deep learning from scratch

Generated by Doxygen 1.8.16

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 malaria_detection Namespace Reference

Functions

- def read_images ()
- def train_test_set (files_df)
- def load_images (train_files, train_labels, val_files, test_files, mean, hdf5_file, IMG_DIMS)
- def discover_dataset (train_files)

Variables

- string datapath = '../Malaria_Dataset/'
- bool SUBTRACT_MEAN = False
- string path_train = "../Malaria_Dataset/cell_images_28/train/"
- string path_val = "../Malaria_Dataset/cell_images_28/validation/"
- string path_test = "../Malaria_Dataset/cell_images_28/test/"
- tuple IMG_DIM = (28, 28)
- hdf5_file = None
- string hdf5_datapath = '../Malaria_Dataset/data_64.hdf5'
- tuple train_shape = (len(train_files), 28, 28 , 3)
- tuple val_shape = (len(val_files), 28, 28, 3)
- tuple test_shape = (len(test_files), 28, 28, 3)
- le = LabelEncoder()
- train_labels_enc = le.transform(train_labels)
- val_labels_enc = le.transform(val_labels)
- test_labels_enc = le.transform(test_labels)
- mean = np.zeros(train_shape[1:], np.float32)
- figsize
- int n = 0
- r = np.random.randint(0 , hdf5_file["train_img"].shape[0] , 1)
- hspace
- wspace
- int BATCH_SIZE = 10
- int EPOCH_SIZE = 5
- data_num = hdf5_file["train_img"].shape[0]
- validation_set_num = hdf5_file["val_img"].shape[0]

- batches_list = list(range(int(ceil(float(data_num) / BATCH_SIZE))))
- val_batches_list = list(range(int(ceil(float(validation_set_num) / BATCH_SIZE))))
- net = dl.Network()
- output_size_conv1
- num_of_outputs_conv1
- output_size_relu1_h
- output_size_relu1_w
- num_of_outputs_relu1
- output_size_conv2
- num_of_outputs_conv2
- output_size_relu2_h
- output_size_relu2_w
- num_of_outputs_relu2
- output_size_pooling2
- num_of_outputs_pooling_2
- output_size_conv3
- num_of_outputs_conv3
- output_size_relu3_h
- output_size_relu3_w
- num_of_outputs_relu3
- output_size_conv4
- num_of_outputs_conv4
- output_size_relu4_h
- output_size_relu4_w
- num_of_outputs_relu4
- output_size_pooling4
- num_of_outputs_pooling_4
- output_size_fully1_h
- output_size_fully1_w
- num_of_outputs_fully1
- output_size_relu5_h
- output_size_relu5_w
- num_of_outputs_relu5
- output_size_fully2_h
- output_size_fully2_w
- num_of_outputs_fully2
- output_size_sigmoid5_h
- output_size_sigmoid5_w
- num_of_outputs_sigmoid5
- int n_values = 2
- list train_cost = [ ]
- list train_cost10 = [ ]
- list iterations = [ ]
- list iterations10 = [ ]
- float train_acc = 0.0
- float val_acc = 0.0
- list epoch_axis = [ ]
- list train_acc_epoch = [ ]
- list val_acc_epoch = [ ]
- images = None
- int iter = 0
- int iter10 = 0
- float batch_10_cost = 0.0
- int i_s = i ∗ BATCH_SIZE
- i_e = min([(i + 1) ∗ BATCH_SIZE, data_num])

- labels = hdf5_file["train_labels"][i_s:i_e]
- labels_one_hot = np.eye(n_values)[labels]
- batch_cost
- batch_accuracy = net.validation(images / 255., labels_one_hot, BATCH_SIZE)
- label
- loc

5.1.1 Function Documentation

5.1.1.1 discover_dataset()

```
def malaria_detection.discover_dataset (
                train_files )
```

Definition at line 113 of file malaria_detection.py.

```
113 def discover_dataset(train_files):
114     shapes = []
115     for file_path in train_files:
116         shapes.append(cv2.imread(file_path).shape)
117     return list(shapes)
118
119
```

5.1.1.2 load_images()

```
def malaria_detection.load_images (
                train_files,
                train_labels,
                val_files,
                test_files,
                mean,
                hdf5_file,
                IMG_DIMS )
```

Definition at line 59 of file malaria_detection.py.

```
59 def load_images(train_files, train_labels, val_files,  test_files, mean, hdf5_file, IMG_DIMS):
60     num = 0
61     # loop over train addresses
62     for i in range(len(train_files)):
63         # print how many images are saved every 1000 images
64         if i % 5000 == 0 and i > 1:
65             print('Train data: {}/{}'.format(i, len(train_files)))
66         addr = train_files[i]
67         img = cv2.imread(addr)
68         img = cv2.resize(img, dsize=IMG_DIMS,
69                          interpolation=cv2.INTER_CUBIC)
70         img = np.array(img, dtype=np.float32)
71
72         if train_labels[i] == "malaria":
73             cv2.imwrite(path_train + str(1) + "/" +str(num)+".png", img)
74         else:
75             cv2.imwrite(path_train+ str(0) + "/" +str(num)+".png", img)
76         num += 1
77         hdf5_file["train_img"][i, ...] = img
78         mean += img / float(len(train_labels))
79
80     # loop over validation addresses
81     for i in range(len(val_files)):
82         # print how many images are saved every 1000 images
83         if i % 5000 == 0 and i > 1:
84             print('Validation data: {}/{}'.format(i, len(val_files)))
85         addr = val_files[i]
86         img = cv2.imread(addr)
```

```
87          img = cv2.resize(img, dsize=IMG_DIMS,
88                           interpolation=cv2.INTER_CUBIC)
89          img = np.array(img, dtype=np.float32)
90
91          if train_labels[i] == "malaria":
92              cv2.imwrite(path_val + str(1) + "/" +str(num)+".png", img)
93          else:
94              cv2.imwrite(path_val+ str(0) + "/" +str(num)+".png", img)
95          num += 1
96          hdf5_file["val_img"][i, ...] = img
97
98      # loop over test addresses
99      for i in range(len(test_files)):
100          # print how many images are saved every 1000 images
101          if i % 5000 == 0 and i > 1:
102              print('Test data: {}/{}'.format(i, len(test_files)))
103          # read an image and resize to (224, 224)
104          # cv2 load images as BGR, convert it to RGB
105          addr = test_files[i]
106          img = cv2.imread(addr)
107          img = cv2.resize(img, dsize=IMG_DIMS,
108                           interpolation=cv2.INTER_CUBIC)
109          img = np.array(img, dtype=np.float32)
110          hdf5_file["test_img"][i, ...] = img
111
112
```

5.1.1.3 read_images()

```
def malaria_detection.read_images ( )
```

Definition at line 25 of file malaria_detection.py.

```
25 def read_images():
26     base_dir = os.path.join('../cell_images')
27     infected_dir = os.path.join(base_dir,'Parasitized')
28     healthy_dir = os.path.join(base_dir,'Uninfected')
29
30     infected_files = glob.glob(infected_dir+'/*.png')
31     healthy_files = glob.glob(healthy_dir+'/*.png')
32
33     np.random.seed(42)
34
35     files_df = pd.DataFrame({
36         'filename': infected_files + healthy_files,
37         'label': ['malaria'] * len(infected_files) + ['healthy'] * len(healthy_files)
38     }).sample(frac=1, random_state=42).reset_index(drop=True)
39     files_df.head()
40     return files_df
41
```

5.1.1.4 train_test_set()

```
def malaria_detection.train_test_set (
               files_df )
```

Definition at line 42 of file malaria_detection.py.

```
42 def train_test_set(files_df):
43     train_files, test_files, train_labels, test_labels = train_test_split(files_df['filename'].values,
44                                                                           files_df['label'].values,
45                                                                           test_size=0.3,
46                                                                           random_state=42)
47     train_files, val_files, train_labels, val_labels = train_test_split(train_files,
48                                                                         train_labels,
49                                                                         test_size=0.1, random_state=42)
50
51     return train_files, train_labels, val_files, val_labels, test_files, test_labels
52
53
```

5.1.2 Variable Documentation

5.1.2.1 batch_10_cost

```
float malaria_detection.batch_10_cost = 0.0
```

Definition at line 457 of file malaria_detection.py.

5.1.2.2 batch_accuracy

```
malaria_detection.batch_accuracy = net.validation(images / 255., labels_one_hot, BATCH_SIZE)
```

Definition at line 503 of file malaria_detection.py.

5.1.2.3 batch_cost

```
malaria_detection.batch_cost
```

Definition at line 474 of file malaria_detection.py.

5.1.2.4 BATCH_SIZE

```
int malaria_detection.BATCH_SIZE = 10
```

Definition at line 226 of file malaria_detection.py.

5.1.2.5 batches_list

```
malaria_detection.batches_list = list(range(int(ceil(float(data_num) / BATCH_SIZE))))
```

Definition at line 232 of file malaria_detection.py.

5.1.2.6 data_num

```
malaria_detection.data_num = hdf5_file["train_img"].shape[0]
```

Definition at line 228 of file malaria_detection.py.

5.1.2.7 datapath

```
string malaria_detection.datapath = '../Malaria_Dataset/'
```

Definition at line 19 of file malaria_detection.py.

5.1.2.8 epoch_axis

```
list malaria_detection.epoch_axis = []
```

Definition at line 445 of file malaria_detection.py.

5.1.2.9 EPOCH_SIZE

```
int malaria_detection.EPOCH_SIZE = 5
```

Definition at line 227 of file malaria_detection.py.

5.1.2.10 figsize

```
malaria_detection.figsize
```

Definition at line 203 of file malaria_detection.py.

5.1.2.11 hdf5_datapath

```
string malaria_detection.hdf5_datapath = '../Malaria_Dataset/data_64.hdf5'
```

Definition at line 152 of file malaria_detection.py.

5.1.2.12 hdf5_file

```
malaria_detection.hdf5_file = None
```

Definition at line 151 of file malaria_detection.py.

5.1.2.13 hspace

```
malaria_detection.hspace
```

Definition at line 209 of file malaria_detection.py.

5.1.2.14 i_e

```
malaria_detection.i_e = min([(i + 1) * BATCH_SIZE, data_num])
```

Definition at line 461 of file malaria_detection.py.

5.1.2.15 i_s

```
int malaria_detection.i_s = i * BATCH_SIZE
```

Definition at line 460 of file malaria_detection.py.

5.1.2.16 images

```
malaria_detection.images = None
```

Definition at line 449 of file malaria_detection.py.

5.1.2.17 IMG_DIM

```
tuple malaria_detection.IMG_DIM = (28, 28)
```

Definition at line 150 of file malaria_detection.py.

5.1.2.18 iter

```
int malaria_detection.iter = 0
```

Definition at line 451 of file malaria_detection.py.

5.1.2.19 iter10

```
int malaria_detection.iter10 = 0
```

Definition at line 452 of file malaria_detection.py.

5.1.2.20 iterations

```
list malaria_detection.iterations = []
```

Definition at line 440 of file malaria_detection.py.

5.1.2.21 iterations10

```
list malaria_detection.iterations10 = []
```

Definition at line 441 of file malaria_detection.py.

5.1.2.22 label

```
malaria_detection.label
```

Definition at line 549 of file malaria_detection.py.

5.1.2.23 labels

```
malaria_detection.labels = hdf5_file["train_labels"][i_s:i_e]
```

Definition at line 468 of file malaria_detection.py.

5.1.2.24 labels_one_hot

```
malaria_detection.labels_one_hot = np.eye(n_values)[labels]
```

Definition at line 469 of file malaria_detection.py.

5.1.2.25 le

```
malaria_detection.le = LabelEncoder()
```

Definition at line 161 of file malaria_detection.py.

5.1.2.26 loc

```
malaria_detection.loc
```

Definition at line 554 of file malaria_detection.py.

5.1.2.27 mean

```
malaria_detection.mean = np.zeros(train_shape[1:], np.float32)
```

Definition at line 182 of file malaria_detection.py.

5.1.2.28 n

```
int malaria_detection.n = 0
```

Definition at line 204 of file malaria_detection.py.

5.1.2.29 n_values

```
int malaria_detection.n_values = 2
```

Definition at line 437 of file malaria_detection.py.

5.1.2.30 net

```
malaria_detection.net = dl.Network()
```

Definition at line 240 of file malaria_detection.py.

5.1.2.31 num_of_outputs_conv1

```
malaria_detection.num_of_outputs_conv1
```

Definition at line 321 of file malaria_detection.py.

5.1.2.32 num_of_outputs_conv2

```
malaria_detection.num_of_outputs_conv2
```

Definition at line 336 of file malaria_detection.py.

5.1.2.33 num_of_outputs_conv3

```
malaria_detection.num_of_outputs_conv3
```

Definition at line 363 of file malaria_detection.py.

5.1.2.34 num_of_outputs_conv4

```
malaria_detection.num_of_outputs_conv4
```

Definition at line 380 of file malaria_detection.py.

5.1.2.35 num_of_outputs_fully1

```
malaria_detection.num_of_outputs_fully1
```

Definition at line 407 of file malaria_detection.py.

5.1.2.36 num_of_outputs_fully2

```
malaria_detection.num_of_outputs_fully2
```

Definition at line 420 of file malaria_detection.py.

5.1.2.37 num_of_outputs_pooling_2

```
malaria_detection.num_of_outputs_pooling_2
```

Definition at line 354 of file malaria_detection.py.

5.1.2.38 num_of_outputs_pooling_4

```
malaria_detection.num_of_outputs_pooling_4
```

Definition at line 398 of file malaria_detection.py.

5.1.2.39 num_of_outputs_relu1

```
malaria_detection.num_of_outputs_relu1
```

Definition at line 330 of file malaria_detection.py.

5.1.2.40 num_of_outputs_relu2

```
malaria_detection.num_of_outputs_relu2
```

Definition at line 347 of file malaria_detection.py.

5.1.2.41 num_of_outputs_relu3

```
malaria_detection.num_of_outputs_relu3
```

Definition at line 373 of file malaria_detection.py.

5.1.2.42 num_of_outputs_relu4

```
malaria_detection.num_of_outputs_relu4
```

Definition at line 391 of file malaria_detection.py.

5.1.2.43 num_of_outputs_relu5

```
malaria_detection.num_of_outputs_relu5
```

Definition at line 414 of file malaria_detection.py.

5.1.2.44 num_of_outputs_sigmoid5

```
malaria_detection.num_of_outputs_sigmoid5
```

Definition at line 427 of file malaria_detection.py.

5.1.2.45 output_size_conv1

```
malaria_detection.output_size_conv1
```

Definition at line 321 of file malaria_detection.py.

5.1.2.46 output_size_conv2

```
malaria_detection.output_size_conv2
```

Definition at line 336 of file malaria_detection.py.

5.1.2.47 output_size_conv3

```
malaria_detection.output_size_conv3
```

Definition at line 363 of file malaria_detection.py.

5.1.2.48 output_size_conv4

```
malaria_detection.output_size_conv4
```

Definition at line 380 of file malaria_detection.py.

5.1.2.49 output_size_fully1_h

```
malaria_detection.output_size_fully1_h
```

Definition at line 407 of file malaria_detection.py.

5.1.2.50 output_size_fully1_w

```
malaria_detection.output_size_fully1_w
```

Definition at line 407 of file malaria_detection.py.

5.1.2.51 output_size_fully2_h

```
malaria_detection.output_size_fully2_h
```

Definition at line 420 of file malaria_detection.py.

5.1.2.52 output_size_fully2_w

```
malaria_detection.output_size_fully2_w
```

Definition at line 420 of file malaria_detection.py.

5.1.2.53 output_size_pooling2

```
malaria_detection.output_size_pooling2
```

Definition at line 354 of file malaria_detection.py.

5.1.2.54 output_size_pooling4

```
malaria_detection.output_size_pooling4
```

Definition at line 398 of file malaria_detection.py.

5.1.2.55 output_size_relu1_h

```
malaria_detection.output_size_relu1_h
```

Definition at line 330 of file malaria_detection.py.

5.1.2.56 output_size_relu1_w

```
malaria_detection.output_size_relu1_w
```

Definition at line 330 of file malaria_detection.py.

5.1.2.57 output_size_relu2_h

malaria_detection.output_size_relu2_h

Definition at line 347 of file malaria_detection.py.

5.1.2.58 output_size_relu2_w

malaria_detection.output_size_relu2_w

Definition at line 347 of file malaria_detection.py.

5.1.2.59 output_size_relu3_h

malaria_detection.output_size_relu3_h

Definition at line 373 of file malaria_detection.py.

5.1.2.60 output_size_relu3_w

malaria_detection.output_size_relu3_w

Definition at line 373 of file malaria_detection.py.

5.1.2.61 output_size_relu4_h

malaria_detection.output_size_relu4_h

Definition at line 391 of file malaria_detection.py.

5.1.2.62 output_size_relu4_w

malaria_detection.output_size_relu4_w

Definition at line 391 of file malaria_detection.py.

5.1.2.63 output_size_relu5_h

malaria_detection.output_size_relu5_h

Definition at line 414 of file malaria_detection.py.

5.1.2.64 output_size_relu5_w

```
malaria_detection.output_size_relu5_w
```

Definition at line 414 of file malaria_detection.py.

5.1.2.65 output_size_sigmoid5_h

```
malaria_detection.output_size_sigmoid5_h
```

Definition at line 427 of file malaria_detection.py.

5.1.2.66 output_size_sigmoid5_w

```
malaria_detection.output_size_sigmoid5_w
```

Definition at line 427 of file malaria_detection.py.

5.1.2.67 path_test

```
string malaria_detection.path_test = "../Malaria_Dataset/cell_images_28/test/"
```

Definition at line 56 of file malaria_detection.py.

5.1.2.68 path_train

```
string malaria_detection.path_train = "../Malaria_Dataset/cell_images_28/train/"
```

Definition at line 54 of file malaria_detection.py.

5.1.2.69 path_val

```
string malaria_detection.path_val = "../Malaria_Dataset/cell_images_28/validation/"
```

Definition at line 55 of file malaria_detection.py.

5.1.2.70 r

```
malaria_detection.r = np.random.randint(0 , hdf5_file["train_img"].shape[0] , 1)
```

Definition at line 207 of file malaria_detection.py.

5.1.2.71 SUBTRACT_MEAN

```
bool malaria_detection.SUBTRACT_MEAN = False
```

Definition at line 21 of file malaria_detection.py.

5.1.2.72 test_labels_enc

```
malaria_detection.test_labels_enc = le.transform(test_labels)
```

Definition at line 165 of file malaria_detection.py.

5.1.2.73 test_shape

```
tuple malaria_detection.test_shape = (len(test_files), 28, 28, 3)
```

Definition at line 157 of file malaria_detection.py.

