Module Interface Specification for ANN (Artificial Neural Network)

Tanya Djavehrpour

April 13, 2024

1 Revision History

Date	Version	Notes
Mar. 19, 2024	1.0	Initial Draft
Apr. 12, 2024	1.1	Modification According to Implementation
Apr. 13, 2024	1.2	Modification According to Reviewers' Feedbacks

2 Symbols, Abbreviations and Acronyms

See SRS Documentation Djavaherpour (2024b) at HERE.

Contents

1 Revision History **i**

2 Symbols, Abbreviations and Acronyms **ii**

3 Introduction **1**

4 Notation **1**

5 Module Decomposition **2**

6 MIS of ANN Control Module **4**
6.1 Module . . . 4
6.2 Uses . . . 4
6.3 Syntax . . . 4
6.3.1 Exported Constants . . . 4
6.3.2 Exported Access Programs . . . 4
6.4 Semantics . . . 4
6.4.1 State Variables . . . 4
6.4.2 Environment Variables . . . 4
6.4.3 Assumptions . . . 4
6.4.4 Access Routine Semantics . . . 5
6.4.5 Local Functions . . . 5

7 MIS of Saved ANN Model Module **6**
7.1 Module . . . 6
7.2 Uses . . . 6
7.3 Syntax . . . 6
7.3.1 Exported Constants . . . 6
7.3.2 Exported Access Programs . . . 6
7.4 Semantics . . . 6
7.4.1 State Variables . . . 6
7.4.2 Environment Variables . . . 6
7.4.3 Assumptions . . . 6
7.4.4 Access Routine Semantics . . . 6
7.4.5 Local Functions . . . 7

8 MIS of Output Module **8**
8.1 Module . . . 8
8.2 Uses . . . 8
8.3 Syntax . . . 8
8.3.1 Exported Constants . . . 8
8.3.2 Exported Access Programs . . . 8

8.4 Semantics . . . 8
8.4.1 State Variables . . . 8
8.4.2 Environment Variables . . . 8
8.4.3 Assumptions . . . 8
8.4.4 Access Routine Semantics . . . 8
8.4.5 Local Functions . . . 9

9 MIS of Input Classifier Module **10**
9.1 Module . . . 10
9.2 Uses . . . 10
9.3 Syntax . . . 10
9.3.1 Exported Constants . . . 10
9.3.2 Exported Access Programs . . . 10
9.4 Semantics . . . 10
9.4.1 State Variables . . . 10
9.4.2 Environment Variables . . . 10
9.4.3 Assumptions . . . 10
9.4.4 Access Routine Semantics . . . 11

10 MIS of Input Image Module **12**
10.1 Module . . . 12
10.2 Uses . . . 12
10.3 Syntax . . . 12
10.3.1 Exported Constants . . . 12
10.3.2 Exported Access Programs . . . 12
10.4 Semantics . . . 12
10.4.1 State Variables . . . 12
10.4.2 Environment Variables . . . 12
10.4.3 Assumptions . . . 12
10.4.4 Access Routine Semantics . . . 13
10.4.5 Local Functions . . . 13

11 MIS of Training Model Module **14**
11.1 Module . . . 14
11.2 Uses . . . 14
11.3 Syntax . . . 14
11.3.1 Exported Constants . . . 14
11.3.2 Exported Access Programs . . . 14
11.4 Semantics . . . 14
11.4.1 State Variables . . . 14
11.4.2 Environment Variables . . . 14
11.4.3 Assumptions . . . 14
11.4.4 Access Routine Semantics . . . 14

11.4.5 Local Functions . . . 15

12 Input Preparing and Preprocessing Module **16**
12.1 Module . . . 16
12.2 Uses . . . 16
12.3 Syntax . . . 16
12.3.1 Exported Constants . . . 16
12.3.2 Exported Access Programs . . . 16
12.4 Semantics . . . 16
12.4.1 State Variables . . . 16
12.4.2 Environment Variables . . . 16
12.4.3 Assumptions . . . 16
12.4.4 Access Routine Semantics . . . 16
12.4.5 Local Functions . . . 17

