

# Contents

<b>1</b>	<b>Class Index</b>	<b>1</b>
1.1	Class List . . . . .	1
<b>2</b>	<b>File Index</b>	<b>3</b>
2.1	File List . . . . .	3
<b>3</b>	<b>Class Documentation</b>	<b>5</b>
3.1	DatabaseOps Class Reference . . . . .	5
3.1.1	Detailed Description . . . . .	5
3.1.2	Member Function Documentation . . . . .	5
3.1.2.1	ExportData() . . . . .	5
3.1.2.2	ImportData() . . . . .	6
3.2	InputLayer Class Reference . . . . .	6
3.2.1	Detailed Description . . . . .	7
3.2.2	Constructor & Destructor Documentation . . . . .	7
3.2.2.1	InputLayer() . . . . .	7
3.2.3	Member Function Documentation . . . . .	7
3.2.3.1	AddTrain() . . . . .	7
3.2.3.2	ApplyAlphas() . . . . .	7
3.2.3.3	ResetTrains() . . . . .	8
3.2.3.4	UpdateAlphas() . . . . .	8
3.2.4	Member Data Documentation . . . . .	8
3.2.4.1	basis . . . . .	8
3.2.4.2	index . . . . .	8

3.2.4.3	trains	8
3.2.4.4	w	8
3.3	MatrixOps Class Reference	9
3.3.1	Detailed Description	9
3.3.2	Member Function Documentation	9
3.3.2.1	Conv()	9
3.3.2.2	Dot()	10
3.3.2.3	Multiply()	10
3.3.2.4	Sum()	10
3.3.2.5	SumArrays()	11
3.3.2.6	SumColumns()	11
3.3.2.7	Transpose()	12
3.4	Network Class Reference	12
3.4.1	Detailed Description	13
3.4.2	Constructor & Destructor Documentation	13
3.4.2.1	Network()	13
3.4.3	Member Function Documentation	13
3.4.3.1	CrossValidate()	13
3.4.3.2	ExportData()	13
3.4.3.3	ImportData()	14
3.4.3.4	ResetNetwork()	14
3.4.3.5	Run()	14
3.4.3.6	Train()	14
3.4.3.7	Validate()	15
3.4.3.8	ValidateDataset()	15
3.4.4	Member Data Documentation	16
3.4.4.1	inputLayer	16
3.4.4.2	outputLayer	16
3.5	OutputLayer Class Reference	16
3.5.1	Detailed Description	17

3.5.2	Constructor & Destructor Documentation . . . . .	17
3.5.2.1	OutputLayer() . . . . .	17
3.5.3	Member Function Documentation . . . . .	17
3.5.3.1	ComputeErrors() . . . . .	17
3.5.3.2	ComputeOutput() . . . . .	17
3.5.3.3	ComputeWinner() . . . . .	18
3.5.3.4	Reset() . . . . .	18
3.5.3.5	UpdateBetas() . . . . .	18
3.5.3.6	UpdateGammas() . . . . .	18
3.5.4	Member Data Documentation . . . . .	19
3.5.4.1	basis . . . . .	19
3.5.4.2	gammas . . . . .	19
3.5.4.3	u . . . . .	19
3.5.4.4	v . . . . .	19
3.5.4.5	y . . . . .	19
3.6	TrainingPool Class Reference . . . . .	19
3.6.1	Detailed Description . . . . .	20
3.6.2	Member Function Documentation . . . . .	20
3.6.2.1	InitTraining() . . . . .	20
3.6.2.2	Join() . . . . .	20
3.6.2.3	TrainAsync() . . . . .	21
3.6.3	Member Data Documentation . . . . .	21
3.6.3.1	runningInstances . . . . .	21
3.7	Utils Class Reference . . . . .	21
3.7.1	Detailed Description . . . . .	22
3.7.2	Member Function Documentation . . . . .	22
3.7.2.1	GenerateAlphaBasis() . . . . .	22
3.7.2.2	GenerateBetaBasis() . . . . .	22
3.7.2.3	GenerateSpikes() . . . . .	22
3.7.2.4	GetTestData() . . . . .	23
3.7.2.5	GetTrainingData() . . . . .	23
3.7.2.6	PrintLine() . . . . .	23
3.7.2.7	RateEncode() . . . . .	24