5.1.2.74 train_acc

```
float malaria_detection.train_acc = 0.0
```

Definition at line 442 of file malaria_detection.py.

5.1.2.75 train_acc_epoch

```
list malaria_detection.train_acc_epoch = []
```

Definition at line 446 of file malaria_detection.py.

5.1.2.76 train_cost

```
list malaria_detection.train_cost = []
```

Definition at line 438 of file malaria_detection.py.

5.1.2.77 train_cost10

```
list malaria_detection.train_cost10 = []
```

Definition at line 439 of file malaria_detection.py.

5.1.2.78 train_labels_enc

```
malaria_detection.train_labels_enc = le.transform(train_labels)
```

Definition at line 163 of file malaria_detection.py.

5.1.2.79 train_shape

```
tuple malaria_detection.train_shape = (len(train_files), 28, 28 , 3)
```

Definition at line 155 of file malaria_detection.py.

5.1.2.80 val_acc

```
float malaria_detection.val_acc = 0.0
```

Definition at line 443 of file malaria_detection.py.

5.1.2.81 val_acc_epoch

```
list malaria_detection.val_acc_epoch = []
```

Definition at line 447 of file malaria_detection.py.

5.1.2.82 val_batches_list

```
malaria_detection.val_batches_list = list(range(int(ceil(float(validation_set_num) / BATCH_SIZE))))
```

Definition at line 236 of file malaria_detection.py.

5.1.2.83 val_labels_enc

```
malaria_detection.val_labels_enc = le.transform(val_labels)
```

Definition at line 164 of file malaria_detection.py.

5.1.2.84 val_shape

```
tuple malaria_detection.val_shape = (len(val_files), 28, 28, 3)
```

Definition at line 156 of file malaria_detection.py.

5.1.2.85 validation_set_num

```
malaria_detection.validation_set_num = hdf5_file["val_img"].shape[0]
```

Definition at line 229 of file malaria_detection.py.

5.1.2.86 wspace

```
malaria_detection.wspace
```

Definition at line 209 of file malaria_detection.py.

5.2 setup Namespace Reference

Variables

- list cpp_args = ['-std=c++11']
- list ext_modules
- name
- version
- author
- author_email
- description

5.2.1 Variable Documentation

5.2.1.1 author

```
setup.author
```

Definition at line 22 of file setup.py.

5.2.1.2 author_email

```
setup.author_email
```

Definition at line 23 of file setup.py.

5.2.1.3 cpp_args

```
list setup.cpp_args = ['-std=c++11']
```

Definition at line 6 of file setup.py.

5.2.1.4 description

```
setup.description
```

Definition at line 24 of file setup.py.

5.2.1.5 ext_modules

```
setup.ext_modules
```

Initial value:

```
1 =  [
2     Extension(
3         'dl',
4         ['Activation.cpp', 'AddNode.cpp', 'ComputationalNode.cpp', 'Connector.cpp', 'Convolution.cpp',
       'DotProduct.cpp', 'Maxpooling.cpp',
5          'Network.cpp', 'Dropout.cpp', 'wrap.cpp'],
6         include_dirs=['../libs/include'],
7         language='c++',
8         extra_compile_args = cpp_args,
9     ),
10 ]
```

Definition at line 8 of file setup.py.

5.2.1.6 name

```
setup.name
```

Definition at line 20 of file setup.py.

5.2.1.7 version

```
setup.version
```

Definition at line 21 of file setup.py.

Chapter 6

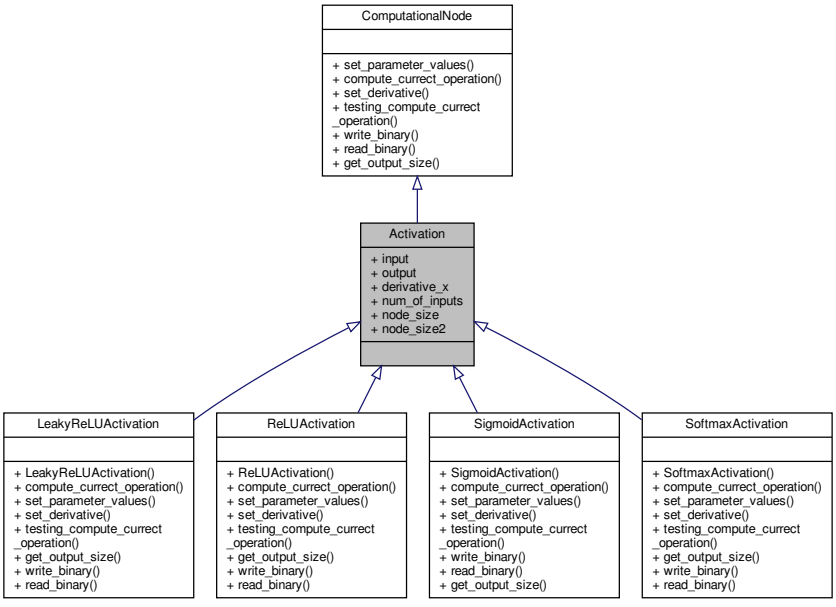
Class Documentation

6.1 Activation Class Reference

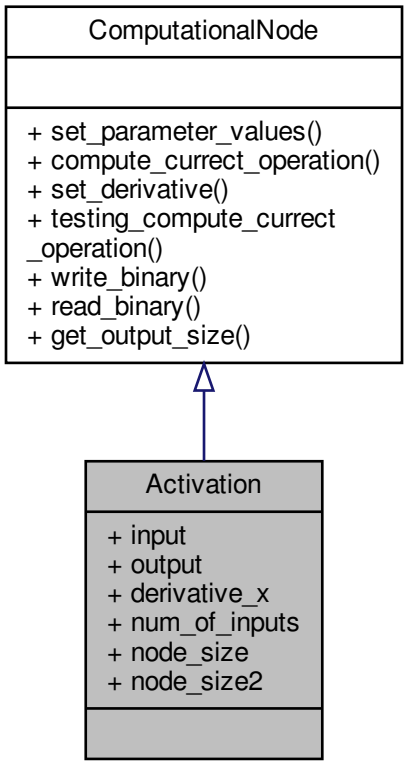
An abstract class, which is the parent of all the activation functions.

`#include <Activation.h>`

Inheritance diagram for Activation:



Collaboration diagram for Activation:



Public Attributes

- std::vector< Eigen::MatrixXd > input

  operation.
- std::vector< Eigen::MatrixXd > output

  operation.
- std::vector< Eigen::MatrixXd > derivative_x

  activation with respect to the input.
- int num_of_inputs

  number of inputs of the activation function
- int node_size

  the height of the input layer
- int node_size2

  the width of the input layer

Additional Inherited Members

6.1.1 Detailed Description

An abstract class, which is the parent of all the activation functions.

This class contains the input, output, and the derivative with respect to input of the activation function. It does not implements the override functions, it only passes to the child classes for the implementation.

Definition at line 16 of file Activation.h.

6.1.2 Member Data Documentation

6.1.2.1 derivative_x

```
std::vector<Eigen::MatrixXd> Activation::derivative_x
```

activation with respect to the input.

a vector of matrices, which is for keeping the derivative of the

Definition at line 22 of file Activation.h.

6.1.2.2 input

```
std::vector<Eigen::MatrixXd> Activation::input
```

operation.

a vector of matrices, which is for the input value for the activation

Definition at line 18 of file Activation.h.

6.1.2.3 node_size

```
int Activation::node_size
```

the height of the input layer

Definition at line 26 of file Activation.h.

6.1.2.4 node_size2

```
int Activation::node_size2
```

the width of the input layer

Definition at line 27 of file Activation.h.

6.1.2.5 num_of_inputs

```
int Activation::num_of_inputs
```

number of inputs of the activation function

Definition at line 25 of file Activation.h.

6.1.2.6 output

```
std::vector<Eigen::MatrixXd> Activation::output
```

operation.

a vector of matrices, which is for the output value for the activation

Definition at line 20 of file Activation.h.

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

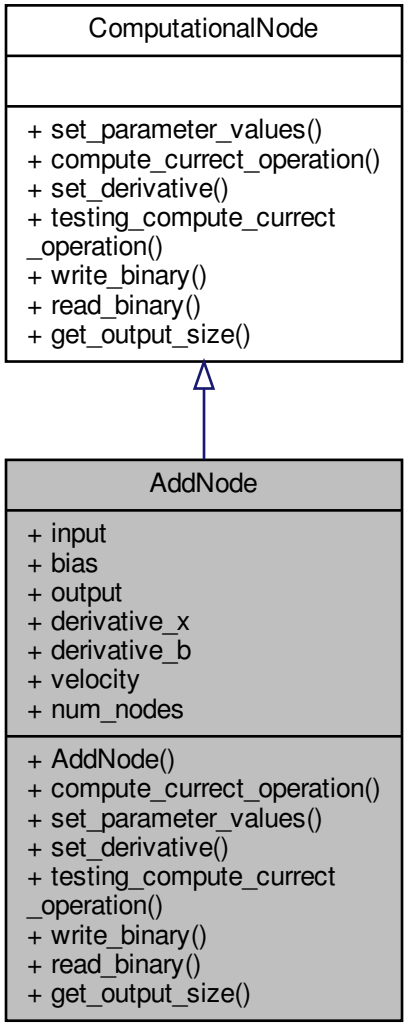
- /home/nehil/dlfsC++/libdl/headers/Activation.h

6.2 AddNode Class Reference

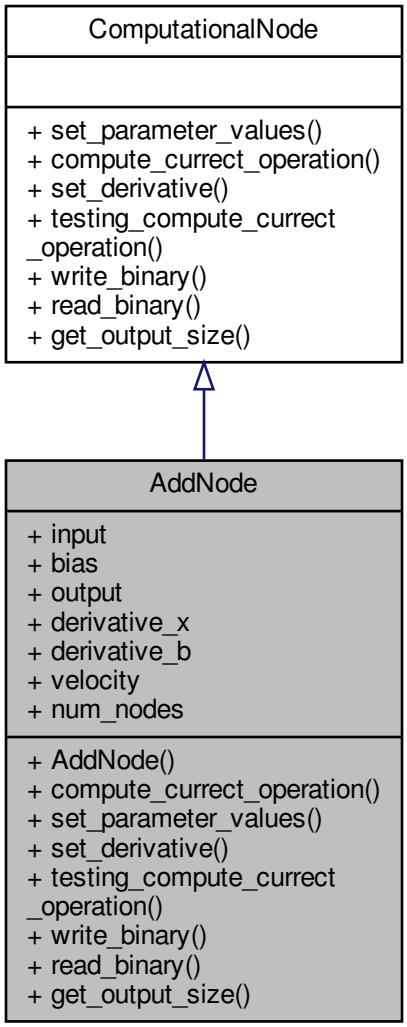
A class which implements the add bias operation in the network. This only called in the fully connected layers.

`#include <AddNode.h>`

Inheritance diagram for AddNode:



Collaboration diagram for AddNode:



Public Member Functions

- AddNode (int num_nodes)
- std::vector< Eigen::MatrixXd > compute_currect_operation (std::vector< Eigen::MatrixXd > tmp_input) override
- void set_parameter_values (double learning_rate, int batch_size) override
- std::vector< Eigen::MatrixXd > set_derivative (std::vector< Eigen::MatrixXd > prev_derivative) override
- std::vector< Eigen::MatrixXd > testing_compute_currect_operation (std::vector< Eigen::MatrixXd > tmp_↩ input) override
- void write_binary (std::ofstream &out) override
- void read_binary (std::ifstream &in) override
- std::array< int, 3 > get_output_size () override

Public Attributes

- std::vector< Eigen::MatrixXd > input

  the input of the bias addition operation.
- Eigen::MatrixXd bias

  the value for the bias to be added onto the given input.
- std::vector< Eigen::MatrixXd > output

  the output value of the bias addition operation.
- std::vector< Eigen::MatrixXd > derivative_x

  derivative of the bias addition operation wrt. the input value.
- Eigen::MatrixXd derivative_b

  derivative of the bias addition operation wrt. the bias value.
- Eigen::VectorXd velocity

  velocity value to use in the momentum optimization fuhnction.
- int num_nodes

6.2.1 Detailed Description

A class which implements the add bias operation in the network. This only called in the fully connected layers.

Definition at line 15 of file AddNode.h.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 AddNode()

```
AddNode::AddNode (
            int num_nodes )  [explicit]
```

A constructor for adding bias operation.

Parameters

num_nodes	an integer argument, represents the number of input neurons in the layer.

All the variables of the add bias computational node is initialized.

< the elements of bias initialized by the number which is generated from the normal distribution.

< Xaivier initialization used.

< velocity for the momentum gradient descent.

Definition at line 8 of file AddNode.cpp.

```
8                                {
12      this->num_nodes = num_nodes;
13      this->input.emplace_back(Eigen::MatrixXd::Zero(num_nodes,1));
14      this->bias = Eigen::MatrixXd::Ones(num_nodes, 1);
15      random_number_generator(this->bias, num_nodes);
16      this->bias = this->bias * (sqrt(2.0 / num_nodes));
17      this->output.emplace_back(Eigen::MatrixXd::Zero(num_nodes,1));
18      this->derivative_x.emplace_back(Eigen::MatrixXd::Ones(num_nodes,1));
```

```
19     this->derivative_b = Eigen::VectorXd::Ones(num_nodes);
20     this->velocity = Eigen::VectorXd::Zero(num_nodes);
21 }
```

6.2.3 Member Function Documentation

6.2.3.1 compute_currect_operation()

```
std::vector< Eigen::MatrixXd > AddNode::compute_currect_operation (
            std::vector< Eigen::MatrixXd > tmp_input )  [override], [virtual]
```

A function to calculate the bias addition on the coming input value.

Parameters

tmp_input	a vector of Eigen matrices, an input coming from the previous computational node.

Returns

a vector of Eigen matrices, returns the values of the input after adding the bias.

The bias addition into the incoming value is calculated.

Implements ComputationalNode.

Definition at line 39 of file AddNode.cpp.

```
39                                                                                  {
40
44     this->input = tmp_input;
45     this->output = tmp_input;
46     this->output[0] += this->bias;
47     return this->output;
48 }
```

6.2.3.2 get_output_size()

```
std::array<int, 3> AddNode::get_output_size ( )  [inline], [override], [virtual]
```

A function to return the output size of this computational node.

Returns

respectively, number of outputs, height of the output, and the width of the output.

Implements ComputationalNode.

Definition at line 84 of file AddNode.h.

```
84                                          { return std::array<int, 3> { 1, this->num_nodes,
85                                                                        1};}
```

6.2.3.3 read_binary()

```
void AddNode::read_binary (
            std::ifstream & in )  [override], [virtual]
```

A function to load the biases back into the network.

Parameters

in	an ifstream object which is from the file opened to load the biases.

< The size of the rows and the columns of the bias matrix.

Implements ComputationalNode.

Definition at line 84 of file AddNode.cpp.

```
84                                          {
85
86     typename Eigen::MatrixXd::Index rows=this->bias.rows(), cols=this->bias.cols();
87     in.read((char*) (&rows),sizeof(typename Eigen::MatrixXf::Index));
88     in.read((char*) (&cols),sizeof(typename Eigen::MatrixXf::Index));
89     //matrix.resize(rows, cols);
90     in.read( (char *) this->bias.data() , rows*cols*sizeof(typename Eigen::MatrixXd::Scalar) );
91
92 }
```

6.2.3.4 set_derivative()

```
std::vector< Eigen::MatrixXd > AddNode::set_derivative (
             std::vector< Eigen::MatrixXd > prev_derivative )  [override], [virtual]
```

A function to find the derivative of add bias operation. The derivative is calculated with respect to both input and the bias.

Parameters

prev_derivative	a vector of Eigen matrices, down flow derivative through this add bias operation.

Returns

a vector of Eigen matrices, the multiplication of the current derivative of the operation and the input down flow derivative.

< The derviavtive with respect to the biases summed up, to be able to use them in mini batch calculation.

Implements ComputationalNode.

Definition at line 68 of file AddNode.cpp.

```
68                                                                                  {
69     this->derivative_x = prev_derivative;
70     this->derivative_b += prev_derivative[0];
71
72     return this->derivative_x;
73 }
```

6.2.3.5 set_parameter_values()

```
void AddNode::set_parameter_values (
             double learning_rate,
             int batch_size )  [override], [virtual]
```

A function to set the new bias value after the backward pass, after that this will set all the variables in this computational node to zero.

Parameters

learning_rate	a double argument, learning rate.
batch_size	an integer argument, the number of samples in one batch.

< Momentum gradient descent is applied.

Implements ComputationalNode.

Definition at line 57 of file AddNode.cpp.

```
57                                                                          {
58     //this->velocity = 0.9 * velocity + this->derivative_b / float(batch_size); //!< Weighted mean of the
       gradient descent is calculated.
59     this->bias -= (learning_rate * this->derivative_b / float(batch_size));
60     this->derivative_x[0].setOnes();
61     this->derivative_b.setOnes();
62     this->input[0].setZero();
63     this->output[0].setZero();
64
65 }
```

6.2.3.6 testing_compute_currect_operation()

```
std::vector< Eigen::MatrixXd > AddNode::testing_compute_currect_operation (
             std::vector< Eigen::MatrixXd > tmp_input )  [override], [virtual]
```

A function to run the forward pass of the add node operation during the testing phase.

Parameters

tmp_input	a vector of Eigen matrices, an input coming from the previous computational node.

Returns

a vector of Eigen matrices, returns the values of the input after adding the bias.

Implements ComputationalNode.

Definition at line 50 of file AddNode.cpp.

```
50                                                                                                          {
51     this->input = tmp_input;
52     this->output = tmp_input;
53     this->output[0] += this->bias;
54     return this->output;
55 }
```

6.2.3.7 write_binary()

```
void AddNode::write_binary (
             std::ofstream & out )  [override], [virtual]
```

A function to save the biases to a file.

Parameters

out	an ofstream object which is from the file opened to save the biases.

< The size of the rows and the columns of the bias matrix.

Implements ComputationalNode.

Definition at line 76 of file AddNode.cpp.

```
76                                          {
77
78     typename Eigen::MatrixXd::Index rows=this->bias.rows(), cols=this->bias.cols();
79     out.write((char*) (&rows), sizeof(typename Eigen::MatrixXd::Index));
80     out.write((char*) (&cols), sizeof(typename Eigen::MatrixXd::Index));
81     out.write((char*) this->bias.data(), rows*cols*sizeof(typename Eigen::MatrixXd::Scalar) );
82 }
```

6.2.4 Member Data Documentation

6.2.4.1 bias

```
Eigen::MatrixXd AddNode::bias
```

the value for the bias to be added onto the given input.

Definition at line 19 of file AddNode.h.

6.2.4.2 derivative_b

```
Eigen::MatrixXd AddNode::derivative_b
```

derivative of the bias addition operation wrt. the bias value.