13 Data Preparing and Preprocessing Module **18**
13.1 Module . . . 18
13.2 Uses . . . 18
13.3 Syntax . . . 18
13.3.1 Exported Constants . . . 18
13.3.2 Exported Access Programs . . . 18
13.4 Semantics . . . 18
13.4.1 State Variables . . . 18
13.4.2 Environment Variables . . . 18
13.4.3 Assumptions . . . 19
13.4.4 Access Routine Semantics . . . 19
13.4.5 Local Functions . . . 19

14 MIS of Training and Testing Module **21**
14.1 Module . . . 21
14.2 Uses . . . 21
14.3 Syntax . . . 21
14.3.1 Exported Constants . . . 21
14.3.2 Exported Access Programs . . . 21
14.4 Semantics . . . 21
14.4.1 State Variables . . . 21
14.4.2 Environment Variables . . . 21
14.4.3 Assumptions . . . 22
14.4.4 Access Routine Semantics . . . 22
14.4.5 Local Functions . . . 23

3 Introduction

The following document details the Module Interface Specifications for ANN (Artificial Neural Network). This document specifies how every module is interfacing with every other parts.

Complementary documents include the System Requirement Specifications (SRS) Djavaherpour (2024b) and Module Guide (MG) Djavaherpour (2024a). The full documentation and implementation can be found at Github repository for ANN.

4 Notation

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | ... | c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by ANN.

Data Type	Notation	Description
1D array	$\mathbf{A}_i$	A linear sequence of elements
2D matrix	$\mathbf{M}_{ij}$	A collection of elements arranged in rows and columns
3D matrix	$\mathbf{M}_{ijk}$	A structure composed of elements arranged in a grid with three dimensions
boolean	*bool*	True or False
dictionary	*dict*	A dictionary to store data as keys and values
string	*str*	A sequence of characters
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
positive Integer	$\mathbf{Z}_+$	a number without a fractional component in (0, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)

The specification of ANN uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of

characters. Tuples contain a list of values, potentially of different types. In addition, ANN uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

5 Module Decomposition

The following table is taken directly from the Module Guide Djavaherpour (2024a) document for this project.

Level 1	**Level 2**
Hardware-Hiding	
Behaviour-Hiding	ANN Control Module Saved ANN Model Module Output Module Input Classifier Module Input Image Module Training Model Module
Software Decision	Input Preparing and Preprocessing Module Data Preparing and Preprocessing Module Training and Testing Module

Table 1: Module Hierarchy

6 MIS of ANN Control Module

6.1 Module

`main`

6.2 Uses

- Hardware-Hiding Module
- Saved ANN Model Module (7)
- Output Module (8)

6.3 Syntax

6.3.1 Exported Constants

None.

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
`main`	-	-	-

6.4 Semantics

6.4.1 State Variables

None.

6.4.2 Environment Variables

None.

6.4.3 Assumptions

- The ANN Control Module assumes that the Hardware-Hiding Module, Saved ANN Model Module, and Output Module are implemented according to their specifications. However, it does include error handling to manage unexpected behaviors or failures in these modules.
- The system environment (operating system, hardware) is assumed to be stable. Also, essential libraries and dependencies are presumed to be correctly installed and configured.

6.4.4 Access Routine Semantics

main():

- transition: Initializes the program.

Note: As the ANN Control Module mainly serves as a coordinator between different modules without maintaining its own state or producing output, its primary function is to ensure the correct sequence of operations and interactions between these modules. It relies on the robustness of the called modules' error handling.

6.4.5 Local Functions

None.

7 MIS of Saved ANN Model Module

7.1 Module

`model`

7.2 Uses

- Hardware-Hiding Module
- Training and Testing Module (14)

7.3 Syntax

7.3.1 Exported Constants

None.

