Training parameters [*Training step*] to feedforward algorithm
Entry	Epoch ID, number of correctly classified images, total number of images, elapsed time [*Epoch*] from feedforward algorithm
Exit	Print epoch ID, accuracy, elapsed time [*Epoch*]
Exit	Error/Confirmation message [*Error/Confirmation*]

Size: 5 CFP

FP 5: Display Cost per epoch (graph – Figure 1.1)

DM	Data Group/ Data attributes
Entry	Epoch ID [*Epoch range*] Input lowest and highest ID values
Read	Epoch ID, Cost [*Epoch*] stored by the feedforward algorithm
Exit	Epoch ID (x-axis, appropriate multiples) [*Epoch range*]
Exit	Cost (y-axis, appropriate multiples) [*Epoch Cost range*]
Exit	Epoch Cost [*Epoch*]
Exit	Error/Confirmation message [*Error/Confirmation*]

Size: 6 CFP

FP 6: Display classification accuracy per epoch (graph – Figure 1.2)

DM	Data Group/ Data attributes
Entry	Epoch ID [*Epoch range*] Input lowest and highest ID values
Read	Epoch ID, Epoch accuracy [*Epoch range*] stored by the feedforward algorithm
Exit	Epoch ID (x-axis, appropriate multiples) [*Epoch range*]
Exit	Epoch accuracy (y-axis, appropriate multiples) [*Epoch accuracy range*]
Exit	Epoch ID, Epoch accuracy [*Epoch*]
Exit	Error/Confirmation message [*Error/Confirmation*]

Size: 6 CFP

FP 7: Determine mini-batch size (graph – Figure 1.3)

DM	Data Group/ Data attributes
Entry	Training parameters (without number of images per mini-batch) [*Training step*]
Exit	Training parameters (without number of images per mini-batch) [*Training step*] to feedforward algorithm
Entry	Mini-batch size [*Mini-batch sizing*] Input the sizes to be compared
Exit	Mini-batch size [*Mini-batch sizing*] Sizes to be compared to feedforward algorithm
Read	Elapsed time, Epoch accuracy of mini-batch size [*Epoch*] stored by the feedforward algorithm
Exit	Elapsed time (x-axis, in seconds) [*Epoch time range*]
Exit	Epoch accuracy (y-axis: appropriate multiples) [*Epoch accuracy range*]
Exit	Mini-badge legend [*Mini-badge*] from Entry above
Exit	Epoch accuracy of last epoch at each second per mini-batch size [*Epoch accuracy per size*
Exit	Error/Confirmation message [*Error/Confirmation*]

Size: 10 CFP

The total functional size of the application is the sum of the sizes of its functional processes FP1 to FP7, that is:

4 CFP + 4 CFP + 4 CFP + 5 CFP + 6 CFP + 6 CFP + 10 CFP = **39 CFP**

REFERENCES

All COSMIC publications are available for free download from the Knowledge Base of www.cosmic-sizing.org.

[1] Measurement Manual

[2] Neural networks and deep learning, M. Nielsen, Determination Press, 2015 (available at neuralnetworksanddeeplearning.com)

[3] Arlan Lesterhuis, Alain Abran, 'COSMIC Sizing of Machine Learning Image Classifier Software Using Neural Networks', 29th IWSM-MENSURA conference, Oct. 7-9, 2019, Haarlem, The Netherlands. Proceedings are online at http://ceur-ws.org/Vol-2476.

[4] Deep learning neural networks, D. Graupe, World Scientific, 2016

APPENDIX A – ARCHITECTURE, MACHINE LEARNING

A.1 The feedforward neural network architecture

The feedforward neural network consists of units (‘neurons’) organized in a number of layers:

- input layer: consists of units that encode the values of the input
- output layer: consists of units that indicate the outcome of the classification
- hidden layers in between

A unit in a layer receives input from all the units in the previous layer, multiplies each input with a ‘weight’ and add a ‘bias’ to the result.

Next an ‘activation’ (‘transfer’) function is applied to this result and the outcome is sent to each unit in the next layer.