Definition at line 24 of file AddNode.h.

6.2.4.3 derivative_x

```
std::vector<Eigen::MatrixXd> AddNode::derivative_x
```

derivative of the bias addition operation wrt. the input value.

Definition at line 22 of file AddNode.h.

6.2.4.4 input

```
std::vector<Eigen::MatrixXd> AddNode::input
```

the input of the bias addition operation.

Definition at line 17 of file AddNode.h.

6.2.4.5 num_nodes

```
int AddNode::num_nodes
```

Definition at line 26 of file AddNode.h.

6.2.4.6 output

```
std::vector<Eigen::MatrixXd> AddNode::output
```

the output value of the bias addition operation.

Definition at line 21 of file AddNode.h.

6.2.4.7 velocity

```
Eigen::VectorXd AddNode::velocity
```

velocity value to use in the momentum optimization fuhnction.

Definition at line 25 of file AddNode.h.

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

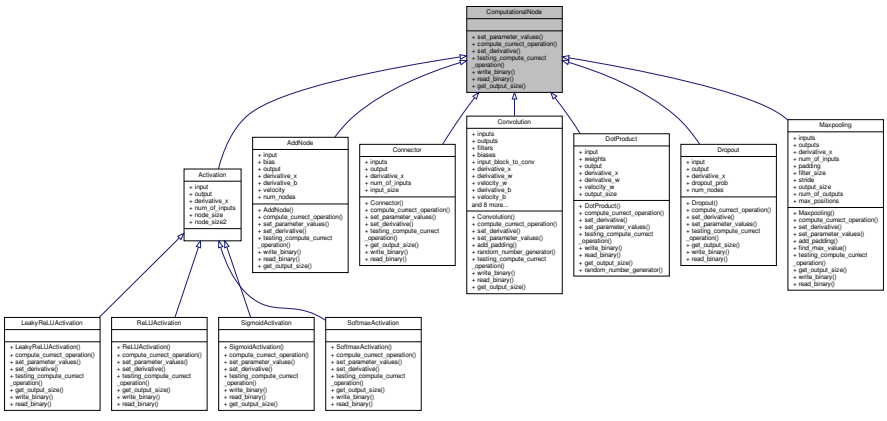
- /home/nehil/dlfsC++/libdl/headers/AddNode.h
- /home/nehil/dlfsC++/libdl/src/AddNode.cpp

6.3 ComputationalNode Class Reference

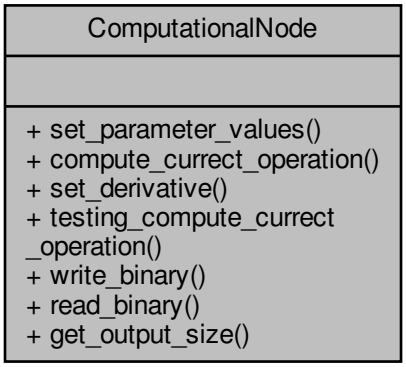
An abstract class, which is the parent of all the computation operations in the network.

```
#include <ComputationalNode.h>
```

Inheritance diagram for ComputationalNode:



Collaboration diagram for ComputationalNode:



Public Member Functions

- virtual void set_parameter_values (double learning_rate, int batch_size)=0
- virtual std::vector< Eigen::MatrixXd > compute_currect_operation (std::vector< Eigen::MatrixXd > input)=0
- virtual std::vector< Eigen::MatrixXd > set_derivative (std::vector< Eigen::MatrixXd > prev_derivative)=0
- virtual std::vector< Eigen::MatrixXd > testing_compute_currect_operation (std::vector< Eigen::MatrixXd > tmp_input)=0
- virtual void write_binary (std::ofstream &out)=0
- virtual void read_binary (std::ifstream &in)=0
- virtual std::array< int, 3 > get_output_size ()=0

6.3.1 Detailed Description

An abstract class, which is the parent of all the computation operations in the network.

Here in this class three functions are defined but not implemented. The functions are for the forward pass, backward pass and the updates in each computational node.

Definition at line 22 of file ComputationalNode.h.

6.3.2 Member Function Documentation

6.3.2.1 compute_currect_operation()

```
virtual std::vector<Eigen::MatrixXd> ComputationalNode::compute_currect_operation (
            std::vector< Eigen::MatrixXd > input )  [pure virtual]
```

A virtual function which will be override by the child classes. This function is for the computation of the forward pass of computational nodes in the network.

Parameters

input	a vector of eigen matrices, the input of the computational nodes.

Returns

the output of the forward pass of that computational node where the function is override.

Implemented in SoftmaxActivation, LeakyReLUActivation, ReLUActivation, Convolution, SigmoidActivation, DotProduct, AddNode, Connector, Dropout, and Maxpooling.

6.3.2.2 get_output_size()

```
virtual std::array<int, 3> ComputationalNode::get_output_size ( )  [pure virtual]
```

A function to return the output size of the current computational node.

Returns

respectively, number of outputs, height of the output, and the width of the output.

Implemented in SoftmaxActivation, LeakyReLUActivation, ReLUActivation, Convolution, DotProduct, Maxpooling, SigmoidActivation, AddNode, Connector, and Dropout.

6.3.2.3 read_binary()

```
virtual void ComputationalNode::read_binary (
            std::ifstream & in )  [pure virtual]
```

A virtual function which will be override by the child classes.

- This function is for loading the weights and the biases before the training of the network starts.

Parameters

in	an ifstream object

Implemented in SoftmaxActivation, LeakyReLUActivation, ReLUActivation, Convolution, Maxpooling, DotProduct, SigmoidActivation, AddNode, Connector, and Dropout.

6.3.2.4 set_derivative()

```
virtual std::vector<Eigen::MatrixXd> ComputationalNode::set_derivative (
            std::vector< Eigen::MatrixXd > prev_derivative )  [pure virtual]
```

A virtual function which will be override by the child classes. This function is for the computation of the backward pass of computational nodes in the network.

Parameters

prev_derivative	a vector of eigen matrices, down flowing gradient value.

Returns

the multiplication of the current gradient of the operational node where this function is override, and the down flowing gradient.

Implemented in SoftmaxActivation, LeakyReLUActivation, ReLUActivation, SigmoidActivation, Convolution, AddNode, Connector, Maxpooling, DotProduct, and Dropout.

6.3.2.5 set_parameter_values()

```
virtual void ComputationalNode::set_parameter_values (
            double learning_rate,
            int batch_size )  [pure virtual]
```

A virtual function which will be override by the child classes. This function is for the update of the trainable parameters(weights, and biases) of the network.

Parameters

learning_rate	a double argument, a step size for the descent algorithm.
batch_size	an integer argument, the number of samples in one batch.

Implemented in SoftmaxActivation, LeakyReLUActivation, ReLUActivation, Convolution, SigmoidActivation, Maxpooling, DotProduct, Dropout, AddNode, and Connector.

6.3.2.6 testing_compute_currect_operation()

```
virtual std::vector<Eigen::MatrixXd> ComputationalNode::testing_compute_currect_operation (
            std::vector< Eigen::MatrixXd > tmp_input )  [pure virtual]
```

A virtual function which will be override by the child classes. This function is for the computation of the forward pass of computational nodes in the network, in the testing and the validation phase.

Parameters

tmp_input	a vector of eigen matrices, the input of the computational nodes.

Returns

the output of the forward pass of that computational node where the function is override.

Implemented in SoftmaxActivation, LeakyReLUActivation, ReLUActivation, Convolution, Maxpooling, SigmoidActivation, DotProduct, AddNode, Connector, and Dropout.

6.3.2.7 write_binary()

```
virtual void ComputationalNode::write_binary (
            std::ofstream & out )  [pure virtual]
```

A virtual function which will be override by the child classes. This function is for saving the weights and the biases after the training of the network.

Parameters

out	an ofstream object

Implemented in SoftmaxActivation, LeakyReLUActivation, ReLUActivation, Convolution, Maxpooling, SigmoidActivation, DotProduct, Connector, Dropout, and AddNode.

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

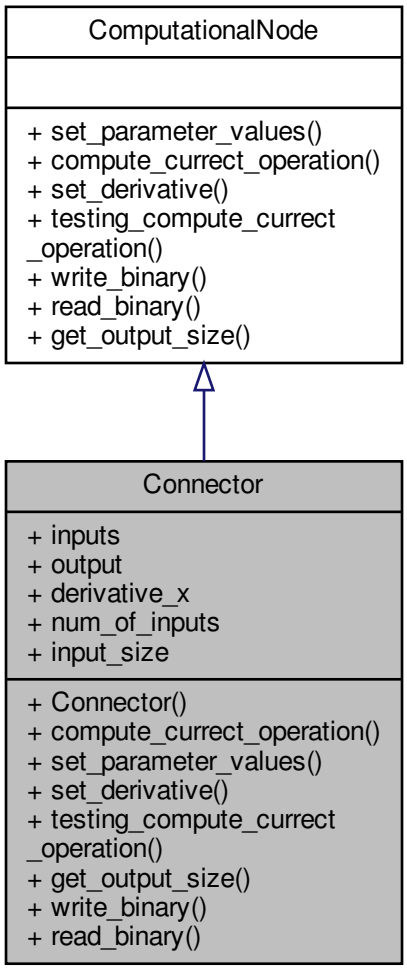
- /home/nehil/dlfsC++/libdl/headers/ComputationalNode.h

6.4 Connector Class Reference

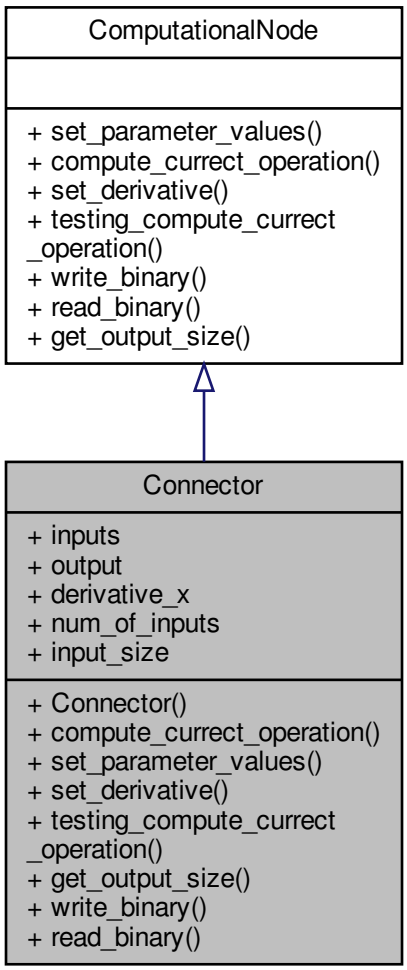
A class which refers to the operation for flattening the output out convolution or max pooling layers into a vector. In this way, the input will not be matrix anymore, and it will be suitable for the fully connected layers.

```
#include <Connector.h>
```

Inheritance diagram for Connector:



Collaboration diagram for Connector:



Public Member Functions

- Connector (int tmp_input_size, int tmp_num_of_inputs)
- std::vector< Eigen::MatrixXd > compute_currect_operation (std::vector< Eigen::MatrixXd > tmp_input) override
- void set_parameter_values (double learning_rate, int batch_size) override
- std::vector< Eigen::MatrixXd > set_derivative (std::vector< Eigen::MatrixXd > prev_derivative) override
- std::vector< Eigen::MatrixXd > testing_compute_currect_operation (std::vector< Eigen::MatrixXd > tmp_↩ input) override
- std::array< int, 3 > get_output_size () override
- void write_binary (std::ofstream &out) override
- void read_binary (std::ifstream &in) override

Public Attributes

- std::vector< Eigen::MatrixXd > inputs

  Feature maps from the convolutional layer.
- std::vector< Eigen::MatrixXd > output

  Flattened output of the input feature maps.
- std::vector< Eigen::MatrixXd > derivative_x

  Derivative with respect to the input feature maps.
- int num_of_inputs

  The number of the feature maps.
- int input_size

  The height and width of one feature map.

6.4.1 Detailed Description

A class which refers to the operation for flattening the output out convolution or max pooling layers into a vector. In this way, the input will not be matrix anymore, and it will be suitable for the fully connected layers.

Definition at line 15 of file Connector.h.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 Connector()

```
Connector::Connector (
            int tmp_input_size,
            int tmp_num_of_inputs )  [explicit]
```

The constructor for the connector computational node. It takes the number of feature maps and height or the width of one of the feature map.

Parameters

tmp_input_size	the height or the width of the input feature map.
tmp_num_of_inputs	the number of the input feature maps.

initialization of all the variables in the connector class.

Definition at line 7 of file Connector.cpp.

```
7                                                                   {
8
12      this->input_size = tmp_input_size;
13      this->num_of_inputs = tmp_num_of_inputs;
14      for (size_t i = 0; i < tmp_num_of_inputs; ++i) {
15          this->inputs.emplace_back(Eigen::MatrixXd::Zero(tmp_input_size, tmp_input_size));
16      }
17
18      this->output.emplace_back(Eigen::MatrixXd::Zero(tmp_input_size * tmp_input_size * tmp_num_of_inputs,
        1));
19
20      for (size_t i = 0; i < tmp_num_of_inputs; ++i) {
21          this->derivative_x.emplace_back(Eigen::MatrixXd::Ones(tmp_input_size, tmp_input_size));
22      }
23  }
```

6.4.3 Member Function Documentation

6.4.3.1 compute_currect_operation()

```
std::vector< Eigen::MatrixXd > Connector::compute_currect_operation (
            std::vector< Eigen::MatrixXd > tmp_input )  [override], [virtual]
```

A function to calculate the forward pass of the connector computational operation. The input consist of all the feature maps from the convolution layer. In this function these maps is put into one matrix which has the size of nx1.

Parameters

tmp_input	the feature maps passed from the convolution layers.

Returns

a matrix which consist all the input maps in it.

< looping through all the nput feature maps.

< a feature map is flattened.

< the result flattened vector is added

< to the correct location of the output vector.

Implements ComputationalNode.

Definition at line 26 of file Connector.cpp.

```
26                                                                                                                  {
27
28     this->inputs = tmp_input;
29     Eigen::MatrixXd transposed_input;
30     transposed_input = Eigen::MatrixXd::Zero(this->input_size, this->input_size);
31
32     int output_idx = 0;
33     for (size_t i = 0; i < this->num_of_inputs; ++i) {
34         transposed_input = this->inputs[i].transpose();
35         Eigen::Map<Eigen::MatrixXd> flattened_input(transposed_input.data(), this->input_size *
       this->input_size, 1);
36         this->output[0].block(output_idx, 0, this->input_size * this->input_size, 1) = flattened_input;
37         output_idx += this->input_size * this->input_size;
39     }
40
41     return this->output;
42 }
```

6.4.3.2 get_output_size()

```
std::array<int, 3> Connector::get_output_size ( )  [inline], [override], [virtual]
```

A function to return the output size of this computational node.

Returns

respectively, number of outputs, height of the output, and the width of the output.

Implements ComputationalNode.

Definition at line 70 of file Connector.h.

```
70                                              { return std::array<int, 3> { 1,
71                                                                            this->num_of_inputs *
       this->input_size *
72                                                                            this->input_size,
73                                                                            1};}
```

6.4.3.3 read_binary()

```
void Connector::read_binary (
            std::ifstream & in )  [inline], [override], [virtual]
```

A virtual function which will be override by the child classes.

- This function is for loading the weights and the biases before the training of the network starts.

Parameters

in	an ifstream object

Implements ComputationalNode.

Definition at line 76 of file Connector.h.

```
76 {};
```

6.4.3.4 set_derivative()

```
std::vector< Eigen::MatrixXd > Connector::set_derivative (
            std::vector< Eigen::MatrixXd > prev_derivative )  [override], [virtual]
```

A function to calculate the backward pass of the connector computational node.

Parameters

prev_derivative	a down flowing derivative value.

Returns

multiplication of the down flowin deriative with the current derivative of the connector operation with respect to the input value.

< The incoming derivative vector turned it into a matrix again.

Implements ComputationalNode.

Definition at line 68 of file Connector.cpp.

```
68                                                                                                  {
69
70      for(size_t i = 0; i < this->num_of_inputs; i++) {
71          this->derivative_x[i].setZero();
72      }
73
74      int output_idx = 0;
75      for (size_t i = 0; i < this->num_of_inputs; ++i) {
76          Eigen::Map<Eigen::MatrixXd> back_in_mat(
77                  prev_derivative[0].block(output_idx, 0, this->input_size * this->input_size, 1).data(),
78                  this->input_size, this->input_size);
79          this->derivative_x[i] = back_in_mat.transpose();
80          output_idx += this->input_size * this->input_size;
81      }
82
83      return this->derivative_x;
84  }
```

6.4.3.5 set_parameter_values()

```
void Connector::set_parameter_values (
            double learning_rate,
            int batch_size )  [override], [virtual]
```

A function to set all the variables of the connector class back to the intial values.

Parameters

learning_rate	
batch_size	

Implements ComputationalNode.

Definition at line 60 of file Connector.cpp.

```
60                                                                                  {
61     this->output[0].setZero();
62     for(size_t i = 0; i < this->num_of_inputs; i++) {
63         this->inputs[i].setZero();
64         this->derivative_x[i].setZero();
65     }
66 }
```

6.4.3.6 testing_compute_currect_operation()

```
std::vector< Eigen::MatrixXd > Connector::testing_compute_currect_operation (
            std::vector< Eigen::MatrixXd > tmp_input )  [override], [virtual]
```

A function to run the forward pass of the flattening operation of the convolution output feature maps during the testing phase.

Parameters

tmp_input	a vector of Eigen matrices, an input coming from the previous computational node.

Returns

a vector of Eigen matrices, returns the values of the input after the dot product operation.

Implements ComputationalNode.

Definition at line 44 of file Connector.cpp.

```
44                                                                                                                  {
45     this->inputs = tmp_input;
46     Eigen::MatrixXd transposed_input;
47     transposed_input = Eigen::MatrixXd::Zero(this->input_size, this->input_size);
48
49     int output_idx = 0;
50     for (size_t i = 0; i < this->num_of_inputs; ++i) {
51         transposed_input = this->inputs[i].transpose();
52         Eigen::Map<Eigen::MatrixXd> flattened_input(transposed_input.data(), this->input_size *
       this->input_size, 1);
53         this->output[0].block(output_idx, 0, this->input_size * this->input_size, 1) = flattened_input;
54         output_idx += this->input_size * this->input_size;
55     }
56
57     return this->output;
58 }
```

6.4.3.7 write_binary()

```
void Connector::write_binary (
            std::ofstream & out )  [inline], [override], [virtual]
```

A virtual function which will be override by the child classes. This function is for saving the weights and the biases after the training of the network.