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
`save_model`	-	*bool*	`PermissionError`
`load_trained_classifier`	$\mathbf{A}_i$, *str*	$\mathbf{Z}_+$	-

7.4 Semantics

7.4.1 State Variables

None.

7.4.2 Environment Variables

None.

7.4.3 Assumptions

None.

7.4.4 Access Routine Semantics

`save_model`():

- transition: Writes the current state of the ANN model (weights and biases) to a `.npy` file as a *dict*.
- output: Returns `True` if model is saved successfully.

- exception: Raises `PermissionError` if the module lacks the necessary permissions to write to `modelFile`. It may also raise an IOError if there are issues with the file system, such as insufficient storage space.

`load_trained_classifier(input_image, model_name)`:

- transition: Loads a trained classifier and uses it to predict a class for the input image.
- output: The predicted class as $\mathbf{Z}_+$, based on the vector of input image and name of the saved model.
- exception: None.

7.4.5 Local Functions

`load_model(file_name)`:

- transition: Load model parameters from a specified code.npy file.
- output: The model parameters stored in the file as a *dict*.
- exception: Raises `FileNotFoundError` if `modelFile` does not exist or cannot be accessed. Additionally, an exception may be raised for data corruption or format mismatch, indicating issues with the integrity or compatibility of the stored model data.

8 MIS of Output Module

8.1 Module

`output`

8.2 Uses

- Hardware-Hiding Module
- Input Classifier Module (9)

8.3 Syntax

8.3.1 Exported Constants

None.

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
`set_class_name`	-	*str*	-
`save_feedback`	-	-	-

8.4 Semantics

8.4.1 State Variables

`class_name`: The *str* as the name of the class to which the image has been classified.

8.4.2 Environment Variables

None.

8.4.3 Assumptions

None.

8.4.4 Access Routine Semantics

`set_class_name`():

- transition: Determines the class name of an image using the Input Classifier Module (9).
- output: he class name determined by the Input Classifier Module (9).

- exception: None.

`save_feedback`():

- transition: ollects and saves user feedback on the classification result to a text file.
- output: None.
- exception: None.

8.4.5 Local Functions

`append_to_file`(`file_path, sentence`):

- transition: Appends a given feedback to a file and prints a success message or error message depending on the outcome of the file operation.
- output: None.
- exception: Raises `FileNotFoundError` if the destination `.txt` file is not found. `IOError` will be raised, if there is a problem with writing to the file.

9 MIS of Input Classifier Module

9.1 Module

`classifier`

9.2 Uses

- Hardware-Hiding Module
- Saved ANN Module (7)
- Input Preparing and Preprocessing Module (12)

9.3 Syntax

9.3.1 Exported Constants

None.

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
`set_image_pixel`	-	-	-
`get_class`	-	*str*	-

9.4 Semantics

9.4.1 State Variables

- `input_image`: An array of input image's pixels after preprocessing.
- `class_name`: The *str* as the name of the class to which the image has been classified.

9.4.2 Environment Variables

None.

9.4.3 Assumptions

None.

9.4.4 Access Routine Semantics

`set_image_pixels`():

- transition: Receives the array of input image from Input Preparing and Preprocessing Module (12) and saves in `input_image`.
- output: None.
- exception: None.

`get_class`():

- transition: Classifies the preprocessed image using a pretrained model from (7) and maps the output to a class name.
- output: The class of the input image.
- exception: None.

10 MIS of Input Image Module

10.1 Module

`input`

10.2 Uses

- Hardware-Hiding Module

10.3 Syntax

10.3.1 Exported Constants

- `HEIGHT`: A value ($\mathbf{Z}_+$) describing acceptable height of input image (currently 32).
- `WIDTH`: A value ($\mathbf{Z}_+$) describing acceptable width of input image (currently 32).
- `IMAGE_FORMAT`: A list of strings (*str*) of acceptable types of input image (currently PNG and JPEG).