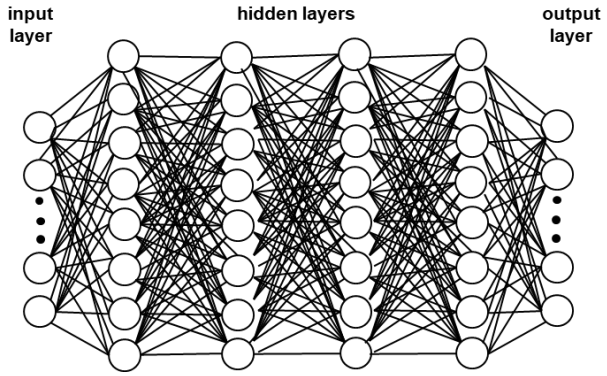

Figure A.1 – Architecture of a feedforward neural network

A.2 Machine learning

During training, a so-called cost function C of the neural network quantifies the average over the error of all individual training examples in a mini-batch. An example of a cost function is $C(w,b) = (1/2n).\sum_x(y(x)-a(x))^2$, where:

- n indicates the number of training examples in a mini-batch,
- x is a training example,
- y(x) its desired output,
- a(x) its actual output
- w indicates the weights in the hidden layers , and
- b the biases in the hidden layers.

During training each training example is input together with its desired value, the latter being used to determine the error between desired and actual output.

With the so-called backpropagation algorithm it appears possible to reduce the ‘average error’ C(w,b) by systematically and repeatedly adapting the weights and biases so that the output a(x) from the network approximates y(x) for all training inputs x: the neural network ‘learns’. The backpropagation algorithm is ‘the workhorse of learning in neural networks’ [2].

The purpose of the backpropagating algorithm is to find a global minimum of the cost function C, or at least a minimum of which the error is acceptable for the purpose of the application.

In practice it is useful to experiment with the sizes of the changes of the weights and biases supplied by the backpropagation algorithm. The sizes of the changes will then be multiplied with a positive factor η (eta), the 'learning rate'.

'Overfitting' is ineffective training, e.g., when the properties of the training set have been learned, rather than the software really has learned to classify. This parameter causes weights to be small so that incidental properties in the input will not easily be learned.

The classification accuracy can be improved by using more training examples. One way of expanding the training data is to add artificial training examples by applying an operation on the present training examples that reflect real-world variation and add the result as new training examples.

A.3 The training, the testing and the validation sets

Learning can take place on basis of 3 sub-sets of the training images, called the training set, the test set and the validation set. The training set is used for training the network, the test set for testing the result of training. The validation set is to be used to determine the values of the 'hyper parameters' learning rate (indicated by η), the size of the mini-batches to be used, and the number of epochs of training.

APPENDIX B - GLOSSARY OF TERMS

This Glossary contains terms that are specific to this Guideline. The Measurement Manual [1] contains the main Glossary of terms of the COSMIC method, In the definitions given below:

- terms are shown in **bold**.
- terms that are defined elsewhere in this Glossary are under-lined, for ease of cross-reference.

Bias. A measure of how easy it is to get a unit to output.

Classification accuracy. Percentage of correctly classified input items.

Cost. Average error between the desired and the actual output of training examples.

Deep neural network. A neural network consisting of more than one hidden layer

Epoch. A single pass through the full training set.

Hidden layer. A layer that is not an input layer or output layer.

Hyper-parameter. Special training parameter, e.g. the learning rate η, the number of epochs, the mini-batch size.

Input layer. The layer that encodes the values of the input.

Learning rate. A positive number (indicated by η) to be multiplied with the sizes of all the changes of weights and biases during learning.

Mini-batch. A fixed number of training examples used to update learning on the basis of the average cost (error) of the training examples in the mini-batch.

Neural network. A set of connected units, grouped in an input layer, an output layer and one or more hidden layers.

Neuron. A part of a neural network containing one weight per input and one bias.

Output layer. The layer that indicates the outcome of the classification.

Overfitting. Ineffective training (e.g. training that learns the properties of input data, rather than really improving classification).

Overtraining. Synonym of overfitting.

Test example. An example used to verify the accuracy of learning.

Training example. An example to learn from.

Training parameter. A parameter that determines the training; in this Case these are the mean and standard deviation of the weights and biases, the sub-set name and the hyper parameters.

Unit. Synonym of neuron.

Validation example. An example used to determine hyper-parameters of the neural network

Weight. A real number expressing the importance of the respective input to the output.

APPENDIX C - ESTIMATING WITH SOFTWARE SIZE

The main use of functional size is to provide an estimate of the effort of implementing an application on basis of its specification.

For estimating on the basis of functional size, both the size and the effort (i.e. the related number of work hours) of a number of applications must be captured. With help of a simple regression analysis ('Ordinary Least Squares'), the relationship between sizes and work hours can now be obtained and made visible in Excel. The relationship between sizes and work hours can be expressed by e.g. a linear function (see Figure B.1). This function can be used for

- estimation of effort of future implementations
- to serve as a second opinion to an estimate given by a project manager or a contractor.