Parameters

out	an ofstream object

Implements ComputationalNode.

Definition at line 75 of file Connector.h.
```
75 {};
```

6.4.4 Member Data Documentation

6.4.4.1 derivative_x

```
std::vector<Eigen::MatrixXd> Connector::derivative_x
```

Derivative with respect to the input feature maps.

Definition at line 19 of file Connector.h.

6.4.4.2 input_size

```
int Connector::input_size
```

The height and width of one feature map.

Definition at line 21 of file Connector.h.

6.4.4.3 inputs

```
std::vector<Eigen::MatrixXd> Connector::inputs
```

Feature maps from the convolutional layer.

Definition at line 17 of file Connector.h.

6.4.4.4 num_of_inputs

```
int Connector::num_of_inputs
```

The number of the feature maps.

Definition at line 20 of file Connector.h.

6.4.4.5 output

```
std::vector<Eigen::MatrixXd> Connector::output
```

Flattened output of the input feature maps.

Definition at line 18 of file Connector.h.

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

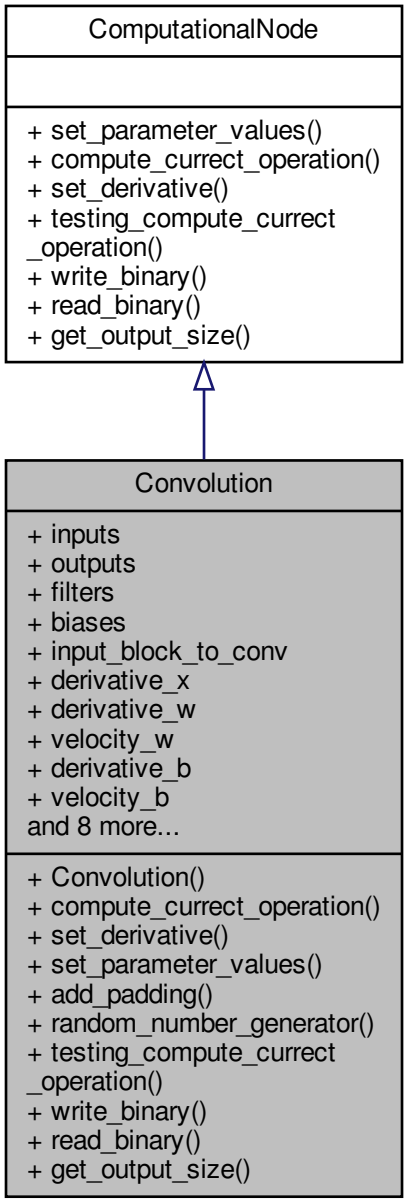
- /home/nehil/dlfsC++/libdl/headers/Connector.h
- /home/nehil/dlfsC++/libdl/src/Connector.cpp

6.5 Convolution Class Reference

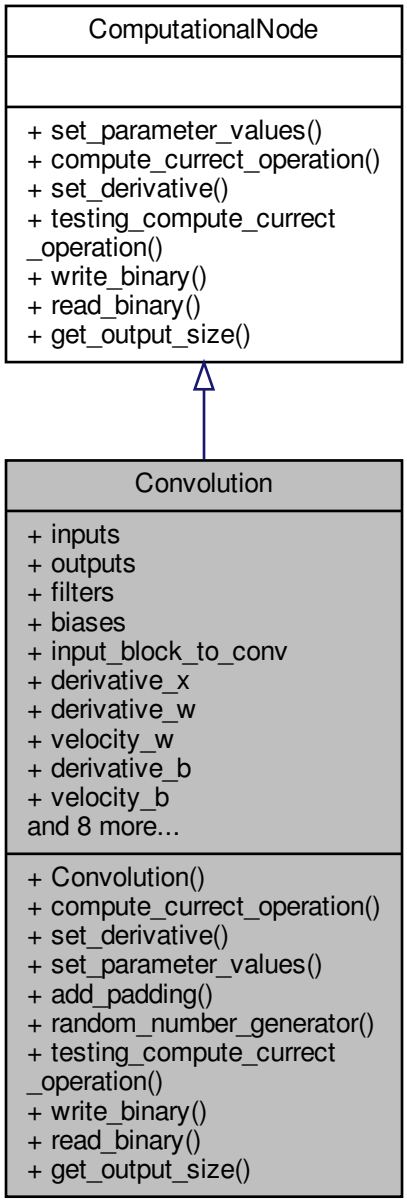
A class which implements the convolution operation in the network.

```
#include <Convolution.h>
```

Inheritance diagram for Convolution:



Collaboration diagram for Convolution:



Public Member Functions

- Convolution (int input_size, int tmp_input_length, int output_size, int tmp_num_of_outputs, int tmp_filter_size, int tmp_filter_size_1, int tmp_num_of_filters, int tmp_stride, int tmp_padding)
- std::vector< Eigen::MatrixXd > compute_currect_operation (std::vector< Eigen::MatrixXd > tmp_input) override
- std::vector< Eigen::MatrixXd > set_derivative (std::vector< Eigen::MatrixXd > prev_derivative) override
- void set_parameter_values (double learning_rate, int batch_size) override
- Eigen::MatrixXd add_padding (int index, const Eigen::MatrixXd &tmp_input)

- void random_number_generator (Eigen::MatrixXd &tmp_filter, int tmp_filter_size, int tmp_input_size)
- std::vector< Eigen::MatrixXd > testing_compute_currect_operation (std::vector< Eigen::MatrixXd > tmp_↩ input) override
- void write_binary (std::ofstream &out) override
- void read_binary (std::ifstream &in) override
- std::array< int, 3 > get_output_size () override

Public Attributes

- std::vector< Eigen::MatrixXd > inputs

  The input feature maps of the convolution computational node.
- std::vector< Eigen::MatrixXd > outputs

  The output of the convolution computation node.
- std::vector< std::vector< Eigen::MatrixXd > > filters

  operation. Since filters have a depth as well, they will be kept in vector of vector of matrices.
- std::vector< Eigen::MatrixXd > biases

  The biases to add to the each filter calculation.
- std::vector< std::vector< std::vector< Eigen::MatrixXd > > > input_block_to_conv

  backpropagation later, in the forward pass all the convolution pathes of input, kept in this vector.
- std::vector< Eigen::MatrixXd > derivative_x

  Derivative of the convolution process with respect to the input.
- std::vector< std::vector< Eigen::MatrixXd > > derivative_w

  convolution filters.
- std::vector< std::vector< Eigen::MatrixXd > > velocity_w

  Velocity to update the filters using the momentum.
- std::vector< Eigen::MatrixXd > derivative_b

  Derivative of convolution process with respect to the biases.
- std::vector< Eigen::MatrixXd > velocity_b

  Velocity to update the biases using the momentum.
- int output_size

  Output square channel size of the convolution node.
- int num_of_outputs

  the depth of the output.
- int padding

  Padding to use during the convolution process.
- int stride

  Stride to use during the convolution process.
- int filter_size

  Filter size.
- int filter_size_1

  filter depth
- int num_of_filters

  number of filters
- int num_of_inputs

  input depth

6.5.1 Detailed Description

A class which implements the convolution operation in the network.

Definition at line 13 of file Convolution.h.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 Convolution()

```
Convolution::Convolution (
            int input_size,
            int tmp_input_length,
            int output_size,
            int tmp_num_of_outputs,
            int tmp_filter_size,
            int tmp_filter_size_1,
            int tmp_num_of_filters,
            int tmp_stride,
            int tmp_padding )  [explicit]
```

A constructor to create a convolution layer.

Parameters

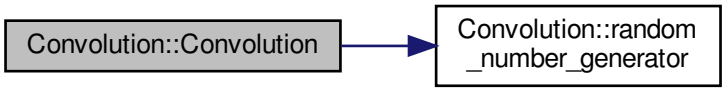
input_size	an integer argument, shows height of width of the input tensor.
tmp_input_length	an integer argument, input depth.
output_size	an integer argument, the height or width of the input tensor.
tmp_num_of_outputs	an integer argument, the depth of the output.
tmp_filter_size	an integer argument, the height or width of the filter.
tmp_filter_size_1	an integer argument, the depth of the filter.
tmp_num_of_filters	an integer argument, the number of filters.
tmp_stride	an integer argument, stride to use in convolution process.
tmp_padding	an integer argument, padding to use in the convolution process.

Definition at line 8 of file Convolution.cpp.

```
9
          {
10      this->output_size = tmp_output_size;
11      this->num_of_outputs = tmp_num_of_outputs;
12      this->filter_size = tmp_filter_size;
13      this->filter_size_1 = tmp_filter_size_1;
14      this->stride = tmp_stride;
15      this->padding = tmp_padding;
16      this->num_of_inputs = tmp_num_of_inputs;
17      this->num_of_filters = tmp_num_of_filters;
18
19      for (size_t i = 0; i < tmp_num_of_inputs; i++) {
20          this->inputs.emplace_back(Eigen::MatrixXd::Zero(input_size + 2 * tmp_padding, input_size + 2 *
        tmp_padding));
21          this->derivative_x.emplace_back(Eigen::MatrixXd::Zero(input_size + 2 * tmp_padding, input_size +
        2 * tmp_padding));
22      }
23
24
25      for (size_t i = 0; i < num_of_filters; i++) {
26          std::vector<Eigen::MatrixXd> filters_per_input;
27          std::vector<Eigen::MatrixXd> derivative_per_input_filters;
28          std::vector<Eigen::MatrixXd> velocity_per_input_filter;
```

```
29
30          for(size_t j = 0; j < tmp_filter_size_1; j++) {
31              filters_per_input.emplace_back(Eigen::MatrixXd::Zero(tmp_filter_size, tmp_filter_size));
32              random_number_generator(filters_per_input[j], tmp_filter_size, tmp_filter_size_1);
33              filters_per_input[j] = filters_per_input[j] * sqrt(2.0 / (tmp_filter_size * tmp_filter_size *
        tmp_filter_size_1));
34              derivative_per_input_filters.emplace_back(Eigen::MatrixXd::Zero(tmp_filter_size,
        tmp_filter_size));
35              velocity_per_input_filter.emplace_back(Eigen::MatrixXd::Zero(tmp_filter_size,
        tmp_filter_size));
36          }
37          this->filters.emplace_back(filters_per_input);
38          this->derivative_w.emplace_back(derivative_per_input_filters);
39          this->velocity_w.emplace_back(velocity_per_input_filter);
40      }
41
42
43
44      for (size_t i = 0; i < tmp_num_of_outputs; i++) {
45          Eigen::MatrixXd output;
46          output = Eigen::MatrixXd::Zero(this->output_size, this->output_size);
47          this->outputs.push_back(output);
48          Eigen::MatrixXd bias;
49          bias = Eigen::MatrixXd::Ones(this->output_size, this->output_size);
50          //bias = 0.1 * Eigen::MatrixXd::Ones(this->output_size, this->output_size);
51          random_number_generator(bias, this->output_size, this->output_size);
52          bias = bias * sqrt(2.0 / (this->output_size * this->output_size));
53          // TODO RANDOM
54          this->biases.emplace_back(bias);
55          this->velocity_b.emplace_back(Eigen::MatrixXd::Zero(this->output_size, this->output_size));
56          this->derivative_b.emplace_back(Eigen::MatrixXd::Ones(this->output_size, this->output_size));
57      }
58
59  }
```

Here is the call graph for this function:



6.5.3 Member Function Documentation

6.5.3.1 add_padding()

```
Eigen::MatrixXd Convolution::add_padding (
            int index,
            const Eigen::MatrixXd & tmp_input )
```

A function to add padding to the input tensor, before the convolution process.

Parameters

index	an integer argument shows the index of the input.
tmp_input	an Egen matrix for the input.

Returns

a channel of the input with padding.

Definition at line 61 of file Convolution.cpp.

```
61                                                                                {
62     this->inputs[index].block(this->padding, this->padding, tmp_input.rows(), tmp_input.cols()) =
       tmp_input;
63     return this->inputs[index];
64 }
```

6.5.3.2 compute_currect_operation()

```
std::vector< Eigen::MatrixXd > Convolution::compute_currect_operation (
            std::vector< Eigen::MatrixXd > tmp_input )  [override], [virtual]
```

A function to calculate the forward pass of convolution process on the coming input value.

Parameters

tmp_input	a vector of Eigen matrices, an input coming from the previous computational node.

Returns

a vector of Eigen matrices, returns result of convolution.

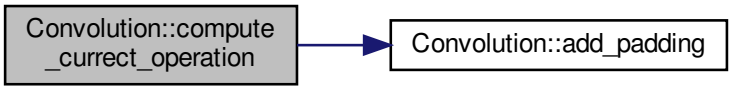
Implements ComputationalNode.

Definition at line 135 of file Convolution.cpp.

```
135                                                                                                                {
136
137     for(size_t i = 0; i < tmp_input.size(); i++) {
138         this->inputs[i] = add_padding(i, tmp_input[i]); // padding is added to the matrix if there is
       any.
139     }
140
141     for(size_t i = 0; i < this->num_of_outputs; i++) {
142         this->outputs[i].setZero();
143     }
144
145     size_t row_start_idx = 0;
146     size_t col_start_idx = 0;
147
148
149     for(size_t filter_idx = 0; filter_idx < this->num_of_filters; filter_idx++) {
150         std::vector<std::vector<Eigen::MatrixXd> input_block_channels;
151         for(size_t filter_channel_idx = 0; filter_channel_idx < this->filter_size_1;
       filter_channel_idx++) {
152             std::vector<Eigen::MatrixXd> input_blocks_per_channel;
153             row_start_idx = 0;
154             for(size_t row = 0; row < this->output_size; row ++) {
155                 col_start_idx = 0;
156                 for(size_t col = 0; col < this->output_size; col ++) {
157                     Eigen::MatrixXd input;
158                     input = this->inputs[filter_channel_idx].block(row_start_idx, col_start_idx,
       this->filter_size,
159                                                                    this->filter_size);
160                     input_blocks_per_channel.emplace_back(input);
161                     this->outputs[filter_idx](row, col) +=
       input.cwiseProduct(this->filters[filter_idx][filter_channel_idx]).sum();
162                     col_start_idx += this->stride;
163                 }
164                 row_start_idx += this->stride;
165             }
166             input_block_channels.emplace_back(input_blocks_per_channel);
167         }
168         this->outputs[filter_idx] += this->biases[filter_idx];
169         this->input_block_to_conv.emplace_back(input_block_channels);
```

```
170     }
171     return this->outputs;
172
173 }
```

Here is the call graph for this function:



6.5.3.3 get_output_size()

```
std::array<int, 3> Convolution::get_output_size ( )  [inline], [override], [virtual]
```

A function to return the output size of this computational node.

Returns

respectively, number of outputs, height of the output, and the width of the output.

Implements ComputationalNode.

Definition at line 121 of file Convolution.h.

```
121                                         { return std::array<int, 3> { this->num_of_filters,
      this->output_size,
122                                                                       this->output_size};}
```

6.5.3.4 random_number_generator()

```
void Convolution::random_number_generator (
            Eigen::MatrixXd & tmp_filter,
            int tmp_filter_size,
            int tmp_input_size )
```

A function to generate random numbers for the initialization of filters and biases. Random numbers are generated by using Xavier initialization.

Parameters

tmp_filter	the filter
tmp_filter_size	size of the filter
tmp_input_size	

Definition at line 254 of file Convolution.cpp.

```
254
            {
255     srand(static_cast<unsigned int>(clock()));
256     std::random_device dev;
257     std::mt19937 engine3(dev());
258     std::normal_distribution<double> distribution(0.0, 1.0);
259     for(size_t i = 0; i < tmp_filter_size; i++){
260         for(size_t j = 0; j < tmp_filter_size; j++) {
261             tmp_filter(i, j) = distribution(engine3);
262         }
263     }
264 }
```

6.5.3.5 read_binary()

```
void Convolution::read_binary (
            std::ifstream & in )  [override], [virtual]
```

A function to load the biases and filters back into the network.

Parameters

in	an ifstream object which is from the file opened to load the biases and filters.

Implements ComputationalNode.

Definition at line 231 of file Convolution.cpp.

```
231                                             {
232
233     for(auto const& vector1: this->filters) {
234         for (auto const& matrix: vector1) {
235             typename Eigen::MatrixXd::Index rows=matrix.rows(), cols=matrix.cols();
236             in.read((char*) (&rows),sizeof(typename Eigen::MatrixXf::Index));
237             in.read((char*) (&cols),sizeof(typename Eigen::MatrixXf::Index));
238             //matrix.resize(rows, cols);
239             in.read( (char *) matrix.data() , rows*cols*sizeof(typename Eigen::MatrixXd::Scalar) );
240         }
241     }
242
243
244     for(auto const& matrix: this->biases) {
245         typename Eigen::MatrixXd::Index rows=matrix.rows(), cols=matrix.cols();
246         in.read((char*) (&rows),sizeof(typename Eigen::MatrixXf::Index));
247         in.read((char*) (&cols),sizeof(typename Eigen::MatrixXf::Index));
248         //matrix.resize(rows, cols);
249         in.read( (char *) matrix.data() , rows*cols*sizeof(typename Eigen::MatrixXd::Scalar) );
250     }
251 }
```

6.5.3.6 set_derivative()

```
std::vector< Eigen::MatrixXd > Convolution::set_derivative (
            std::vector< Eigen::MatrixXd > prev_derivative )  [override], [virtual]
```

A function to calculate the backpropagation of the convolution process.

Parameters

prev_derivative	a vector of Eigen matrices, down flow derivative through this layer.

Returns

a vector of Eigen matrices, the multiplication of the current derivative of the operation and the input down flow derivative.

Implements ComputationalNode.

Definition at line 175 of file Convolution.cpp.

```
175 {
176
177     //this->derivative_b = prev_derivatives;
178
179     for(size_t j = 0; j < this->num_of_inputs; j++) {
180         this->derivative_x[j].setZero();
181     }
182
183     for(size_t i = 0; i < this->num_of_outputs; i++) {
184         this->derivative_b[i] += prev_derivatives[i];
185     }
186     size_t row_start_idx = 0;
187     size_t col_start_idx = 0;
188
189     for(size_t prev_der = 0; prev_der < prev_derivatives.size(); prev_der++) {
190
191         for(size_t channel_idx = 0; channel_idx < this->num_of_inputs; channel_idx++) {
192             int input_blocks_idx = 0;
193             row_start_idx = 0;
194             for(size_t i = 0; i < prev_derivatives[prev_der].rows(); i++) {
195                 col_start_idx = 0;
196                 for(size_t j = 0; j < prev_derivatives[prev_der].cols(); j++) {
197                     this->derivative_w[prev_der][channel_idx] +=
       this->input_block_to_conv[prev_der][channel_idx][input_blocks_idx] * prev_derivatives[prev_der](i,
       j);
198                     this->derivative_x[channel_idx].block(row_start_idx, col_start_idx,
       this->filter_size,
199                                                                       this->filter_size) +=
       this->filters[prev_der][channel_idx] * prev_derivatives[prev_der](i, j);
200                     input_blocks_idx ++;
201                     col_start_idx += this->stride;
202                 }
203                 row_start_idx += this->stride;
204             }
205         }
206     }
207
208     this->input_block_to_conv.clear();
209
210     return this->derivative_x;
211 }
```

6.5.3.7 set_parameter_values()

```
void Convolution::set_parameter_values (
            double learning_rate,
            int batch_size ) [override], [virtual]
```

A function to update the filters and biases of the convolution node.