<b>4 File Documentation</b>	<b>25</b>
4.1 Constants.h File Reference	25
4.1.1 Variable Documentation	25
4.1.1.1 CLASSES	25
4.1.1.2 Ka	26
4.1.1.3 Kb	26
4.1.1.4 LEARNING_RATE	26
4.1.1.5 NEURONS_IN	26
4.1.1.6 P_RANGE	26
4.1.1.7 T	26
4.1.1.8 TEST_IMAGES_PATH	26
4.1.1.9 TEST_LABELS_PATH	26
4.1.1.10 TRAIN_IMAGES_PATH	27
4.1.1.11 TRAIN_LABELS_PATH	27
4.1.1.12 TYI	27
4.1.1.13 TYO	27
4.2 SNN.cpp File Reference	27
4.2.1 Function Documentation	27
4.2.1.1 main()	27
<b>Index</b>	<b>29</b>

# Chapter 1

## Class Index

### 1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">DatabaseOps</a>	5
<a href="#">InputLayer</a>	6
<a href="#">MatrixOps</a>	9
<a href="#">Network</a>	12
<a href="#">OutputLayer</a>	16
<a href="#">TrainingPool</a>	19
<a href="#">Utils</a>	21



## Chapter 2

# File Index

### 2.1 File List

Here is a list of all documented files with brief descriptions:

<a href="#">Constants.h</a>	25
<b>DatabaseOps.h</b>	??
<b>Layer.h</b>	??
<b>MatrixOps.h</b>	??
<b>Network.h</b>	??
<a href="#">SNN.cpp</a>	27
<b>stdafx.h</b>	??
<b>targetver.h</b>	??
<b>TrainingPool.h</b>	??
<b>Utils.h</b>	??





## Chapter 3

# Class Documentation

### 3.1 DatabaseOps Class Reference

```
#include <DatabaseOps.h>
```

#### Static Public Member Functions

- static void [ExportData](#) ([InputLayer](#) \*inputLayer, [OutputLayer](#) \*outputLayer, string fileName)
- static void [ImportData](#) ([InputLayer](#) \*inputLayer, [OutputLayer](#) \*outputLayer, string fileName)

#### 3.1.1 Detailed Description

This class contains methods to perform I/O operations on a SQLite 3 database

#### 3.1.2 Member Function Documentation

##### 3.1.2.1 ExportData()

```
void DatabaseOps::ExportData (
    InputLayer * inputLayer,
    OutputLayer * outputLayer,
    string fileName ) [static]
```

Generates a SQLite database at the specified path and exports all the weights and hyperparameters to it

#### Parameters

<i>inputLayer</i>	The input layer
<i>outputLayer</i>	The output layer
<i>fileName</i>	The output file name

### 3.1.2.2 ImportData()

```
void DatabaseOps::ImportData (
    InputLayer * inputLayer,
    OutputLayer * outputLayer,
    string fileName ) [static]
```

Reads and imports the weights of the network from the specified SQLite database

#### Parameters

<i>inputLayer</i>	The input layer
<i>outputLayer</i>	The output layer
<i>fileName</i>	The input file name

The documentation for this class was generated from the following files:

- DatabaseOps.h
- DatabaseOps.cpp

## 3.2 InputLayer Class Reference

```
#include <Layer.h>
```

### Public Member Functions

- [InputLayer](#) ()
- void [AddTrain](#) (array< bool, [T](#) > &train)
- void [ResetTrains](#) ()
- array< array< double, [T-1](#) >, [CLASSES](#) \*[NEURONS\\_IN](#) > [ApplyAlphas](#) ()
- void [UpdateAlphas](#) (array< array< double, [T](#) >, [CLASSES](#) > &errors)

### Public Attributes

- array< array< double, [TYI](#) >, [CLASSES](#) \*[NEURONS\\_IN](#) > [w](#)

### Protected Attributes

- array< array< bool, [T](#) >, [NEURONS\\_IN](#) > [trains](#)
- array< array< double, [Ka](#) >, [TYI](#) > [basis](#)
- short [index](#) = 0

### 3.2.1 Detailed Description

This class represents an input layer in the neural network and contains all the methods to perform operations on it

### 3.2.2 Constructor & Destructor Documentation

#### 3.2.2.1 InputLayer()

```
InputLayer::InputLayer ( )
```

The cosntructor initialises the values of the weights and the basis matrix

### 3.2.3 Member Function Documentation

#### 3.2.3.1 AddTrain()

```
void InputLayer::AddTrain (
    array< bool, T > & train )
```

Adds a train of spikes to the array, as position [index]

#### Parameters

<i>train</i>	The train of spikes
--------------	---------------------

#### 3.2.3.2 ApplyAlphas()

```
array< array< double, T-1 >, CLASSES *NEURONS_IN > InputLayer::ApplyAlphas ( )
```

Convolve alphas with the trains and apply the basis matrix. This is one part of the process to compute the potential of the output neuron

#### Returns

The preprocessed spike trains

### 3.2.3.3 ResetTrains()

```
void InputLayer::ResetTrains ( )
```

Reset the values of the trains and index

### 3.2.3.4 UpdateAlphas()

```
void InputLayer::UpdateAlphas (
    array< array< double, T >, CLASSES > & errors )
```

Updates the weights w using the errors errors The errors for updating the weights