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
`set_image`	-	$\mathbf{M}_{ijk}$	`FileNotFoundError`, `InvalidSize`, `InvalidFormat`

10.4 Semantics

10.4.1 State Variables

None.

10.4.2 Environment Variables

None.

10.4.3 Assumptions

None.

10.4.4 Access Routine Semantics

`set_image`():

- transition: Receives the input image from end user and returns its matrix as `inputImage`.
- output: None.
- exception: Raises `FileNotFoundError` if `inputImagePath` does not exist or cannot be accessed. Also, `InvalidSize` is raised when the size of input image in not compatible with `HEIGHT` or `WIDTH`. Additionally, `InvalidFormat` is thrown if the input image's format is not compatible with `IMAGE_FORMAT`.

10.4.5 Local Functions

None.

11 MIS of Training Model Module

11.1 Module

`training_model`

11.2 Uses

- Data Preparing and Preprocessing Module (13)

11.3 Syntax

11.3.1 Exported Constants

- `LAYERS_NUMBER`: A value ($\mathbf{Z}_+$) describing the number of neural network's layers.
- `LAYERS_NEURONS`: An array including each layer's number of neurons.

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
`create_gradients`	-	$\mathbf{A}_i$	-

11.4 Semantics

11.4.1 State Variables

None.

11.4.2 Environment Variables

None.

11.4.3 Assumptions

None.

11.4.4 Access Routine Semantics

`create_gradients`():

- transition: Creates zero arrays for all needed gradients based on the `LAYERS_NUMBER` and `LAYERS_NEURONS`.
- output: All gradients' zero vector ($\mathbf{A}_i$)
- exception: None.

11.4.5 Local Functions

None.

12 Input Preparing and Preprocessing Module

12.1 Module

input_prep

12.2 Uses

Data Preparing and Preprocessing Module (13)
Input Image Module (10)

12.3 Syntax

12.3.1 Exported Constants

None.

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
get_input	-	$\mathbf{A}_i$	-

12.4 Semantics

12.4.1 State Variables

- input_image: Incling input image pixels.

12.4.2 Environment Variables

None.

12.4.3 Assumptions

None.

12.4.4 Access Routine Semantics

get_input():

- transition: Execute the preprocessing steps and return the processed image. This method sequentially calls other methods to load, convert to grayscale, normalize, and flatten the image data.
- output: an $\mathbf{A}_i$ including prepared and preprocessed input image.
- exception: None.

12.4.5 Local Functions

- `set_image_pixels`():
  - transition: Loads an image using the Input Image Module (10). and store it in `input_image`.
  - output: None.
  - exception: None.
- `rgb2gray`():
  - transition: Uses Data Preparing and Preprocessing Module (13) to convert RGB data (`input_image`) into grayscale to reduce the complexity.
  - output: None.
  - exception: None.
- `prep_pixels`():
  - transition: Normalizes grayscaled (`input_image`) to change the range of data between 0 and 1, using Data Preparing and Preprocessing Module (13).
  - output: None.
  - exception: None.
- `flat_data`():
  - transition: Data is flatten since (`input_image`) should be vectorized with the size of 1024. After grayscaling input image is a $\mathbf{M}_{ij}$. this should be an $\mathbf{A}_i$ to be used by implemented model.
  - output: None.
  - exception: None.

13 Data Preparing and Preprocessing Module

13.1 Module

`data`

13.2 Uses

None.

13.3 Syntax

13.3.1 Exported Constants

None.