Parameters

learning_rate	a double argument, learning rate.
batch_size	an integer argument, the number of samples in one batch.

Implements ComputationalNode.

Definition at line 67 of file Convolution.cpp.

```
67                                                                          {
68
69      //this->input_block_to_conv.clear();
70      for(size_t i = 0; i < this->num_of_inputs; i++) {
71          this->inputs[i].setZero();
72      }
73
74      for(size_t i = 0; i < this->num_of_outputs; i++) {
75          //this->velocity_b[i] = 0.9 * this->velocity_b[i] + this->derivative_b[i]/float(batch_size);
76          this->biases[i] -= learning_rate * this->derivative_b[i]/float(batch_size);
77
78          for(size_t j = 0; j < this->filter_size_1; j++) {
79              //this->velocity_w[i][j] = 0.9 * this->velocity_w[i][j] +
        this->derivative_w[i][j]/float(batch_size);
80              this->filters[i][j] -= learning_rate * this->derivative_w[i][j]/float(batch_size);
81          }
82      }
83
84      for(size_t i = 0; i < this->num_of_outputs; i++) {
85          this->derivative_b[i].setZero();
86          this->outputs[i].setZero();
87          for(size_t j = 0; j < this->filter_size_1; j++) {
88              this->derivative_w[i][j].setZero();
89          }
90      }
91
92
93  }
```

6.5.3.8 testing_compute_currect_operation()

```
std::vector< Eigen::MatrixXd > Convolution::testing_compute_currect_operation (
            std::vector< Eigen::MatrixXd > tmp_input )  [override], [virtual]
```

A function to run the forward pass of the convolution operation during the testing phase.

Parameters

tmp_input	a vector of Eigen matrices, an input coming from the previous computational node.

Returns

a vector of Eigen matrices, returns the values of the input after the convolution operation.

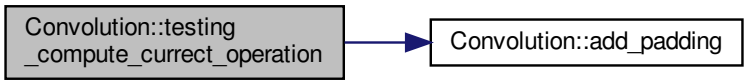
Implements ComputationalNode.

Definition at line 96 of file Convolution.cpp.

```
96
        {
97
98      for(size_t i = 0; i < tmp_input.size(); i++) {
99          this->inputs[i] = add_padding(i, tmp_input[i]); // padding is added to the matrix if there is
        any.
100      }
101
102      for(size_t i = 0; i < this->num_of_outputs; i++) {
103          this->outputs[i].setZero();
104      }
105
106      size_t row_start_idx = 0;
107      size_t col_start_idx = 0;
108
109
110      for(size_t filter_idx = 0; filter_idx < this->num_of_filters; filter_idx++) {
111          for(size_t filter_channel_idx = 0; filter_channel_idx < this->filter_size_1;
        filter_channel_idx++) {
112              row_start_idx = 0;
113              for(size_t row = 0; row < this->output_size; row ++) {
114                  col_start_idx = 0;
115                  for(size_t col = 0; col < this->output_size; col ++) {
116                      Eigen::MatrixXd input;
```

```
117                     input = this->inputs[filter_channel_idx].block(row_start_idx, col_start_idx,
    this->filter_size,
118                                                                    this->filter_size);
119                     this->outputs[filter_idx](row, col) +=
    input.cwiseProduct(this->filters[filter_idx][filter_channel_idx]).sum();
120                     col_start_idx += this->stride;
121                 }
122                 row_start_idx += this->stride;
123             }
124         }
125         this->outputs[filter_idx] += this->biases[filter_idx];
126     }
127     return this->outputs;
128
129 }
```

Here is the call graph for this function:



6.5.3.9 write_binary()

```
void Convolution::write_binary (
            std::ofstream & out )  [override], [virtual]
```

A function to save the filters and biases of the convolution layer to a file.

Parameters

out	an ofstream object which is from the file opened to save the biases and filters.

Implements ComputationalNode.

Definition at line 215 of file Convolution.cpp.

```
215                                                 {
216     for(auto const& vector1: this->filters) {
217         for (auto const& matrix: vector1) {
218             typename Eigen::MatrixXd::Index rows=matrix.rows(), cols=matrix.cols();
219             out.write((char*) (&rows), sizeof(typename Eigen::MatrixXd::Index));
220             out.write((char*) (&cols), sizeof(typename Eigen::MatrixXd::Index));
221             out.write((char*) matrix.data(), rows*cols*sizeof(typename Eigen::MatrixXd::Scalar) );
222         }
223     }
224     for(auto const& matrix: this->biases) {
225         typename Eigen::MatrixXd::Index rows=matrix.rows(), cols=matrix.cols();
226         out.write((char*) (&rows), sizeof(typename Eigen::MatrixXd::Index));
227         out.write((char*) (&cols), sizeof(typename Eigen::MatrixXd::Index));
228         out.write((char*) matrix.data(), rows*cols*sizeof(typename Eigen::MatrixXd::Scalar) );
229     }
230 }
```

6.5.4 Member Data Documentation

6.5.4.1 biases

```
std::vector<Eigen::MatrixXd> Convolution::biases
```

The biases to add to the each filter calculation.

Definition at line 19 of file Convolution.h.

6.5.4.2 derivative_b

```
std::vector<Eigen::MatrixXd> Convolution::derivative_b
```

Derivative of convolution process with respect to the biases.

Definition at line 27 of file Convolution.h.

6.5.4.3 derivative_w

```
std::vector<std::vector<Eigen::MatrixXd> > Convolution::derivative_w
```

convolution filters.

Derivative of convolution process with respect to the

Definition at line 24 of file Convolution.h.

6.5.4.4 derivative_x

```
std::vector<Eigen::MatrixXd> Convolution::derivative_x
```

Derivative of the convolution process with respect to the input.

Definition at line 23 of file Convolution.h.

6.5.4.5 filter_size

```
int Convolution::filter_size
```

Filter size.

Definition at line 35 of file Convolution.h.

6.5.4.6 filter_size_1

```
int Convolution::filter_size_1
```

filter depth

Definition at line 36 of file Convolution.h.

6.5.4.7 filters

```
std::vector<std::vector<Eigen::MatrixXd> > Convolution::filters
```

operation. Since filters have a depth as well, they will be kept in vector of vector of matrices.

The filters which is going to be used in convolution

Definition at line 17 of file Convolution.h.

6.5.4.8 input_block_to_conv

```
std::vector<std::vector<std::vector<Eigen::MatrixXd> > > Convolution::input_block_to_conv
```

backpropagation later, in the forward pass all the convolution pathes of input, kept in this vector.

To be able to calculate the

Definition at line 20 of file Convolution.h.

6.5.4.9 inputs

```
std::vector<Eigen::MatrixXd> Convolution::inputs
```

The input feature maps of the convolution computational node.

Definition at line 15 of file Convolution.h.

6.5.4.10 num_of_filters

```
int Convolution::num_of_filters
```

number of filters

Definition at line 37 of file Convolution.h.

6.5.4.11 num_of_inputs

```
int Convolution::num_of_inputs
```

input depth

Definition at line 38 of file Convolution.h.

6.5.4.12 num_of_outputs

```
int Convolution::num_of_outputs
```

the depth of the output.

Definition at line 32 of file Convolution.h.

6.5.4.13 output_size

```
int Convolution::output_size
```

Output square channel size of the convolution node.

Definition at line 31 of file Convolution.h.

6.5.4.14 outputs

```
std::vector<Eigen::MatrixXd> Convolution::outputs
```

The output of the convolution computation node.

Definition at line 16 of file Convolution.h.

6.5.4.15 padding

```
int Convolution::padding
```

Padding to use during the convolution process.

Definition at line 33 of file Convolution.h.

6.5.4.16 stride

```
int Convolution::stride
```

Stride to use during the convolution process.

Definition at line 34 of file Convolution.h.

6.5.4.17 velocity_b

```
std::vector<Eigen::MatrixXd> Convolution::velocity_b
```

Velocity to update the biases using the momentum.

Definition at line 28 of file Convolution.h.

6.5.4.18 velocity_w

```
std::vector<std::vector<Eigen::MatrixXd> > Convolution::velocity_w
```

Velocity to update the filters using the momentum.

Definition at line 26 of file Convolution.h.

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

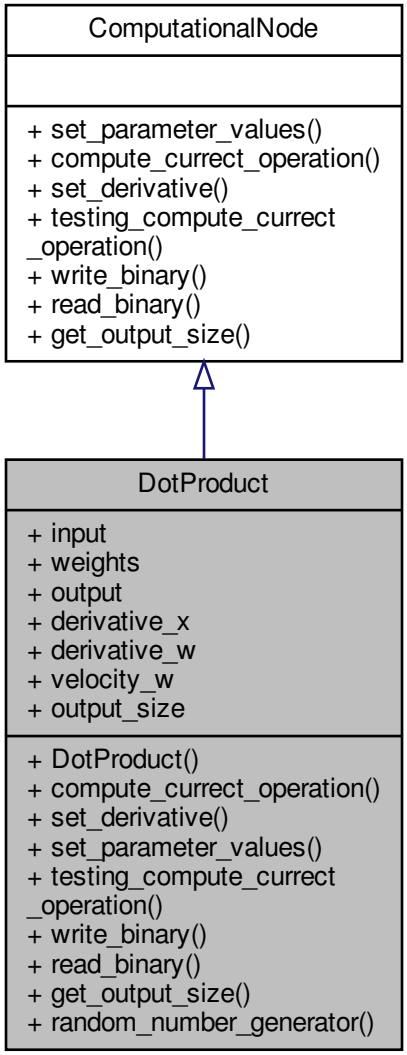
- /home/nehil/dlfsC++/libdl/headers/Convolution.h
- /home/nehil/dlfsC++/libdl/src/Convolution.cpp

6.6 DotProduct Class Reference

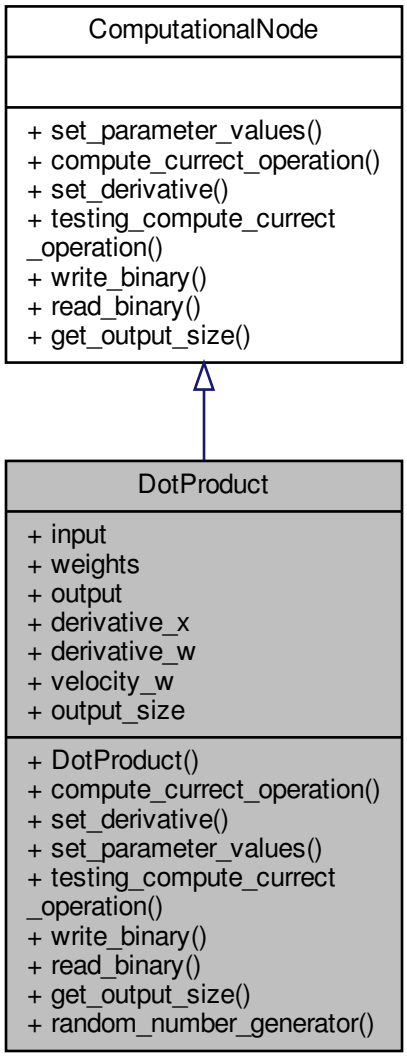
A class which implements the dot product operation in the network.

`#include <DotProduct.h>`

Inheritance diagram for DotProduct:



Collaboration diagram for DotProduct:



Public Member Functions

- DotProduct (int row, int column)
- std::vector< Eigen::MatrixXd > compute_currect_operation (std::vector< Eigen::MatrixXd > tmp_input) override
- std::vector< Eigen::MatrixXd > set_derivative (std::vector< Eigen::MatrixXd > prev_derivative) override
- void set_parameter_values (double learning_rate, int batch_size) override
- std::vector< Eigen::MatrixXd > testing_compute_currect_operation (std::vector< Eigen::MatrixXd > tmp_↩ input) override
- void write_binary (std::ofstream &out) override
- void read_binary (std::ifstream &in) override
- std::array< int, 3 > get_output_size () override

Static Public Member Functions

- static void random_number_generator (Eigen::MatrixXd &tmp_weights, int input_nodes_layer1, int input_↩ nodes_layer2)

Public Attributes

- std::vector< Eigen::MatrixXd > input

  the input value of the dot product operation.
- Eigen::MatrixXd weights

  weights of this layer.
- std::vector< Eigen::MatrixXd > output

  the output value of the dot product operation will be stored here after the calculation.
- std::vector< Eigen::MatrixXd > derivative_x

  the derivative of the dot product operation with respect to the input value will be kept here.
- Eigen::MatrixXd derivative_w

  the derivative of the dot product operation with respect to weights.
- Eigen::MatrixXd velocity_w

  the velocity for the momentum gradient descent.
- int output_size

  the height of the output layer

6.6.1 Detailed Description

A class which implements the dot product operation in the network.

Definition at line 14 of file DotProduct.h.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 DotProduct()

```
DotProduct::DotProduct (
            int row,
            int column )
```

The constructor of the dot product computation node.

Parameters

row	the number of neurons in the next layer.
column	the number of neurons in the previous layer.
tmp_layer_num	the layer id of the dot product layer.

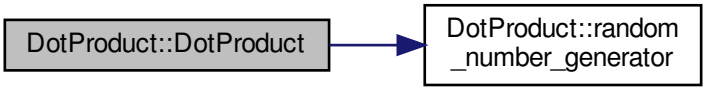
initialization of all the class variables.

< velocity will be use in the case of using

Definition at line 7 of file DotProduct.cpp.

```
7                                           {
11      this->output_size = row;
12      this->input.emplace_back(Eigen::MatrixXd::Zero(column,1));
13      this->output.emplace_back(Eigen::MatrixXd::Zero(row,1));
14      this->weights = Eigen::MatrixXd::Zero(row, column);
15      random_number_generator(this->weights, row, column);
16      this->weights = this->weights * sqrt(2.0/column);
17      this->derivative_x.emplace_back(Eigen::MatrixXd::Zero(column,1));
18      this->derivative_w = Eigen::MatrixXd::Zero(row, column);
19      this->velocity_w = Eigen::MatrixXd::Zero(row, column);
20  }
```

Here is the call graph for this function:



6.6.3 Member Function Documentation

6.6.3.1 compute_currect_operation()

```
std::vector< Eigen::MatrixXd > DotProduct::compute_currect_operation (
              std::vector< Eigen::MatrixXd > tmp_input )  [override], [virtual]
```

The forward pass of the dot product will be calculated here.

Parameters

tmp_input	the input value passing through the previous nodes.

Returns

the result of the dot product operation.

< the input value is assigned to the input variable of the computational node.

< the output will be the multipllication of the weights and the input.

Implements ComputationalNode.

Definition at line 22 of file DotProduct.cpp.

```
22                                                                                                  {
23      this->input = tmp_input;
24      this->output[0] = this->weights * tmp_input[0];
25      return this->output;
26
27  }
```

6.6.3.2 get_output_size()

```
std::array<int, 3> DotProduct::get_output_size ( )  [inline], [override], [virtual]
```

A function to return the output size of this computational node.

Returns

respectively, number of outputs, height of the output, and the width of the output.

Implements ComputationalNode.

Definition at line 95 of file DotProduct.h.

```
95                                                  { return std::array<int, 3> { 1, this->output_size,
96                                                                                  1};}
```

6.6.3.3 random_number_generator()

```
void DotProduct::random_number_generator (
            Eigen::MatrixXd & tmp_weights,
            int input_nodes_layer1,
            int input_nodes_layer2 )  [static]
```

A function to generate random numbers for the initialization of weights. Random numbers are generated by using Xavier initialization.

Parameters

tmp_weights	an Eigen matrix, weight value in this computational node.
input_nodes_layer1	an integer argument, number of rows of the weight matrix.
input_nodes_layer2	an integer argument, number of columns of the weight matrix.

For each element of the weight matrix of the dot product computational node, assigned by the normal gaussian distribution.

Definition at line 50 of file DotProduct.cpp.

```
50
                {
54      srand(static_cast<unsigned int>(clock()));
55      std::random_device dev;
56      std::mt19937 engine3(dev());
57      std::normal_distribution<double> distribution(0.0,1.0);
58      for(size_t i = 0; i < input_nodes_layer1; i++){
59          for(size_t j = 0; j < input_nodes_layer2; j++) {
60              tmp_weights(i, j) = distribution(engine3);
61          }
62      }
63
64  }
```

6.6.3.4 read_binary()

```
void DotProduct::read_binary (
            std::ifstream & in )  [override], [virtual]
```

A function to load the weights before the training is started.

Parameters

in	an ifstream object that is from the file opened to load the weights.

< the row and the column size of the matrix that will be loaded to the network.

Implements ComputationalNode.

Definition at line 96 of file DotProduct.cpp.

```
96                                          {
97         typename Eigen::MatrixXd::Index rows=this->weights.rows(), cols=this->weights.cols();
98         in.read((char*) (&rows),sizeof(typename Eigen::MatrixXf::Index));
99         in.read((char*) (&cols),sizeof(typename Eigen::MatrixXf::Index));
100         in.read( (char *) this->weights.data() , rows*cols*sizeof(typename Eigen::MatrixXd::Scalar) );
101
102 }
```

6.6.3.5 set_derivative()

```
std::vector< Eigen::MatrixXd > DotProduct::set_derivative (
             std::vector< Eigen::MatrixXd > prev_derivative )  [override], [virtual]
```

A function to calculate the backpropagation of the dot product operation.

Parameters

prev_derivative	the down flowing derivative.

Returns

the multiplication of the current derivative and the down flowing derivative.

< in this line, the derivative with respect to the weights

< is just added to each other to make a mini batch calculation.

Implements ComputationalNode.

Definition at line 68 of file DotProduct.cpp.

```
68                                                                                          {
69
70     this->derivative_x[0].setZero();
71
72     for(size_t i = 0 ; i < this->weights.cols(); i++) {
73         this->derivative_x[0](i) = (this->weights.col(i).cwiseProduct(prev_derivative[0])).sum();
74     }
75
76
77     for(size_t j = 0 ; j < this->weights.rows(); j++){
78         this->derivative_w.row(j) += this->input[0].transpose() * prev_derivative[0](j);
79     }
81
82     return this->derivative_x;
83 }
```

6.6.3.6 set_parameter_values()

```
void DotProduct::set_parameter_values (
             double learning_rate,
             int batch_size )  [override], [virtual]
```

A function to update the weights after the backpropagation operation.