## 3.2.4 Member Data Documentation

### 3.2.4.1 basis

```
array<array<double, Ka>, TYI> InputLayer::basis [protected]
```

The basis matrix A contains the basis functions to apply to the weights w

### 3.2.4.2 index

```
short InputLayer::index = 0 [protected]
```

Index of next train in the array

### 3.2.4.3 trains

```
array<array<bool, T>, NEURONS_IN> InputLayer::trains [protected]
```

The trains of spikes generated by rate encoding process

### 3.2.4.4 w

```
array<array<double, TYI>, CLASSES*NEURONS_IN> InputLayer::w
```

The weights w for every synapse

The documentation for this class was generated from the following files:

- Layer.h
- Layer.cpp

## 3.3 MatrixOps Class Reference

```
#include <MatrixOps.h>
```

### Static Public Member Functions

- static array< double, [T-1](#) > [Conv](#) (array< bool, [T](#) > const &f, array< double, [TYI](#) > const &g)
- static array< double, [T-1](#) > [SumColumns](#) (array< array< double, [T-1](#) >, [CLASSES](#) \*[NEURONS\\_IN](#) > &vect, const short cl)
- template<std::size\_t SIZE>  
static void [Multiply](#) (double c, array< double, SIZE > &vect)
- template<std::size\_t ROWS, std::size\_t COLS>  
static array< double, ROWS > [Dot](#) (array< array< double, COLS >, ROWS > &basis, array< double, COLS > &weight)
- template<std::size\_t SIZE>  
static double [Sum](#) (const array< double, SIZE > &vect)
- template<std::size\_t ROWS, std::size\_t COLS>  
static array< array< double, ROWS >, COLS > [Transpose](#) (array< array< double, COLS >, ROWS > &input)
- template<std::size\_t SIZE>  
static array< double, SIZE > [SumArrays](#) (array< double, SIZE > a, array< double, SIZE > b)

### 3.3.1 Detailed Description

This class contains custom matrix operations

### 3.3.2 Member Function Documentation

#### 3.3.2.1 Conv()

```
array< double, T-1 > MatrixOps::Conv (
    array< bool, T > const & f,
    array< double, TYI > const & g ) [static]
```

Performs a partial convolution to aggregate the trains and alpha

#### Parameters

<i>f</i>	The spike train
<i>g</i>	The weights vector

#### Returns

The result of the convolution

### 3.3.2.2 Dot()

```
template<std::size_t ROWS, std::size_t COLS>
array< double, ROWS > MatrixOps::Dot (
    array< array< double, COLS >, ROWS > & basis,
    array< double, COLS > & weight ) [inline], [static]
```

Performs the dot product

#### Parameters

<i>ROWS</i>	The number of rows
<i>COLS</i>	The number of columns
<i>basis</i>	The basis matrix
<i>weight</i>	The weight vector

#### Returns

the result of the operation

### 3.3.2.3 Multiply()

```
template<std::size_t SIZE>
void MatrixOps::Multiply (
    double c,
    array< double, SIZE > & vect ) [inline], [static]
```

Multiplies an array by a constant

#### Parameters

<i>SIZE</i>	The length of the array
<i>c</i>	The constant
<i>vect</i>	The array

### 3.3.2.4 Sum()

```
template<std::size_t SIZE>
double MatrixOps::Sum (
    const array< double, SIZE > & vect ) [inline], [static]
```

Sums all the values in a vector

#### Parameters

<i>SIZE</i>	The length of the array
-------------	-------------------------

## Parameters

<i>ROWS</i>	The number of rows
<i>COLS</i>	The number of columns
<i>vect</i>	The vector

## Returns

The sum of the elements

## 3.3.2.5 SumArrays()

```
template<std::size_t SIZE>
array< double, SIZE > MatrixOps::SumArrays (
    array< double, SIZE > a,
    array< double, SIZE > b ) [inline], [static]
```

Performs element-wise sum of the elements in the arrays

## Parameters

<i>SIZE</i>	The length of the arrays
<i>a</i>	Array 1
<i>b</i>	Array 2

## Returns

The element-wise sum of the arrays

## 3.3.2.6 SumColumns()

```
array< double, T-1 > MatrixOps::SumColumns (
    array< array< double, T-1 >, CLASSES *NEURONS_IN > & vect,
    const short cl ) [static]
```

Sum the matrix along the columns, but only every cl rows

## Parameters

<i>vect</i>	The matrix
<i>cl</i>	The frequency of rows to be summed, i.e. cl=2 half of the rows are summed