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
`get_dataset`	-	$[\mathbf{M}_{ij}, \mathbf{A}_i]$, $[\mathbf{M}_{ij}, \mathbf{A}_i]$	-
`rgb2gray`	$[\mathbf{M}_{ij}]$, *bool*	-	-
`prep_pixels`	$[\mathbf{M}_{ij}]$	-	-

13.4 Semantics

13.4.1 State Variables

- `train_data`: Data structure holding train data images and their labels. Since train images after processing are vectors ($\mathbf{A}_i$), a list of these images is a matix ($\mathbf{M}_{ij}$). Alos, labels are saving in a vector ($\mathbf{A}_i$). Consequently, this data structure is a list in [$\mathbf{M}_{ij}$, $\mathbf{A}_i$] format.
- `test_data`: Data structure holding test data images and their labels. Since test images after processing are vectors ($\mathbf{A}_i$), a list of these images is a matix ($\mathbf{M}_{ij}$). Alos, labels are saving in a vector ($\mathbf{A}_i$). Consequently, this data structure is a list in [$\mathbf{M}_{ij}$, $\mathbf{A}_i$] format.
- `train_images`: A list incling traing data images.
- `test_images`: Alist incling test data images.

13.4.2 Environment Variables

None.

13.4.3 Assumptions

None.

13.4.4 Access Routine Semantics

`get_dataset`():

- transition: Process and retrieve the training and testing datasets. This method orchestrates various data processing steps including loading data, converting to grayscale, normalizing pixel values, flattening images, and shuffling data. It ensures the dataset is properly formatted and prepared for use in machine learning models.
- output: `train_data`, `test_data`
- exception: None.

`rgb2gray`(images, input_image):

- transition: Converts RGB data into grayscale in order to reduce complexity. First input is a numpy array representing one or more images in RGB format. If a single image, it should be a $\mathbf{M}_{ijk}$ array (height, width, color_channels). If multiple images, it should be a 4D array (number_images, height, width, color_channels). Second input as a *bool*. It is a flag to indicate if the provided 'images' parameter is a single image (True) or a batch of images (False). Default is False.
- output: The transformed grayscale images. If 'images' was a single image, the return is a $\mathbf{M}_{ij}$ (height, width). If 'images' was a batch of images, the return is a $\mathbf{M}_{ijk}$ array (number_images, height, width).
- exception: None.

`prep_pixels`(images):

- transition: Normalizes grayscaled images to change the range of data between 0 and 1.
- output: Normalized images.
- exception: None.

13.4.5 Local Functions

- `save_data`():
  - transition: Downloads the file from Google Drive and saves it locally.
  - output: None.

  - exception: This function raises `UnableToDownload`, when there is a problem with downloading or extracting data.

- `load_data`():
  - transition: Loads `train_data` and `test_data`, ecnods the labels. The dataset is load using `keras.datasets`. This function updates `train_images` and `test_images` as well.
  - output: None.
  - exception: This function raises `UnableToDownload`, when there is a problem with downloading or extracting data.

- `flat_data`(images):
  - transition: Data is flatten since imagws should be vectorized with the size of 1024. After grayscaling each image is a $\mathbf{M}_{ij}$. These images should be an $\mathbf{A}_i$ to be used in training and testing process by implemented model.
  - output: Flattened images.
  - exception: None.

- `shuffle_data`(data, images):
  - transition: Combines images with their corresponding labels, one-hot encodes the labels, and then shuffles the combined data for randomness.
  - output: `train_data` and `test_data`.
  - exception: A tuple as [$\mathbf{M}_{ij}$, $\mathbf{A}_i$], where each tuple contains a shuffled image and its label.

14 MIS of Training and Testing Module

The details of fucntions using here are described in SRS document Djavaherpour (2024b).

14.1 Module

`train_and_test`

14.2 Uses

- Training Model Module (11)
- Data Preparing and Preprocessing Module (13)

14.3 Syntax

14.3.1 Exported Constants

- `BATCH_SIZE`: The partition size of the dataset for each step of learning, typically a power of two.
- `LEARNING_RATE`: Speed at which the model learns; controls adjustments to the weights.
- `EPOCHS`: Total number of training cycles through the entire dataset.