Parameters

learning_rate	
batch_size	

< intead of only using the gradient, the weighted mean of the gradient is used.

After the update of the weights, all the class variables assigned back to their initial values.

Implements ComputationalNode.

Definition at line 36 of file DotProduct.cpp.

```
36                                                                          {
37     //this->velocity_w = 0.9 * this->velocity_w + this->derivative_w/float(batch_size); //!< the momentum
       gradient descent calculated here.
38     this->weights -= (learning_rate * this->derivative_w/float(batch_size));
39
43     this->derivative_w.setZero();
44     this->derivative_x[0].setZero();
45     this->input[0].setZero();
46     this->output[0].setZero();
47 }
```

6.6.3.7 testing_compute_currect_operation()

```
std::vector< Eigen::MatrixXd > DotProduct::testing_compute_currect_operation (
             std::vector< Eigen::MatrixXd > tmp_input )  [override], [virtual]
```

A function to run the forward pass of the dot product operation during the testing phase.

Parameters

tmp_input	a vector of Eigen matrices, an input coming from the previous computational node.

Returns

a vector of Eigen matrices, returns the values of the input after the dot product operation.

Implements ComputationalNode.

Definition at line 29 of file DotProduct.cpp.

```
29
       {
30     this->input = tmp_input;
31     this->output[0] = this->weights * tmp_input[0];
32     return this->output;
33 }
```

6.6.3.8 write_binary()

```
void DotProduct::write_binary (
             std::ofstream & out )  [override], [virtual]
```

A function to save the weights after the training is done.

Parameters

out	an ofstream object that is from the file opened to save the weights.

< the row and the column size of the matrix that will be saved to a file.

Implements ComputationalNode.

Definition at line 88 of file DotProduct.cpp.

```
88                                          {
89
90     typename Eigen::MatrixXd::Index rows=this->weights.rows(), cols=this->weights.cols();
91     out.write((char*) (&rows), sizeof(typename Eigen::MatrixXd::Index));
92     out.write((char*) (&cols), sizeof(typename Eigen::MatrixXd::Index));
93     out.write((char*) this->weights.data(), rows*cols*sizeof(typename Eigen::MatrixXd::Scalar) );
94 }
```

6.6.4 Member Data Documentation

6.6.4.1 derivative_w

```
Eigen::MatrixXd DotProduct::derivative_w
```

the derivative of the dot product operation with respect to weights.

Definition at line 24 of file DotProduct.h.

6.6.4.2 derivative_x

```
std::vector<Eigen::MatrixXd> DotProduct::derivative_x
```

the derivative of the dot product operation with respect to the input value will be kept here.

Definition at line 22 of file DotProduct.h.

6.6.4.3 input

```
std::vector<Eigen::MatrixXd> DotProduct::input
```

the input value of the dot product operation.

Definition at line 16 of file DotProduct.h.

6.6.4.4 output

```
std::vector<Eigen::MatrixXd> DotProduct::output
```

the output value of the dot product operation will be stored here after the calculation.

Definition at line 20 of file DotProduct.h.

6.6.4.5 output_size

```
int DotProduct::output_size
```

the height of the output layer

Definition at line 28 of file DotProduct.h.

6.6.4.6 velocity_w

```
Eigen::MatrixXd DotProduct::velocity_w
```

the velocity for the momentum gradient descent.

Definition at line 26 of file DotProduct.h.

6.6.4.7 weights

```
Eigen::MatrixXd DotProduct::weights
```

weights of this layer.

Definition at line 18 of file DotProduct.h.

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

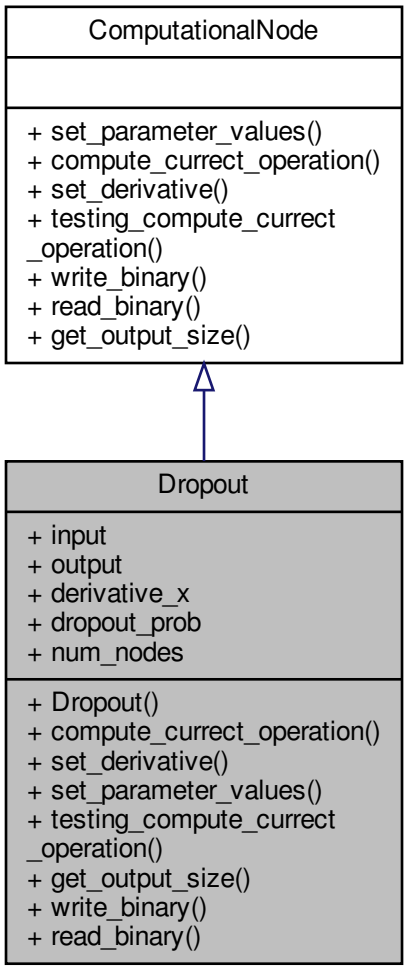
- /home/nehil/dlfsC++/libdl/headers/DotProduct.h
- /home/nehil/dlfsC++/libdl/src/DotProduct.cpp

6.7 Dropout Class Reference

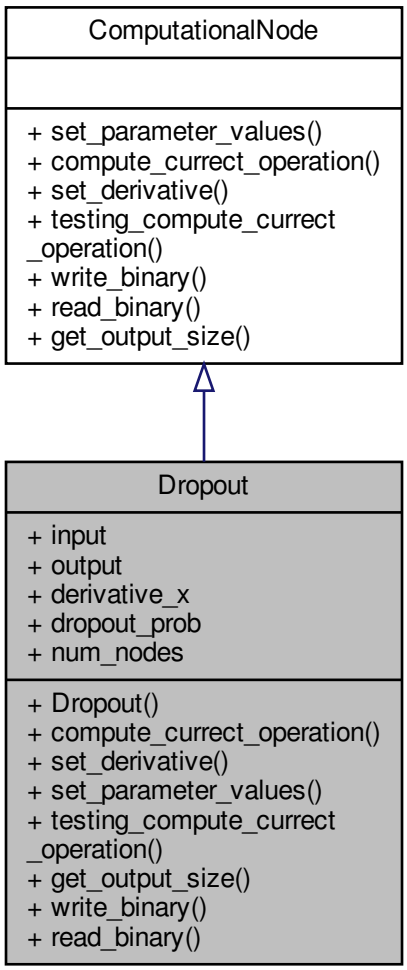
A class which implements the dropout computation in the network.

```
#include <Dropout.h>
```

Inheritance diagram for Dropout:



Collaboration diagram for Dropout:



Public Member Functions

- Dropout (int num_nodes, double dropout_prob)
- std::vector< Eigen::MatrixXd > compute_currect_operation (std::vector< Eigen::MatrixXd > tmp_input) override
- std::vector< Eigen::MatrixXd > set_derivative (std::vector< Eigen::MatrixXd > prev_derivative) override
- void set_parameter_values (double learning_rate, int batch_size) override
- std::vector< Eigen::MatrixXd > testing_compute_currect_operation (std::vector< Eigen::MatrixXd > tmp_↩ input) override
- std::array< int, 3 > get_output_size () override
- void write_binary (std::ofstream &out) override
- void read_binary (std::ifstream &in) override

Public Attributes

- std::vector< Eigen::MatrixXd > input

  the input of the dropout computational node.
- std::vector< Eigen::MatrixXd > output

  the output of the dropout computational node.
- std::vector< Eigen::MatrixXd > derivative_x

  the derivative of the dropout operation with respect to the input
- double dropout_prob

  the probability of dropout
- int num_nodes

6.7.1 Detailed Description

A class which implements the dropout computation in the network.

Dropout operation can only be used in the fully connected part of the network.

Definition at line 15 of file Dropout.h.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 Dropout()

```
Dropout::Dropout (
            int num_nodes,
            double dropout_prob )  [explicit]
```

A constructor for the dropout computational node.

Parameters

num_nodes	an integer argument, the number of neurons in the dropout layer
dropout_prob	a double argument, the probability of dropout operation

Definition at line 8 of file Dropout.cpp.

```
8                                                       {
9     this->num_nodes = num_nodes;
10     this->dropout_prob = tmp_dropout_prob;
11     this->input.emplace_back(Eigen::MatrixXd::Zero(num_nodes,1));
12     this->output.emplace_back(Eigen::MatrixXd::Zero(num_nodes,1));
13     this->derivative_x.emplace_back(Eigen::MatrixXd::Ones(num_nodes,1));
14
15 }
```

6.7.3 Member Function Documentation

6.7.3.1 compute_currect_operation()

```
std::vector< Eigen::MatrixXd > Dropout::compute_currect_operation (
            std::vector< Eigen::MatrixXd > tmp_input )  [override], [virtual]
```

A function to compute the forward pass of the dropout computational node.

Parameters

tmp_input	a vector of matrices, input of the dropout layer

Returns

the result of the forward pass of the dropout layer.

Implements ComputationalNode.

Definition at line 24 of file Dropout.cpp.

```
24                                                                                                  {
25     this->input = tmp_input;
26     std::random_device rd;
27     std::mt19937 gen(rd());
28     std::bernoulli_distribution dist(1 - this->dropout_prob); // bernoulli_distribution takes chance of
       true n constructor
29
30     for (int i = 0; i < this->input[0].size(); i++) {
31         int random_num = dist(gen);
32         this->derivative_x[0](i, 0) = random_num;
33         this->output[0](i, 0) = random_num;
34     }
35
36     this->output[0] /= this->dropout_prob;
37     this->derivative_x[0] /= this->dropout_prob;
38
39     this->output[0] = this->output[0].cwiseProduct(this->input[0]);
40
41     return this->output;
42
43 }
```

6.7.3.2 get_output_size()

```
std::array<int, 3> Dropout::get_output_size ( )  [inline], [override], [virtual]
```

A function to return the output size of this computational node.

Returns

respectively, number of outputs, height of the output, and the width of the output.

Implements ComputationalNode.

Definition at line 70 of file Dropout.h.

```
70                                                  { return std::array<int, 3> { 1, this->num_nodes,
71                                                                                    1};}
```

6.7.3.3 read_binary()

```
void Dropout::read_binary (
             std::ifstream & in )  [inline], [override], [virtual]
```

A virtual function which will be override by the child classes.

- This function is for loading the weights and the biases before the training of the network starts.

Parameters

in	an ifstream object

Implements ComputationalNode.

Definition at line 73 of file Dropout.h.

```
73 {};
```

6.7.3.4 set_derivative()

```
std::vector< Eigen::MatrixXd > Dropout::set_derivative (
            std::vector< Eigen::MatrixXd > prev_derivative )  [override], [virtual]
```

A function to calculate the backward pass of the dropout computational layer.

Parameters

prev_derivative	a vector of matrices, the down flowing derivative of the network.

Returns

the multiplication of the down flowing derivative and the gradient of the dropout layer

Implements ComputationalNode.

Definition at line 65 of file Dropout.cpp.

```
65                                                                                  {
66     this->derivative_x[0] = this->derivative_x[0].cwiseProduct(prev_derivative[0]);
67     return this->derivative_x;
68 }
```

6.7.3.5 set_parameter_values()

```
void Dropout::set_parameter_values (
            double learning_rate,
            int batch_size )  [override], [virtual]
```

A function to set weights and biases by using the derivatives, which are calculated during the backward pass, and to set input, output and derivative values to zero. Since there is no weight or bias in the dropout layer, it is used only for setting all the input, output and, derivatives back to their initial values.

Parameters

learning_rate	a double argument, learning rate.
batch_size	an integer argument, the number of elements in one batch.

Implements ComputationalNode.

Definition at line 17 of file Dropout.cpp.

```
17                                                                                {
18     this->input[0].setZero();
19     this->derivative_x[0].setZero();
20     this->output[0].setZero();
21
22 }
```

6.7.3.6 testing_compute_currect_operation()

```
std::vector< Eigen::MatrixXd > Dropout::testing_compute_currect_operation (
            std::vector< Eigen::MatrixXd > tmp_input )  [override], [virtual]
```

A function to compute the forward pass of the dropout computational node during the testing phase.

Parameters

tmp_input	a vector of matrices, input of the dropout layer

Returns

the result of the forward pass of the dropout layer.

Implements ComputationalNode.

Definition at line 45 of file Dropout.cpp.

```
45                                                                                                                {
46     this->input = tmp_input;
47     std::random_device rd;
48     std::mt19937 gen(rd());
49     std::bernoulli_distribution dist(1 - this->dropout_prob); // bernoulli_distribution takes chance of
       true n constructor
50
51     for (int i = 0; i < this->input[0].size(); i++) {
52         int random_num = dist(gen);
53         this->derivative_x[0](i, 0) = random_num;
54         this->output[0](i, 0) = random_num;
55     }
56
57     this->output[0] /= this->dropout_prob;
58     this->derivative_x[0] /= this->dropout_prob;
59
60     this->output[0] = this->output[0].cwiseProduct(this->input[0]);
61
62     return this->output;
63 }
```

6.7.3.7 write_binary()

```
void Dropout::write_binary (
            std::ofstream & out )  [inline], [override], [virtual]
```

A virtual function which will be override by the child classes. This function is for saving the weights and the biases after the training of the network.

Parameters

out	an ofstream object

Implements ComputationalNode.

Definition at line 72 of file Dropout.h.

```
72 {};
```

6.7.4 Member Data Documentation

6.7.4.1 derivative_x

```
std::vector<Eigen::MatrixXd> Dropout::derivative_x
```

the derivative of the dropout operation with respect to the input

Definition at line 20 of file Dropout.h.

6.7.4.2 dropout_prob

```
double Dropout::dropout_prob
```

the probability of dropout

Definition at line 21 of file Dropout.h.

6.7.4.3 input

```
std::vector<Eigen::MatrixXd> Dropout::input
```

the input of the dropout computational node.

Definition at line 18 of file Dropout.h.

6.7.4.4 num_nodes

```
int Dropout::num_nodes
```

Definition at line 22 of file Dropout.h.

6.7.4.5 output

```
std::vector<Eigen::MatrixXd> Dropout::output
```

the output of the dropout computational node.

Definition at line 19 of file Dropout.h.

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

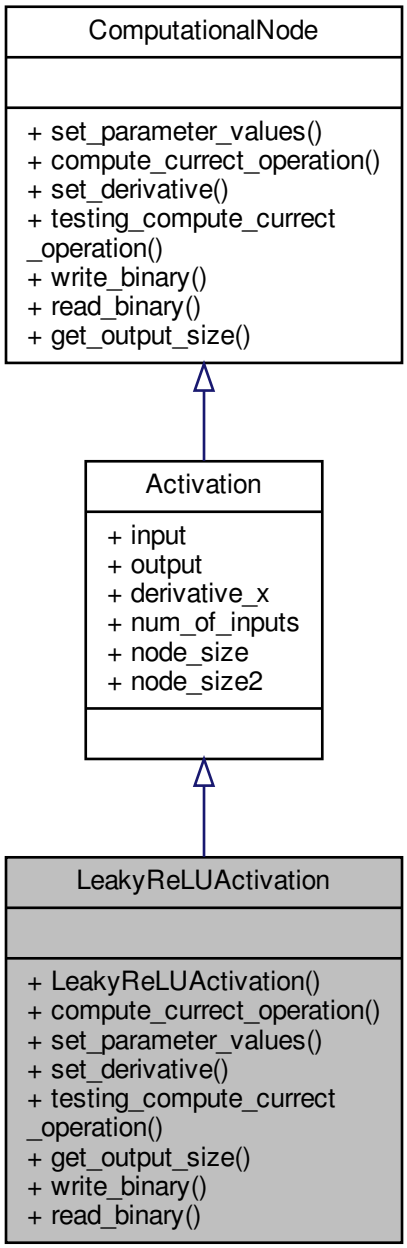
- /home/nehil/dlfsC++/libdl/headers/Dropout.h
- /home/nehil/dlfsC++/libdl/src/Dropout.cpp

6.8 LeakyReLUActivation Class Reference

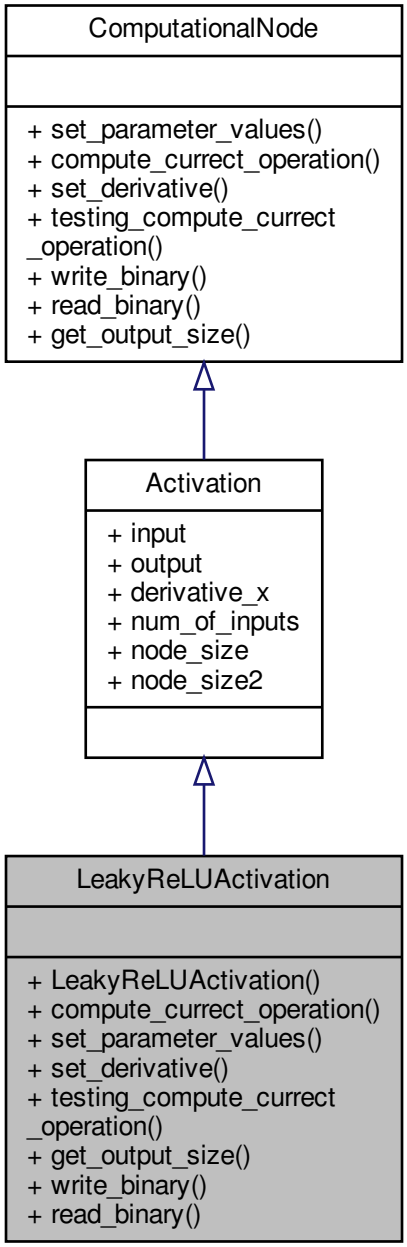
A class for the Leaky Rectified Linear Units (RELU) Activation function.

`#include <Activation.h>`

Inheritance diagram for LeakyReLUActivation:



Collaboration diagram for LeakyReLUActivation:



Public Member Functions

- LeakyReLUActivation (int num_of_inputs, int num_nodes1, int num_nodes2)
- std::vector< Eigen::MatrixXd > compute_currect_operation (std::vector< Eigen::MatrixXd > tmp_input) override
- void set_parameter_values (double learning_rate, int batch_size) override
- std::vector< Eigen::MatrixXd > set_derivative (std::vector< Eigen::MatrixXd > prev_derivative) override

- std::vector< Eigen::MatrixXd > testing_compute_currect_operation (std::vector< Eigen::MatrixXd > tmp_input) override
- std::array< int, 3 > get_output_size () override
- void write_binary (std::ofstream &out) override
- void read_binary (std::ifstream &in) override

Additional Inherited Members

6.8.1 Detailed Description

A class for the Leaky Rectified Linear Units (RELU) Activation function.