**Returns**

The sum

**3.3.2.7 Transpose()**

```
template<std::size_t ROWS, std::size_t COLS>
array< array< double, ROWS >, COLS > MatrixOps::Transpose (
    array< array< double, COLS >, ROWS > & input ) [inline], [static]
```

Performs a matrix transpose

**Parameters**

<i>ROWS</i>	The number of rows
<i>COLS</i>	The number of columns
<i>input</i>	The input matrix

**Returns**

The transposed matrix

The documentation for this class was generated from the following files:

- MatrixOps.h
- MatrixOps.cpp

**3.4 Network Class Reference**

```
#include <Network.h>
```

**Public Member Functions**

- [Network](#) ()
- char [Run](#) (array< unsigned char, [NEURONS\\_IN](#) > image)
- template<std::size\_t FILTER\_SIZE>
  - void [Train](#) (short epochs, int trainingImages=60000, vector< pair< array< unsigned char, [NEURONS\\_IN](#) >, unsigned char >> \*trainingData=nullptr, array< unsigned char, FILTER\_SIZE > \*filter=nullptr)
- void [ImportData](#) (string fileName="data.db")
- void [ExportData](#) (string fileName="data.db")
- template<std::size\_t FILTER\_SIZE>
  - int [Validate](#) (int testImages=10000, bool testSet=true, array< unsigned char, FILTER\_SIZE > \*filter=NULL)
- int [ValidateDataset](#) (vector< pair< array< unsigned char, [NEURONS\\_IN](#) >, unsigned char >> &trainingSet)
- int [CrossValidate](#) ()



## Protected Member Functions

- void [ResetNetwork](#) ()

## Protected Attributes

- [InputLayer](#) inputLayer
- [OutputLayer](#) outputLayer

### 3.4.1 Detailed Description

The [Network](#) class contains the methods to perform all the possible operations on the network

### 3.4.2 Constructor & Destructor Documentation

#### 3.4.2.1 Network()

```
Network::Network ( )
```

The constructor instantiates the layers

### 3.4.3 Member Function Documentation

#### 3.4.3.1 CrossValidate()

```
int Network::CrossValidate ( )
```

Performs 10-folds cross-validation on the entire training set - single epoch

#### Returns

The average of correct classification - Max 6000

#### 3.4.3.2 ExportData()

```
void Network::ExportData (
    string fileName = "data.db" )
```

Export network weights to a database file

**Parameters**

<i>fileName</i>	The file name of the database to which export the weights
-----------------	---

**3.4.3.3 ImportData()**

```
void Network::ImportData (
    string fileName = "data.db" )
```

Import network weights from a database file

**Parameters**

<i>fileName</i>	The file name of the database from which import the weights
-----------------	---

**3.4.3.4 ResetNetwork()**

```
void Network::ResetNetwork ( ) [protected]
```

To reset both layers after processing every image

**3.4.3.5 Run()**

```
char Network::Run (
    array< unsigned char, NEURONS_IN > image )
```

Run the network on an image and return the predicted class

**Parameters**

<i>image</i>	The input image to be classified
--------------	----------------------------------

**Returns**

The predicted class of the image

**3.4.3.6 Train()**

```
template<std::size_t FILTER_SIZE>
void Network::Train (
```

```

        short epochs,
        int trainingImages = 60000,
        vector< pair< array< unsigned char, NEURONS_IN >, unsigned char >> * trainingData = nullptr,
        array< unsigned char, FILTER_SIZE > * filter = nullptr )

```

Trains the network

#### Parameters

<i>FILTER_SIZE</i>	The number of elements in the filter
<i>epochs</i>	The number of iterations to perform
<i>trainingImages</i>	The number of images to train the network on
<i>trainingData</i>	The data to train the network on - if it's not nullptr, it overrides the trainingImages parameter
<i>filter</i>	The filter to limit the labels in the training - if nullptr, all 10 digits are considered

#### 3.4.3.7 Validate()

```

template<std::size_t FILTER_SIZE>
int Network::Validate (
    int testImages = 10000,
    bool testSet = true,
    array< unsigned char, FILTER_SIZE > * filter = NULL )

```

Validates the network on any data

#### Parameters

<i>FILTER_SIZE</i>	The number of elements in the filter
<i>testImages</i>	The number of images used for validation. The range should be [1 10000] for the test set and [1 60000] for the training one
<i>testSet</i>	If true the test set is used, otherwise the training set
<i>filter</i>	The filter to limit the labels in the training - if nullptr, all 10 digits are considered

#### Returns

The number of correct classifications

#### 3.4.3.8 ValidateDataset()

```

int Network::ValidateDataset (
    vector< pair< array< unsigned char, NEURONS_IN >, unsigned char >> & trainingSet )