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
`train`	-	*dict*, $\mathbf{A}_i$, $\mathbf{Z}_+$, $\mathbf{Z}_+$	-
`result`	*dict*, $\mathbf{A}_i$	-	-
`calculate_percentage_of_accuracy`	$\mathbf{M}_{ij}$, *dict*, *bool*	-	-
	or $[\mathbf{M}_{ijk}, \mathbf{M}_i]$, *dict*, *bool*	-	-

14.4 Semantics

14.4.1 State Variables

- `layers`: Layers dimensions are defined based on the model architecture and are saved in an $\mathbf{A}_i$ named layers.

14.4.2 Environment Variables

None.

14.4.3 Assumptions

None.

14.4.4 Access Routine Semantics

`train`():

- transition: Trains the model based on constant variables defined in 11. Gradient arrays and train data are needed for training. The training process involves initializing parameters, performing forward and backward passes for each batch, and updating the parameters using the computed gradients. The process is repeated for a specified number of epochs.
- output: The final trained parameters, cost for each epoch, start time, and end time of training.
- exception: None.

`result`(epochs_costs, trained_params):

- transition: Displays the accuracy percentages for training and testing datasets and plots the training cost over epochs. This method calculates and prints the accuracy on both the training and testing datasets using the trained model parameters. It also generates a plot of the training costs over epochs to visually assess the model's learning progress.
- output: This method prints the accuracy percentages for the training and test datasets to the console and displays a line plot of the training costs over epochs, showing changes in cost with each epoch.
- exception: None.

`calculate_percentage_of_accuracy`(data, parameters, input_image = False):

- transition: Calculates the accuracy of the neural network model on a given dataset or finds the predicted class of an input image. Accuracy is determined by comparing the predicted labels against the actual labels and calculating the percentage of correct predictions.
- output: Accuracy percentage if input_image is `False`, or class index if `True`.
- exception: None.

14.4.5 Local Functions

`set_layers`():

- transition: `layers` is set based on Train Model Module (11) and the gradients.
- output: None.
- exception: None.
- `sigmoid`(x):
  - transition: Calculates sigmoid function for x, as the activation function.
  - output: Sigmoid value of x.
  - exception: None.
- `initialize_parameters`(layers):
  - transition: Allocates random normal weights ($\mathbf{M}_{ij}$) and zero biases ($\mathbf{A}_i$) for each layer.
  - output: Returns a dictionary (named `parameters`) that the keys define weights or biases, and the values are allocated random numbers to each of them.
  - exception: None.
- `compute_cost`(predicted, actual):
  - transition: Calculates the sum of the squared errors based on the predicted and actual values.
  - output: Returns the sum of the squared errors
  - exception: None.
- `feed_forward`(predicted, parameters, layersNumb):
  - transition: Calculates feedforwarding process as described in SRS Djavaherpour (2024b). This is done by using the predicted value of previous step, parameters and the number of layers, `parameters` dictionary and the number of layers.
  - output: Returns the new predicted value and a `cache` including new and old parameters.
  - exception: None.
- `extract_parameters`(cache):
  - transition: Extracts parameters saved during forwardfeeding from the cache, based on `layers`, `parameters` dictionary and the number of layers.

  - output: Returns extracted parameters.
  - exception: None.

- `backpropagation`(cache, predicted, actual, layers):
  - transition: Calculates backpropagation process as described in SRS Djavaherpour (2024b) to calculate gradients of wights and biases.
  - output: A dictionary as gradients that keys are labels of gradients to define weights or biases, and values are gradients.
  - exception: None.

References

Tanya Djavaherpour. Module guide. https://github.com/tanya-jp/ANN-CAS741/blob/main/docs/Design/SoftArchitecture/MG.pdf, 2024a.

Tanya Djavaherpour. System requirements specification. https://github.com/tanya-jp/ANN-CAS741/blob/main/docs/SRS/SRS.pdf, 2024b.

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.