Definition at line 165 of file Activation.h.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 LeakyReLUActivation()

```
LeakyReLUActivation::LeakyReLUActivation (
            int num_of_inputs,
            int num_nodes1,
            int num_nodes2 )  [explicit]
```

A constructor for Leaky RELU activation operation. This class can be used for both fully connected layer, and the convolution layers.

Parameters

num_of_inputs	an integer argument, the number of feature maps.
num_nodes1	an integer argument, refers to the height input matrix.
num_nodes2	an integer argument, refers to width of the input matrix.

the initializaion of the input, output and the derivative with respect to the input variables.

Definition at line 133 of file Activation.cpp.

```
133                                                                                                 {
137     this->num_of_inputs = num_of_inputs;
138     this->node_size = num_nodes1;
139     this->node_size2 = num_nodes2;
140     for (int i = 0; i < num_of_inputs; ++i) {
141         this->input.emplace_back(Eigen::MatrixXd::Zero(num_nodes1,num_nodes2));
142         this->output.emplace_back(Eigen::MatrixXd::Zero(num_nodes1,num_nodes2));
143         this->derivative_x.emplace_back(Eigen::MatrixXd::Zero(num_nodes1,num_nodes2));
144     }
145 }
```

6.8.3 Member Function Documentation

6.8.3.1 compute_currect_operation()

```
std::vector< Eigen::MatrixXd > LeakyReLUActivation::compute_currect_operation (
            std::vector< Eigen::MatrixXd > tmp_input )  [override], [virtual]
```

The function is to calculate the the current operation of the Leaky RELU node. The Leaky RELU activation function is applied to the given input. If the input value is smaller than 0, then the output will be 0.1 multiplied by the input value. Else it will be the value itself.

Parameters

tmp_input	a vector of Eigen matrices, an input coming from the previous computational node.

Returns

a vector of Eigen matrices, returns the values of the input after applying Leaky RELU activation function.

< calculation of leaky RELU.

Implements ComputationalNode.

Definition at line 148 of file Activation.cpp.

```
148
        {
149
150     this->input = tmp_input;
151     this->output = tmp_input;
152     for(size_t idx = 0; idx < tmp_input.size(); idx ++) {
153         for(size_t i = 0; i < tmp_input[idx].rows(); i++) {
154             for(size_t j = 0; j < tmp_input[idx].cols(); j++) {
155                 if(tmp_input[idx](i, j) <= 0) this->output[idx](i, j) *= 0.1;
156             }
157         }
158     }
159
160     return this->output;
161 }
```

6.8.3.2 get_output_size()

```
std::array<int, 3> LeakyReLUActivation::get_output_size ( )  [inline], [override], [virtual]
```

A function to return the output size of this computational node.

Returns

respectively, number of outputs, height of the output, and the width of the output.

Implements ComputationalNode.

Definition at line 220 of file Activation.h.

```
220                                             { return std::array<int, 3> { this->num_of_inputs,
       this->node_size,
221                                                                           this->node_size2};}
```

6.8.3.3 read_binary()

```
void LeakyReLUActivation::read_binary (
             std::ifstream & in )  [inline], [override], [virtual]
```

A virtual function which will be override by the child classes.

- This function is for loading the weights and the biases before the training of the network starts.

Parameters

in	an ifstream object

Implements ComputationalNode.

Definition at line 224 of file Activation.h.

```
224 {};
```

6.8.3.4 set_derivative()

```
std::vector< Eigen::MatrixXd > LeakyReLUActivation::set_derivative (
            std::vector< Eigen::MatrixXd > prev_derivative )  [override], [virtual]
```

A function to calculate the derivative of the Leaky RELU operation. Derivative of the previous operations in the network are received as the input paramater. The result is the multiplication of the current derivative of the operation and the input down flow derivative.

Parameters

prev_derivative	a vector of Eigen matrices, down flow derivative through this Leaky RELU activation function.

Returns

a vector of Eigen matrices, the multiplication of the current derivative of the operation and the input down flow derivative.

Implements ComputationalNode.

Definition at line 187 of file Activation.cpp.

```
187                                                                                                 {
188
189     for(size_t i = 0; i < prev_derivative.size(); i++) {
190         derivative_x[i].setZero();
191     }
192
193     for(size_t i = 0; i < prev_derivative.size(); i++) {
194         for (int j = 0; j < prev_derivative[i].rows(); j++) {
195             for (int k = 0; k < prev_derivative[i].cols(); k++) {
196                 if(this->input[i](j,k) <= 0) this->derivative_x[i](j, k) = 0.1 * prev_derivative[i](j,
    k);
197                 else this->derivative_x[i](j, k) = 1 * prev_derivative[i](j, k);
198             }
199         }
200     }
201     return this->derivative_x;
202 }
```

6.8.3.5 set_parameter_values()

```
void LeakyReLUActivation::set_parameter_values (
            double learning_rate,
            int batch_size )  [override], [virtual]
```

A function to set weights and biases by using the derivatives, which are calculated during the backward pass, and to set input, output and derivative values to zero. Since there is no weight or bias in the activation function layers, it is used only for setting all the input, output and, derivatives to zero.

Parameters

learning_rate	a double argument, learning rate.
batch_size	an integer argument, the number of elements in one batch.

Implements ComputationalNode.

Definition at line 178 of file Activation.cpp.

```
178                                                                                {
179     for (int i = 0; i < this->input.size(); ++i) {
180         this->derivative_x[i].setZero();
181         this->input[i].setZero();
182         this->output[i].setZero();
183     }
184 }
```

6.8.3.6 testing_compute_currect_operation()

```
std::vector< Eigen::MatrixXd > LeakyReLUActivation::testing_compute_currect_operation (
            std::vector< Eigen::MatrixXd > tmp_input )  [override], [virtual]
```

A function to run the forward pass of the leaky relu activation operation during the testing phase.

Parameters

tmp_input	a vector of Eigen matrices, an input coming from the previous computational node.

Returns

a vector of Eigen matrices, returns the values of the input after the leaky relu activation operation.

Implements ComputationalNode.

Definition at line 163 of file Activation.cpp.

```
164                                                 {
165     this->input = tmp_input;
166     this->output = tmp_input;
167     for(size_t idx = 0; idx < tmp_input.size(); idx ++) {
168         for(size_t i = 0; i < tmp_input[idx].rows(); i++) {
169             for(size_t j = 0; j < tmp_input[idx].cols(); j++) {
170                 if(tmp_input[idx](i, j) <= 0) this->output[idx](i, j) *= 0.1;
171             }
172         }
173     }
174
175     return this->output;
176 }
```

6.8.3.7 write_binary()

```
void LeakyReLUActivation::write_binary (
            std::ofstream & out )  [inline], [override], [virtual]
```

A virtual function which will be override by the child classes. This function is for saving the weights and the biases after the training of the network.

Parameters

out	an ofstream object

Implements ComputationalNode.

Definition at line 223 of file Activation.h.

```
223 {};
```

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

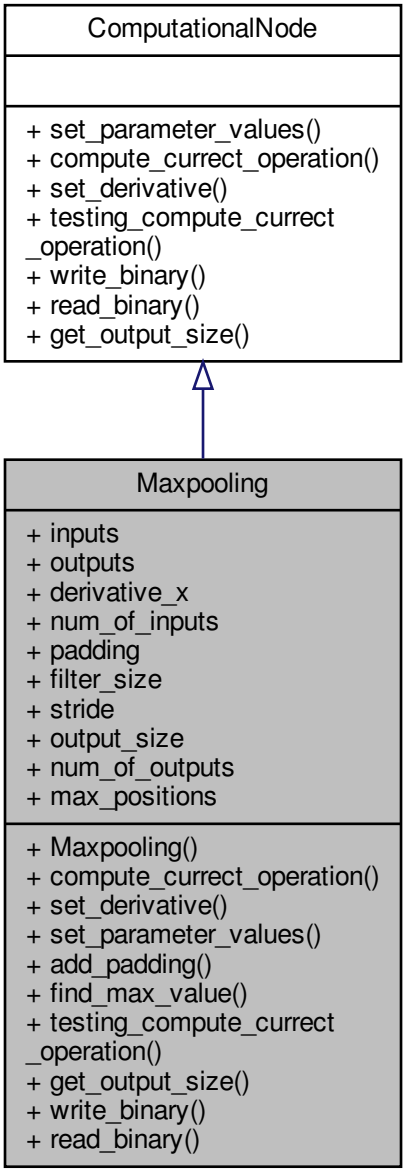
- /home/nehil/dlfsC++/libdl/headers/Activation.h
- /home/nehil/dlfsC++/libdl/src/Activation.cpp

6.9 Maxpooling Class Reference

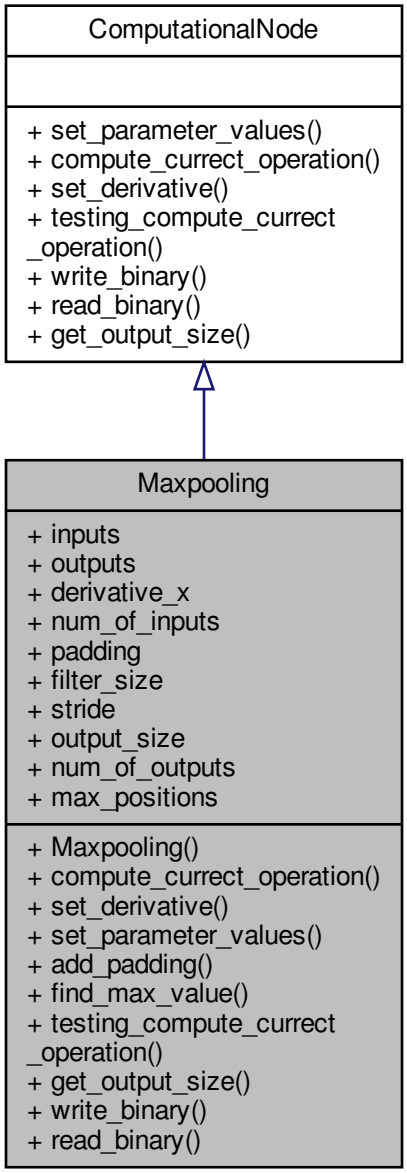
A class which implements the maxpooling operation in the network.

```
#include <Maxpooling.h>
```

Inheritance diagram for Maxpooling:



Collaboration diagram for Maxpooling:



Public Member Functions

- Maxpooling (int input_size, int tmp_num_of_inputs, int tmp_output_size, int tmp_num_of_outputs, int tmp_↩ filter_size, int tmp_stride, int tmp_padding)
- std::vector< Eigen::MatrixXd > compute_currect_operation (std::vector< Eigen::MatrixXd > input) override
- std::vector< Eigen::MatrixXd > set_derivative (std::vector< Eigen::MatrixXd > prev_derivative) override
- void set_parameter_values (double learning_rate, int batch_size) override
- Eigen::MatrixXd add_padding (int index, const Eigen::MatrixXd &tmp_input)
- double find_max_value (int index, size_t row_start, size_t col_start)

- std::vector< Eigen::MatrixXd > testing_compute_currect_operation (std::vector< Eigen::MatrixXd > tmp_↩ input) override
- std::array< int, 3 > get_output_size () override
- void write_binary (std::ofstream &out) override
- void read_binary (std::ifstream &in) override

Public Attributes

- std::vector< Eigen::MatrixXd > inputs

  Input to the maxpooling layer. Consist of one or multiple feature maps.
- std::vector< Eigen::MatrixXd > outputs

  The output of the mapooling operation. The input and output depts will be the same.
- std::vector< Eigen::MatrixXd > derivative_x

  The derivative of the maxpooling operation.
- int num_of_inputs

  Number of input feature maps.
- int padding

  Padding of the maxpooling operation.
- int filter_size

  The height or the width of the filter which is used for the maxpooling.
- int stride

  The stride of the maxpooling operation.
- int output_size

  The size of the output feature map, after the application of the.
- int num_of_outputs

  the number of outputs
- std::vector< std::vector< std::pair< int, int > > > max_positions

  A vector of vector of integer number pairs to keet the position where the max value is found.

6.9.1 Detailed Description

A class which implements the maxpooling operation in the network.

Definition at line 13 of file Maxpooling.h.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 Maxpooling()

```
Maxpooling::Maxpooling (
            int input_size,
            int tmp_num_of_inputs,
            int tmp_output_size,
            int tmp_num_of_outputs,
            int tmp_filter_size,
            int tmp_stride,
            int tmp_padding )  [explicit]
```

A constructor for the maxpooling operation.

Parameters

input_size	The height or the width of one of the input feature maps.
tmp_num_of_inputs	The number of input feature maps
tmp_output_size	The height or the width of the output of the max pooling operation.
tmp_num_of_outputs	The number of output feature maps, which is the same with the number of input feature maps.
tmp_filter_size	The height or the width of the maxpooling filter
tmp_stride	The stride which is going to be used during the maxpooling operation.
tmp_padding	The padding which is going to be used during the maxpooling operation.

Definition at line 8 of file Maxpooling.cpp.

```
8
                                                    {
9     for (size_t i = 0; i < tmp_num_of_inputs; i++) {
10         this->inputs.emplace_back(Eigen::MatrixXd::Zero(input_size + 2 * tmp_padding, input_size + 2 *
       tmp_padding));
11         this->derivative_x.emplace_back(Eigen::MatrixXd::Zero(input_size + 2 * tmp_padding, input_size +
       2 * tmp_padding));
12     }
13     this->output_size = tmp_output_size;
14
15     for (size_t i = 0; i < tmp_num_of_outputs; i++) {
16         Eigen::MatrixXd output;
17         output = Eigen::MatrixXd::Zero(this->output_size, this->output_size);
18         this->outputs.push_back(output);
19     }
20     this->num_of_inputs = tmp_num_of_inputs;
21     this->filter_size = tmp_filter_size;
22     this->padding = tmp_padding;
23     this->stride = tmp_stride;
24     this->num_of_outputs = tmp_num_of_outputs;
25
26 }
```

6.9.3 Member Function Documentation

6.9.3.1 add_padding()

```
Eigen::MatrixXd Maxpooling::add_padding (
            int index,
            const Eigen::MatrixXd & tmp_input )
```

A function to add padding to the input tensor, before the maxpooling process.

Parameters

index	an integer argument shows the index of the input.
tmp_input	an Egen matrix for the input.

Returns

a channel of the input with padding.

Definition at line 28 of file Maxpooling.cpp.

```
28                                                                                  {
29     this->inputs[index].block(this->padding, this->padding, tmp_input.rows(), tmp_input.cols()) =
       tmp_input;
30     return this->inputs[index];
31 }
```

6.9.3.2 compute_currect_operation()

```
std::vector< Eigen::MatrixXd > Maxpooling::compute_currect_operation (
            std::vector< Eigen::MatrixXd > input )  [override], [virtual]
```

A function to calculate the forward pass of the maxpooling operation.

Parameters

input	incoming feature maps

Returns

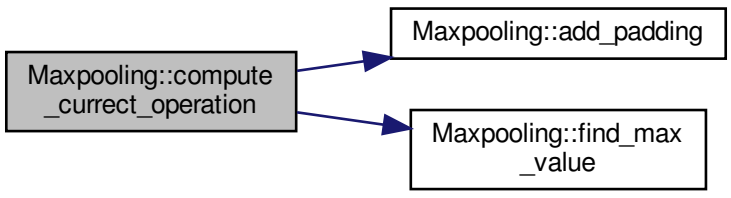
output of the maxpooling operation which has the same depth with the input.

Implements ComputationalNode.

Definition at line 49 of file Maxpooling.cpp.

```
49                                                                                                  {
50
51     for(size_t i = 0; i < tmp_input.size(); i++) {
52         this->inputs[i] = add_padding(i, tmp_input[i]);
53     }
54
55     size_t row_start_idx = 0;
56     size_t col_start_idx = 0;
57
58     size_t output_idx = 0;
59
60     for(size_t input_idx = 0; input_idx < this->num_of_inputs; input_idx ++) {
61         row_start_idx = 0;
62         for(size_t row_idx = 0 ; row_idx < this->outputs[output_idx].rows(); row_idx ++) {
63             col_start_idx = 0;
64             for(size_t col_idx = 0; col_idx < this->outputs[output_idx].cols(); col_idx ++) {
65                 this->outputs[output_idx](row_idx, col_idx) = find_max_value(input_idx, row_start_idx,
       col_start_idx);
66                 col_start_idx += this->stride;
67             }
68             row_start_idx += this->stride;
69         }
70         output_idx ++;
71     }
72     return this->outputs;
73 }
```

Here is the call graph for this function:



6.9.3.3 find_max_value()

```
double Maxpooling::find_max_value (
            int index,
            size_t row_start,
            size_t col_start )
```

A function to find the max values in a particaular block.

Parameters

index	an integer argument, the index of the input.
row_start	an integer argument, row start of the block.
col_start	an integer argument, column start of the block

Returns

the max index of the block.

Definition at line 33 of file Maxpooling.cpp.

```
33                                                                                  {
34     Eigen::MatrixXi::Index maxRow, maxCol;
35     double max = this->inputs[index].block(row_start, col_start, this->filter_size,
       this->filter_size).maxCoeff(&maxRow, &maxCol);
36     if(this->max_positions.size() <= index) {
37         std::vector< std::pair<int, int> > tmp;
38         tmp.emplace_back(std::make_pair(maxRow + row_start, maxCol + col_start));
39         this->max_positions.emplace_back(tmp);
40     }
41     else {
42         this->max_positions[index].emplace_back(std::make_pair(maxRow + row_start, maxCol + col_start));
43     }
44
45     return max;
46 }
```

6.9.3.4 get_output_size()

```
std::array<int, 3> Maxpooling::get_output_size ( )  [inline], [override], [virtual]
```

A function to return the output size of this computational node.

Returns

respectively, number of outputs, height of the output, and the width of the output.

Implements ComputationalNode.

Definition at line 91 of file Maxpooling.h.

```
91                                          { return std::array<int, 3> { this->num_of_outputs,
       this->output_size,
92                                                                        this->output_size};}
```

6.9.3.5 read_binary()

```
void Maxpooling::read_binary (
            std::ifstream & in )  [inline], [override], [virtual]
```

A virtual function which will be override by the child classes.

- This function is for loading the weights and the biases before the training of the network starts.

Parameters

in	an ifstream object

Implements ComputationalNode.

Definition at line 94 of file Maxpooling.h.

```
94 {};
```

6.9.3.6 set_derivative()

```
std::vector< Eigen::MatrixXd > Maxpooling::set_derivative (
            std::vector< Eigen::MatrixXd > prev_derivative )  [override], [virtual]
```

A function to calculate the backward pass of the max pooling operation

Parameters

prev_derivative	The downflowing derivative

Returns

The multiplication of the down flowing derivative and the derivative of the max pooling operation.

Implements ComputationalNode.