```

Validates the network on the given images

## Parameters

<i>trainingSet</i>	The images that should be used to validate the network on
--------------------	---

## Returns

The number of correct classifications

### 3.4.4 Member Data Documentation

#### 3.4.4.1 inputLayer

`InputLayer` `Network::inputLayer` [protected]

The input layer

#### 3.4.4.2 outputLayer

`OutputLayer` `Network::outputLayer` [protected]

The output layer

The documentation for this class was generated from the following files:

- Network.h
- Network.cpp

## 3.5 OutputLayer Class Reference

```
#include <Layer.h>
```

### Public Member Functions

- `OutputLayer` ()
- void `Reset` ()
- void `ComputeOutput` (array< array< double, `T-1` >, `CLASSES` \*`NEURONS_IN` > &synapsesOut)
- array< array< double, `T` >, `CLASSES` > `ComputeErrors` (unsigned char label) const
- char `ComputeWinner` () const
- void `UpdateBetas` (array< array< double, `T` >, `CLASSES` > &errors)
- void `UpdateGammas` (array< array< double, `T` >, `CLASSES` > &errors)

### Public Attributes

- `array< array< double, TYO >, CLASSES > v`
- `array< double, CLASSES > gammas`

### Protected Attributes

- `array< array< double, T >, CLASSES > u`
- `array< array< bool, T >, CLASSES > y`
- `array< array< double, Kb >, TYO > basis`

## 3.5.1 Detailed Description

This class represents an output layer in the neural network and contains all the methods to perform operations on it

## 3.5.2 Constructor & Destructor Documentation

### 3.5.2.1 OutputLayer()

```
OutputLayer::OutputLayer ( )
```

The cosnstructor initialises the values of the weights and the basis matrix

## 3.5.3 Member Function Documentation

### 3.5.3.1 ComputeErrors()

```
array< array< double, T >, CLASSES > OutputLayer::ComputeErrors (
    unsigned char label ) const
```

Computes the errors at every time t for the given class label The class for which compute the errors

#### Returns

An array of errors for every class label

### 3.5.3.2 ComputeOutput()

```
void OutputLayer::ComputeOutput (
    array< array< double, T-1 >, CLASSES *NEURONS\_IN > & synapsesOut )
```

Computes the potential and output starting from the preprocessed trains

**Parameters**

<i>synapsesOut</i>	The preprocessed spike trains from the input layer
--------------------	--

**3.5.3.3 ComputeWinner()**

```
char OutputLayer::ComputeWinner ( ) const
```

Returns the "Winner" of the network: uses rate decoding to determine the class with the most spikes

**Returns**

The prediction of the network

**3.5.3.4 Reset()**

```
void OutputLayer::Reset ( )
```

Reset the values of the potential and output

**3.5.3.5 UpdateBetas()**

```
void OutputLayer::UpdateBetas (
    array< array< double, T >, CLASSES > & errors )
```

Updates the weights  $v$  using the errors  $errors$  The errors for updating the weights

**See also**

[ComputeErrors](#)

**3.5.3.6 UpdateGammas()**

```
void OutputLayer::UpdateGammas (
    array< array< double, T >, CLASSES > & errors )
```

Updates the biases using the errors  $errors$  The errors for updating the biases

**See also**

[ComputeErrors](#)

### 3.5.4 Member Data Documentation

#### 3.5.4.1 basis

```
array<array<double, Kb>, TYO> OutputLayer::basis [protected]
```

The basis matrix B contains the basis functions to apply to the weights v

#### 3.5.4.2 gammas

```
array<double, CLASSES> OutputLayer::gammas
```

The biases of every class

#### 3.5.4.3 u

```
array<array<double, T>, CLASSES> OutputLayer::u [protected]
```

The potential of every output neuron at every time t

#### 3.5.4.4 v

```
array<array<double, TYO>, CLASSES> OutputLayer::v
```

The weights of the feedback kernel

#### 3.5.4.5 y

```
array<array<bool, T>, CLASSES> OutputLayer::y [protected]
```

The output spike trains of every class

The documentation for this class was generated from the following files:

- Layer.h
- Layer.cpp

## 3.6 TrainingPool Class Reference

```
#include <TrainingPool.h>
```

### Static Public Member Functions

- static void [TrainAsync](#) (short epochs, int trainingImages, string dbOut, bool validate)
- static void [Join](#) ()

### Static Protected Member Functions

- static void [InitTraining](#) (short epochs, int trainingImages, string dbOut, bool validate)

### Static Protected Attributes

- static vector< thread > [runningInstances](#)

## 3.6.1 Detailed Description

This class contains methods for performing multiple training sessions on networks in parallel and exporting the results