With a new application size and the work hours of implementations the company data repository can be updated for estimations of future implementations.

As an example, suppose that the specifications of a number of applications below have been measured and realized, with the indicated implementation effort:

Application	Size (CFP)	Effort (hours)
Application1	50	190
Application2	90	355
Application3	45	165
Application4	60	315
Application5	85	370
Application6	120	365
Application7	30	195
Application8	65	295

These data result in the graph of Figure B.1, with the relationship between size and effort indicated by the trend line. The Figure also displays the formula of the trend line and the 'determination coefficient' R^2. The determination coefficient indicates the preciseness of the model, of which the maximum is 1. The closer to 1, the better the line fits the points.

As the formula is a 'best fit', it makes no sense to use all decimals of the indicated formula, it produces 'order of magnitude' estimates of effort needed to implement an application, given its sizes in CFP. Therefore the formula can be rounded, meaning that efforts can be estimated with help of the simple formula.

Effort (hours) = 2,5 * Size (CFP) + 109

As an example, with a measured size of a specification of 100 CFP, the estimated effort would become 2,5 * 100 + 109 = 359 hours.

Note that it is vital to realize that:

- The hours of all applications need to be collected for the same set of implementation activities. Each estimate is the estimate of the hours to perform *that* set of implementation activities.
- The formula is calibrated for the organization where these numbers were collected. An organization with a different development environment may get different efforts.
- The formula is calibrated for the available size range, i.e. don't use it for sizes far outside this range.
- The formula will change when new data are added.

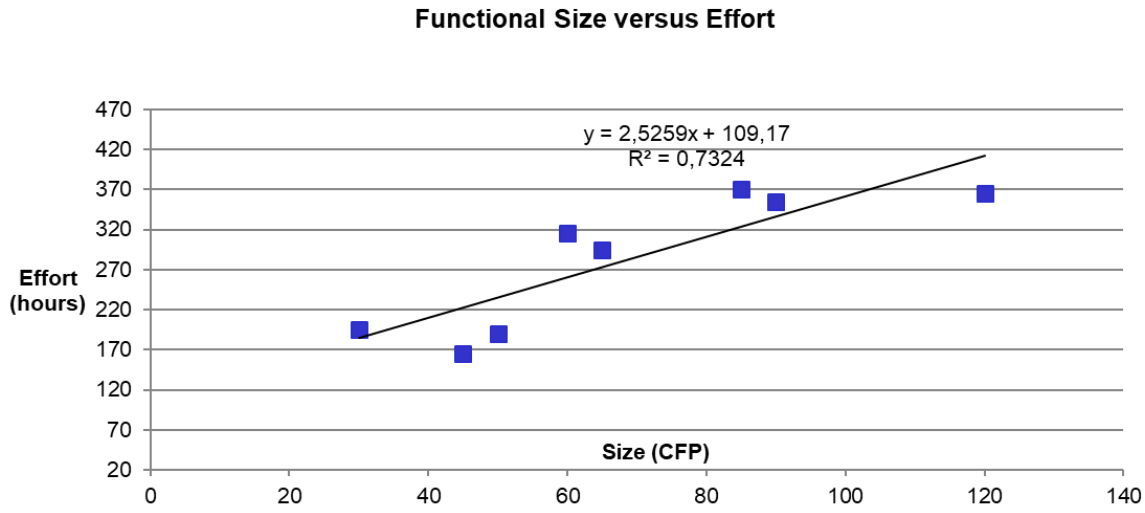

Figure C.1 – The estimation model

APPENDIX D – GENERAL INFORMATION

D.1 Acknowledgements

Version 1.0 authors and reviewers 2018 (alphabetical order)		
Alain Abran* École de Technologie Supérieure, Université du Québec, Canada	Arlan Lesterhuis* MPC, The Netherlands	Bruce Reynolds, Tecolote Research, USA

* Author(s) of this Guideline

D.2 Version control

The following table gives the history of the versions of this document.

DATE	REVIEWER(S)	Modifications / Additions
15-10-2019	COSMIC Measurement Practices Committee	First version 1.0 issued

D.3 Change requests, comments, questions

Where the reader believes there is a defect in the text, a need for clarification, or that some text needs enhancing, please send an email to: mpc-chair@cosmic-sizing.org

You can use the forum on cosmic-sizing.org/forums to post your questions and receive answers from our world-wide community. The quality of any answers will depend on the knowledge and experience of the community member that writes the answer; the MPC cannot guarantee the correctness. Commercial organizations exist that can provide training and consultancy or tool support for the method. Please consult the www.cosmic-sizing.org web-site for further detail.