## 3.6.2 Member Function Documentation

### 3.6.2.1 InitTraining()

```
void TrainingPool::InitTraining (
    short epochs,
    int trainingImages,
    string dbOut,
    bool validate ) [static], [protected]
```

Method that makes the call to the Train function of the [Network](#) class

#### Parameters

<i>epochs</i>	The number of epochs
<i>trainingImages</i>	The number of images to train the network on. The number should be 1-60000
<i>dbOut</i>	The path of the database where the data should be exported
<i>validate</i>	True if validation on the entire test set should be performed after the training

### 3.6.2.2 Join()

```
void TrainingPool::Join ( ) [static]
```

Wait for all running threads to finish executing



### 3.6.2.3 TrainAsync()

```
void TrainingPool::TrainAsync (
    short epochs,
    int trainingImages,
    string dbOut,
    bool validate ) [static]
```

Trains a network in a parallel thread and exports the training data

#### Parameters

<i>epochs</i>	The number of epochs
<i>trainingImages</i>	The number of images to train the network on. The number should be 1-60000
<i>dbOut</i>	The path of the database where the data should be exported
<i>validate</i>	True if validation on the entire test set should be performed after the training

## 3.6.3 Member Data Documentation

### 3.6.3.1 runningInstances

```
vector< thread > TrainingPool::runningInstances [static], [protected]
```

A list of threads that are currently running

The documentation for this class was generated from the following files:

- TrainingPool.h
- TrainingPool.cpp

## 3.7 Utils Class Reference

```
#include <Utils.h>
```

### Static Public Member Functions

- static array< float, [NEURONS\\_IN](#) > [RateEncode](#) (array< unsigned char, [NEURONS\\_IN](#) > &image)
- static array< bool, [T](#) > [GenerateSpikes](#) (float probability)
- template<std::size\_t FILTER\_SIZE>  
static vector< pair< array< unsigned char, [NEURONS\\_IN](#) >, unsigned char > > [GetTrainingData](#) (int NumberOfImages, array< unsigned char, FILTER\_SIZE > \*filter=nullptr)
- template<std::size\_t FILTER\_SIZE>  
static vector< pair< array< unsigned char, [NEURONS\\_IN](#) >, unsigned char > > [GetTestData](#) (int NumberOfImages, array< unsigned char, FILTER\_SIZE > \*filter=nullptr)
- static array< array< double, [Ka](#) >, [TYI](#) > [GenerateAlphaBasis](#) ()
- static array< array< double, [Kb](#) >, [TYO](#) > [GenerateBetaBasis](#) ()
- static void [PrintLine](#) (string &&str)

### 3.7.1 Detailed Description

This class contains general methods used in the programme

### 3.7.2 Member Function Documentation

#### 3.7.2.1 GenerateAlphaBasis()

```
array< array< double, Ka >, TYI > Utils::GenerateAlphaBasis ( ) [static]
```

Generates the basis matrix A

##### Returns

The basis matrix A

#### 3.7.2.2 GenerateBetaBasis()

```
array< array< double, Kb >, TYO > Utils::GenerateBetaBasis ( ) [static]
```

Generates the basis matrix B

##### Returns

The basis matrix B

#### 3.7.2.3 GenerateSpikes()

```
array< bool, T > Utils::GenerateSpikes (
    float probability ) [static]
```

Generate spikes trains from probabilities

##### Parameters

<i>probability</i>	The probability of spiking
--------------------	----------------------------

##### Returns

The train of spikes

#### 3.7.2.4 GetTestData()

```
template<std::size_t FILTER_SIZE>
vector< pair< array< unsigned char, NEURONS_IN >, unsigned char > > Utils::GetTestData (
    int NumberOfImages,
    array< unsigned char, FILTER_SIZE > * filter = nullptr ) [static]
```

Returns the test data

##### Parameters

<i>FILTER_SIZE</i>	The number of elements in the filter
<i>NumberOfImages</i>	Number of images to read from the database
<i>filter</i>	The filter to limit the labels in the training - if nullptr, all 10 digits are considered

##### Returns

The a list of <image, label> pairs

#### 3.7.2.5 GetTrainingData()

```
template<std::size_t FILTER_SIZE>
vector< pair< array< unsigned char, NEURONS_IN >, unsigned char > > Utils::GetTrainingData (
    int NumberOfImages,
    array< unsigned char, FILTER_SIZE > * filter = nullptr ) [static]
```

Returns the training data

##### Parameters

<i>FILTER_SIZE</i>	The number of elements in the filter
<i>NumberOfImages</i>	Number of images to read from the database
<i>filter</i>	The filter to limit the labels in the training - if nullptr, all 10 digits are considered

##### Returns

The a list of <image, label> pairs

#### 3.7.2.6 PrintLine()

```
void Utils::PrintLine (
    string && str ) [static]
```

Thread-safe method for printing to terminal

**Parameters**

<i>str</i>	The string to be printed to terminal
------------	--------------------------------------

**3.7.2.7 RateEncode()**

```
array< float, NEURONS_IN > Utils::RateEncode (
    array< unsigned char, NEURONS_IN > & image ) [static]
```

Perform rate encode on the pixels of the image

**Parameters**

<i>image</i>	The input image
--------------	-----------------

**Returns**

An array of probabilities of spiking

The documentation for this class was generated from the following files:

- Utils.h
- Utils.cpp

## Chapter 4

# File Documentation

### 4.1 Constants.h File Reference

```
#include "stdafx.h"
#include <cmath>
#include <fstream>
```

#### Variables

- const short [CLASSES](#) = 10
- const short [NEURONS\\_IN](#) = 784
- const short [T](#) = 4
- const short [TYI](#) = 3
- const short [Ka](#) = [TYI](#)
- const short [TYO](#) = 2
- const short [Kb](#) = [TYO](#)
- const float [LEARNING\\_RATE](#) = 0.00005
- const std::pair< float, float > [P\\_RANGE](#) = { 0, 0.5 }
- const string [TRAIN\\_IMAGES\\_PATH](#) = "D:\\train-images.idx3-ubyte"
- const string [TRAIN\\_LABELS\\_PATH](#) = "D:\\train-labels.idx1-ubyte"
- const string [TEST\\_IMAGES\\_PATH](#) = "D:\\t10k-images.idx3-ubyte"
- const string [TEST\\_LABELS\\_PATH](#) = "D:\\t10k-labels.idx1-ubyte"

#### 4.1.1 Variable Documentation

##### 4.1.1.1 CLASSES

```
const short CLASSES = 10
```

Number of classes

#### 4.1.1.2 Ka

```
const short Ka = TYI
```

Number of alpha bases

#### 4.1.1.3 Kb

```
const short Kb = TYO
```

Number of beta bases

#### 4.1.1.4 LEARNING\_RATE

```
const float LEARNING_RATE = 0.00005
```

Learning rate

#### 4.1.1.5 NEURONS\_IN

```
const short NEURONS_IN = 784
```

Number of input neurons

#### 4.1.1.6 P\_RANGE

```
const std::pair<float, float> P_RANGE = { 0, 0.5 }
```

Range of probabilities for spike decoding

#### 4.1.1.7 T

```
const short T = 4
```

Length of the spike train

#### 4.1.1.8 TEST\_IMAGES\_PATH

```
const string TEST_IMAGES_PATH = "D:\\t10k-images.idx3-ubyte"
```

Test images database path

#### 4.1.1.9 TEST\_LABELS\_PATH

```
const string TEST_LABELS_PATH = "D:\\t10k-labels.idx1-ubyte"
```

Test labels database path

#### 4.1.1.10 TRAIN\_IMAGES\_PATH

```
const string TRAIN_IMAGES_PATH = "D:\\train-images.idx3-ubyte"
```

Training images database path

#### 4.1.1.11 TRAIN\_LABELS\_PATH

```
const string TRAIN_LABELS_PATH = "D:\\train-labels.idx1-ubyte"
```

Training labels database path

#### 4.1.1.12 TYI

```
const short TYI = 3
```

Length of short-term memory window in presynaptic phase

#### 4.1.1.13 TYO

```
const short TYO = 2
```

Length of short-term memory window in feedback phase

## 4.2 SNN.cpp File Reference

```
#include "stdafx.h"
#include "Network.h"
#include "Utils.h"
#include <time.h>
#include <opencv2/opencv.hpp>
#include "TrainingPool.h"
```

### Functions

- int [main](#) (int argc, char \*\*argv)

#### 4.2.1 Function Documentation

##### 4.2.1.1 main()

```
int main (
    int argc,
    char ** argv )
```

This is where your program should be written





# Index

- AddTrain
  - InputLayer, [7](#)
- ApplyAlphas
  - InputLayer, [7](#)
- basis
  - InputLayer, [8](#)
  - OutputLayer, [19](#)
- CLASSES
  - Constants.h, [25](#)
- ComputeErrors
  - OutputLayer, [17](#)
- ComputeOutput
  - OutputLayer, [17](#)
- ComputeWinner
  - OutputLayer, [18](#)
- Constants.h, [25](#)
  - CLASSES, [25](#)
  - Ka, [25](#)
  - Kb, [26](#)
  - LEARNING\_RATE, [26](#)
  - NEURONS\_IN, [26](#)
  - P\_RANGE, [26](#)
  - T, [26](#)
  - TEST\_IMAGES\_PATH, [26](#)
  - TEST\_LABELS\_PATH, [26](#)
  - TRAIN\_IMAGES\_PATH, [26](#)
  - TRAIN\_LABELS\_PATH, [27](#)
  - TYI, [27](#)
  - TYO, [27](#)
- Conv
  - MatrixOps, [9](#)
- CrossValidate
  - Network, [13](#)
- DatabaseOps, [5](#)
  - ExportData, [5](#)
  - ImportData, [6](#)
- Dot
  - MatrixOps, [9](#)
- ExportData
  - DatabaseOps, [5](#)
  - Network, [13](#)
- gammas
  - OutputLayer, [19](#)
- GenerateAlphaBasis
  - Utils, [22](#)
- GenerateBetaBasis
  - Utils, [22](#)
- GenerateSpikes
  - Utils, [22](#)
- GetTestData
  - Utils, [23](#)
- GetTrainingData
  - Utils, [23](#)
- ImportData
  - DatabaseOps, [6](#)
  - Network, [14](#)
- index
  - InputLayer, [8](#)
- InitTraining
  - TrainingPool, [20](#)
- InputLayer, [6](#)
  - AddTrain, [7](#)
  - ApplyAlphas, [7](#)
  - basis, [8](#)
  - index, [8](#)
  - InputLayer, [7](#)
  - ResetTrains, [7](#)
  - trains, [8](#)
  - UpdateAlphas, [8](#)
  - w, [8](#)
- inputLayer
  - Network, [16](#)
- Join
  - TrainingPool, [20](#)
- Ka
  - Constants.h, [25](#)
- Kb
  - Constants.h, [26](#)
- LEARNING\_RATE
  - Constants.h, [26](#)
- main
  - SNN.cpp, [27](#)
- MatrixOps, [9](#)
  - Conv, [9](#)
  - Dot, [9](#)
  - Multiply, [10](#)
  - Sum, [10](#)
  - SumArrays, [11](#)
  - SumColumns, [11](#)
  - Transpose, [12](#)
- Multiply
  - MatrixOps, [10](#)

- NEURONS\_IN
  - Constants.h, 26
- Network, 12
  - CrossValidate, 13
  - ExportData, 13
  - ImportData, 14
  - inputLayer, 16
  - Network, 13
  - outputLayer, 16
  - ResetNetwork, 14
  - Run, 14
  - Train, 14
  - Validate, 15
  - ValidateDataset, 15
- OutputLayer, 16
  - basis, 19
  - ComputeErrors, 17
  - ComputeOutput, 17
  - ComputeWinner, 18
  - gammas, 19
  - OutputLayer, 17
  - Reset, 18
  - u, 19
  - UpdateBetas, 18
  - UpdateGammas, 18
  - v, 19
  - y, 19
- outputLayer
  - Network, 16
- P\_RANGE
  - Constants.h, 26
- PrintLine
  - Utils, 23
- RateEncode
  - Utils, 24
- Reset
  - OutputLayer, 18
- ResetNetwork
  - Network, 14
- ResetTrains
  - InputLayer, 7
- Run
  - Network, 14
- runningInstances
  - TrainingPool, 21
- SNN.cpp, 27
  - main, 27
- Sum
  - MatrixOps, 10
- SumArrays
  - MatrixOps, 11
- SumColumns
  - MatrixOps, 11
- T
  - Constants.h, 26
- TEST\_IMAGES\_PATH
  - Constants.h, 26
- TEST\_LABELS\_PATH
  - Constants.h, 26
- TRAIN\_IMAGES\_PATH
  - Constants.h, 26
- TRAIN\_LABELS\_PATH
  - Constants.h, 27
- TYI
  - Constants.h, 27
- TYO
  - Constants.h, 27
- Train
  - Network, 14
- TrainAsync
  - TrainingPool, 20
- TrainingPool, 19
  - InitTraining, 20
  - Join, 20
  - runningInstances, 21
  - TrainAsync, 20
- trains
  - InputLayer, 8
- Transpose
  - MatrixOps, 12
- u
  - OutputLayer, 19
- UpdateAlphas
  - InputLayer, 8
- UpdateBetas
  - OutputLayer, 18
- UpdateGammas
  - OutputLayer, 18
- Utils, 21
  - GenerateAlphaBasis, 22
  - GenerateBetaBasis, 22
  - GenerateSpikes, 22
  - GetTestData, 23
  - GetTrainingData, 23
  - PrintLine, 23
  - RateEncode, 24
- v
  - OutputLayer, 19
- Validate
  - Network, 15
- ValidateDataset
  - Network, 15
- w
  - InputLayer, 8
- y
  - OutputLayer, 19

T