Definition at line 112 of file Maxpooling.cpp.

```
112                                                                                                    {
113
114     for(size_t i = 0; i < this->num_of_inputs; i++) {
115         this->derivative_x[i].setZero();
116     }
117
118     for(size_t i = 0; i < prev_derivative.size(); i++) {
119         size_t counter = 0;
120         for(size_t der_i = 0 ; der_i < prev_derivative[i].rows(); der_i++) {
121             for(size_t der_j = 0 ; der_j < prev_derivative[i].cols(); der_j++) {
122                 std::pair<int, int> position = this->max_positions[i][counter];
123                 this->derivative_x[i](position.first, position.second) = prev_derivative[i](der_i,
      der_j);
124                 counter++;
125             }
126         }
127     }
128
129     for(size_t i = 0; i < this->max_positions.size(); i++) {
130         this->max_positions[i].clear();
131     }
132     this->max_positions.clear();
133
134
135     return this->derivative_x;
136 }
```

6.9.3.7 set_parameter_values()

```
void Maxpooling::set_parameter_values (
            double learning_rate,
            int batch_size )  [override], [virtual]
```

A function to set all the class variables back to its initial values during the update step of the training.

Parameters

learning_rate	a double argument, learning rate
batch_size	an integer argument, number of samples in one batch.

Implements ComputationalNode.

Definition at line 102 of file Maxpooling.cpp.

```
102                                                                                 {
103
104     for(size_t i = 0; i < this->num_of_inputs; i++) {
105         this->inputs[i].setZero();
106         this->derivative_x[i].setZero();
107         this->outputs[i].setZero();
108     }
109 }
```

6.9.3.8 testing_compute_currect_operation()

```
std::vector< Eigen::MatrixXd > Maxpooling::testing_compute_currect_operation (
            std::vector< Eigen::MatrixXd > tmp_input )  [override], [virtual]
```

A function to run the forward pass of the convolution operation during the testing phase.

Parameters

tmp_input	a vector of Eigen matrices, an input coming from the previous computational node.

Returns

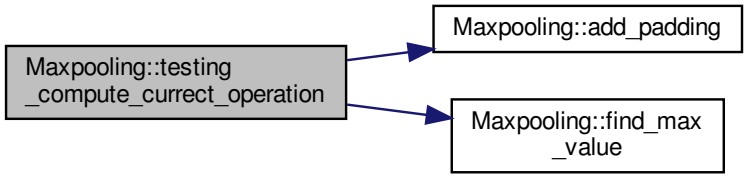
a vector of Eigen matrices, returns the values of the input after the convolution operation.

Implements ComputationalNode.

Definition at line 76 of file Maxpooling.cpp.

```
76
      {
77     for(size_t i = 0; i < tmp_input.size(); i++) {
78         this->inputs[i] = add_padding(i, tmp_input[i]);
79     }
80
81     size_t row_start_idx = 0;
82     size_t col_start_idx = 0;
83
84     size_t output_idx = 0;
85
86     for(size_t input_idx = 0; input_idx < this->num_of_inputs; input_idx ++) {
87         row_start_idx = 0;
88         for(size_t row_idx = 0 ; row_idx < this->outputs[output_idx].rows(); row_idx ++) {
89             col_start_idx = 0;
90             for(size_t col_idx = 0; col_idx < this->outputs[output_idx].cols(); col_idx ++) {
91                 this->outputs[output_idx](row_idx, col_idx) = find_max_value(input_idx, row_start_idx,
      col_start_idx);
92                 col_start_idx += this->stride;
93             }
94             row_start_idx += this->stride;
95         }
96         output_idx ++;
97     }
98     return this->outputs;
99 }
```

Here is the call graph for this function:



6.9.3.9 write_binary()

```
void Maxpooling::write_binary (
            std::ofstream & out )  [inline], [override], [virtual]
```

A virtual function which will be override by the child classes. This function is for saving the weights and the biases after the training of the network.

Parameters

out	an ofstream object

Implements ComputationalNode.

Definition at line 93 of file Maxpooling.h.

```
93 {};
```

6.9.4 Member Data Documentation

6.9.4.1 derivative_x

```
std::vector<Eigen::MatrixXd> Maxpooling::derivative_x
```

The derivative of the maxpooling operation.

Definition at line 17 of file Maxpooling.h.

6.9.4.2 filter_size

```
int Maxpooling::filter_size
```

The height or the width of the filter which is used for the maxpooling.

Definition at line 21 of file Maxpooling.h.

6.9.4.3 inputs

```
std::vector<Eigen::MatrixXd> Maxpooling::inputs
```

Input to the maxpooling layer. Consist of one or multiple feature maps.

Definition at line 15 of file Maxpooling.h.

6.9.4.4 max_positions

```
std::vector<std::vector<std::pair<int, int> > > Maxpooling::max_positions
```

A vector of vector of integer number pairs to keet the position where the max value is found.

Definition at line 25 of file Maxpooling.h.

6.9.4.5 num_of_inputs

```
int Maxpooling::num_of_inputs
```

Number of input feature maps.

Definition at line 19 of file Maxpooling.h.

6.9.4.6 num_of_outputs

```
int Maxpooling::num_of_outputs
```

the number of outputs

Definition at line 24 of file Maxpooling.h.

6.9.4.7 output_size

```
int Maxpooling::output_size
```

The size of the output feature map, after the application of the.

Definition at line 23 of file Maxpooling.h.

6.9.4.8 outputs

```
std::vector<Eigen::MatrixXd> Maxpooling::outputs
```

The output of the mapooling operation. The input and output depts will be the same.

Definition at line 16 of file Maxpooling.h.

6.9.4.9 padding

```
int Maxpooling::padding
```

Padding of the maxpooling operation.

Definition at line 20 of file Maxpooling.h.

6.9.4.10 stride

```
int Maxpooling::stride
```

The stride of the maxpooling operation.

Definition at line 22 of file Maxpooling.h.

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

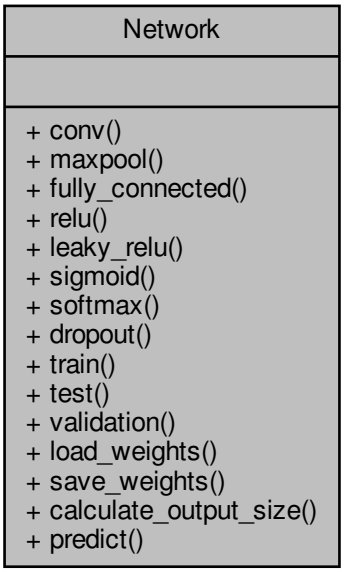
- /home/nehil/dlfsC++/libdl/headers/Maxpooling.h
- /home/nehil/dlfsC++/libdl/src/Maxpooling.cpp

6.10 Network Class Reference

A class which contains all functions to build the network, do training, validation and testing.

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

Collaboration diagram for Network:



Public Member Functions

- std::pair< int, int > conv (int input_size, int num_of_inputs, int filter_size, int filter_depth, int num_of_filters, int stride, int padding)
- std::pair< int, int > maxpool (int input_size, int num_of_inputs, int filter_size, int stride, int padding)
- std::array< int, 3 > fully_connected (int in_h, int in_w, int num_of_inputs, int out)
- std::array< int, 3 > relu (int num_of_inputs, int node_size, int node_size2)
- std::array< int, 3 > leaky_relu (int num_of_inputs, int node_size, int node_size2)
- std::array< int, 3 > sigmoid (int num_of_inputs, int node_size, int node_size2)
- std::array< int, 3 > softmax (int num_of_inputs, int node_size, int node_size2)
- std::array< int, 3 > dropout (int in_h, int num_of_inputs, int out, double drop_out_value)
- std::vector< double > train (std::vector< std::vector< Eigen::MatrixXd >> samples, std::vector< Eigen::↩ MatrixXd > targets, std::string error_func, int epoch, int batch_size, double learning_rate, double learning↩ _rate_decay)
- double test (std::vector< std::vector< Eigen::MatrixXd >> &input, std::vector< Eigen::MatrixXd > &target)
- double validation (std::vector< std::vector< Eigen::MatrixXd >> samples, std::vector< Eigen::MatrixXd > targets, int batch_size)
- void load_weights (const std::string &file_path)
- void save_weights (const std::string &file_path)
- int calculate_output_size (int input_size, int tmp_filter_size, int tmp_stride, int tmp_padding)
- int predict (std::vector< Eigen::MatrixXd > &input)

6.10.1 Detailed Description

A class which contains all functions to build the network, do training, validation and testing.

This class contains different functions to add new computational nodes to the computational graph. End user can directly reach these functions and build up their own network structure. Also after the generation of the network, end user will have functions to train, validate and test the network.

Definition at line 35 of file Network.h.

6.10.2 Member Function Documentation

6.10.2.1 calculate_output_size()

```
int Network::calculate_output_size (
            int input_size,
            int tmp_filter_size,
            int tmp_stride,
            int tmp_padding )
```

A function which calculates the output size of a max pooling or coonvolution operation.

Parameters

input_size	an integer argument, the height or the width of the input square matrix.
tmp_filter_size	an integer argument, the height or width of the square convolution or max pooling kernel.
tmp_stride	an integer argument, the stride of the convolution or max pooling operation.
tmp_padding	an integer argument, the padding of the convolution or max pooling operation.

Returns

an integer, the size of the output square matrix.

Definition at line 211 of file Network.cpp.

```
211                                                                                                                    {
212     return (input_size - tmp_filter_size + 2 * tmp_padding) / tmp_stride + 1;
213 }
```

6.10.2.2 conv()

```
std::pair< int, int > Network::conv (
            int input_size,
            int num_of_inputs,
            int filter_size,
            int filter_depth,
            int num_of_filters,
            int stride,
            int padding )
```

A function to create a new convolution layer in the network.

Parameters

input_size	an integer argument, the size of one side of the square input.
num_of_inputs	an integer argument, depth of the input.
filter_size	an integer argument, the size of one side of the square convolution filter, if the filter $3 * 3$, then this parameter must be 3.
filter_depth	an integer argument, the depth of the convolution filter, is must be equal to the depth of the input.
num_of_filters	an integer argument, the number of filters will be used in the convolution operation.
stride	an integer argument, the stride of the convolution.
padding	an integer argument, the padding of the convolution.

Returns

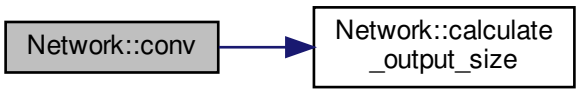
return value will be a pair of integers, which shows respectively the size of the output feature map, and the number of output feature maps.

Definition at line 9 of file Network.cpp.

```
10                                                                                                                     {
11
12     if(filter_depth != num_of_inputs) {
13         throw std::invalid_argument("Expected the input dept and the filter depth to be equal!");
14     }
15     if(filter_size > input_size + 2 * padding) {
16         throw std::invalid_argument("Expected given filter size to be smaller than the input size!");
17     }
18
19     if(input_size < 1 || num_of_inputs < 1 || filter_size < 1 || filter_depth < 1 || num_of_filters < 1)
       {
20         throw std::invalid_argument("Expected input_size, num_of_inputs, filter_size, filter_depth, "
21                                     "num_of_filters of the convolution layer bigger than or equal to
       1!");
22     }
23
24     if(stride < 0 || padding < 0 ) {
25         throw std::invalid_argument("Expected stride and padding parameters to be bigger than or equal to
       0!");
```

```
26      }
27
28      if(!this->computationalGraph.empty()) {
29          std::array<int, 3> output_of_the_previous_layer =
        this->computationalGraph.back()->get_output_size();
30          if(output_of_the_previous_layer[0] != num_of_inputs || output_of_the_previous_layer[1] !=
        input_size
31          || output_of_the_previous_layer[2] != input_size) {
32              throw std::invalid_argument("Expected to match the output size of the previous layer with "
33                                          "the input size of the convolution layer!");
34          }
35      }
36
37      int output_size = calculate_output_size(input_size, filter_size, stride, padding);
38      this->computationalGraph.push_back(new Convolution(input_size, num_of_inputs, output_size,
        num_of_filters,
39              filter_size, filter_depth, num_of_filters, stride, padding));
40      return std::make_pair(output_size, num_of_filters);
41
42  }
```

Here is the call graph for this function:



6.10.2.3 dropout()

```
std::array< int, 3 > Network::dropout (
              int in_h,
              int num_of_inputs,
              int out,
              double drop_out_value )
```

A function to add the dropout computational node to the network. Dropout function can only be used in the fully connected part of the network.

Parameters

in_h	an integer argument, the height of the input
num_of_inputs	an integer argument, the depth of the input variable.
out	an integer argument, the output height of the input.
drop_out_value	a double argument, the propability of dropout.

Returns

respectively, the output height, the output width, and the depth of the output is going to be returned.

Definition at line 173 of file Network.cpp.

```
173                                                                                 {
```

```
174     if(in_w != 1 || num_of_inputs != 1) {
175         throw std::invalid_argument("Dropout function cannot be used in the convolution layers.");
176     }
177 
178     if(!this->computationalGraph.empty()) {
179         std::array<int, 3> output_of_the_previous_layer =
        this->computationalGraph.back()->get_output_size();
180         if(output_of_the_previous_layer[0] != num_of_inputs || output_of_the_previous_layer[1] != out
181            || output_of_the_previous_layer[2] != in_w) {
182             throw std::invalid_argument("Expected to match the output size of the previous layer with "
183                                         "the input size of the dropout layer!");
184         }
185     }
186 
187     this->computationalGraph.push_back(new Dropout(out, drop_out_value));
188     std::array<int, 3> output = {out, 1, num_of_inputs};
189     return output;
190 }
```

6.10.2.4 fully_connected()

```
std::array< int, 3 > Network::fully_connected (
            int in_h,
            int in_w,
            int num_of_inputs,
            int out )
```

A function to add fully connected layer in the network.

Parameters

in_h	an integer argument, the height of the input variable.
in_w	an integer argument, the width of the input variable.
num_of_inputs	an input argument, the depth of the input variable.
out	an integer argument, the height of the output layer.

Returns

respectively, the output height, the output width, and the depth of the output is going to be returned.

Definition at line 44 of file Network.cpp.

```
44                                                                                                  {
45 
46     if(in_h < 1 || num_of_inputs < 1 || in_w < 1 || out < 1 ) {
47         throw std::invalid_argument("Expected all the parameters of the convolution layer to be bigger
       than or equal"
48                                     " to 1.");
49     }
50 
51     if(!this->computationalGraph.empty()) {
52         std::array<int, 3> output_of_the_previous_layer =
       this->computationalGraph.back()->get_output_size();
53         if(output_of_the_previous_layer[0] != num_of_inputs) {
54             throw std::invalid_argument("Expected to match the output depth of the previous layer with
       the input depth"
55                                         "of the fully connected layer.!");
56         }
57     }
58     if(in_w > 1) {
59         this->computationalGraph.emplace_back(new Connector(in_h, num_of_inputs));
60     }
61 
62     std::array<int, 3> output_of_the_previous_layer = this->computationalGraph.back()->get_output_size();
63     if(output_of_the_previous_layer[1] != in_h * in_w * num_of_inputs) {
64         throw std::invalid_argument("Expected the output of the flattening layer to be equal to the
       multiplication of"
65                                     " the height width and depth of the fully connected layer!");
```

```
66      }
67      this->computationalGraph.push_back(new DotProduct(out, in_h * in_w * num_of_inputs));
68      this->computationalGraph.push_back(new AddNode(out));
69
70      std::array<int, 3> output = {out, 1, 1};
71      return output;
72  }
```

6.10.2.5 leaky_relu()

```
std::array< int, 3 > Network::leaky_relu (
              int num_of_inputs,
              int node_size,
              int node_size2 )
```

A function to add the leaky relu activation function to the network.

Parameters

num_of_inputs	an integer argument, the depth of the input variable.
node_size	an integer argument, the height of the input.
node_size2	an integer argument, the width of the input.

Returns

respectively, the output height, the output width, and the depth of the output is going to be returned.

Definition at line 119 of file Network.cpp.

```
119                                                                                                                 {
120
121     if(!this->computationalGraph.empty()) {
122         std::array<int, 3> output_of_the_previous_layer =
        this->computationalGraph.back()->get_output_size();
123         if(output_of_the_previous_layer[0] != num_of_inputs || output_of_the_previous_layer[1] !=
        node_size
124             || output_of_the_previous_layer[2] != node_size2) {
125              throw std::invalid_argument("Expected to match the output size of the previous layer with "
126                                          "the input size of the leaky relu activation layer!");
127         }
128     }
129
130     this->computationalGraph.emplace_back(new LeakyReLUActivation(num_of_inputs, node_size,
        node_size2));
131     std::array<int, 3> output = {node_size, node_size2, num_of_inputs};
132     return output;
133 }
```

6.10.2.6 load_weights()

```
void Network::load_weights (
              const std::string & file_path )
```

A function to load the pre-trained weights to the network. This function can be invoked by user from Python files.

Parameters

file_path	a string argument, a path to the location of the txt file where the pre-trained weights and biases will be written from.

Definition at line 201 of file Network.cpp.

```
201                                                 {
202     std::ifstream in(file_path, std::ios::in | std::ios::binary);
203     for(auto operation : this->computationalGraph ) {
204         operation->read_binary(in);
205     }
206
207     in.close();
208
209 }
```

6.10.2.7 maxpool()

```
std::pair< int, int > Network::maxpool (
            int input_size,
            int num_of_inputs,
            int filter_size,
            int stride,
            int padding )
```

A function to create a new max pooling layer in the network. Maxpooling operation does not change the depth of the input of the layer.

Parameters

input_size	an integer argument, the size of one side of the square input.
num_of_inputs	an integer argument, depth of the input.
filter_size	an integer argument, the size of one side of the square convolution filter, if the filter $3 * 3$, then this parameter must be 3.
stride	an integer argument, the stride of the maxpooling operation.
padding	an integer argument, the padding of the maxpooling.

Returns

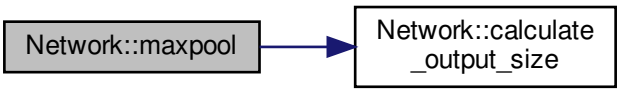
return value will be a pair of integers, which shows respectively the size of the output feature map, and the number of output feature maps.

Definition at line 74 of file Network.cpp.

```
74
          {
75
76     if(filter_size > input_size + 2 * padding) {
77         throw std::invalid_argument("Expected given filter size to be smaller than the input size!");
78     }
79
80     if(input_size < 1 || num_of_inputs < 1 || filter_size < 1) {
81         throw std::invalid_argument("Expected input_size, num_of_inputs,"
82                                     " num_of_filters of the maxpooling layer bigger than or equal to
       1!");
83     }
84     if(stride < 0 || padding < 0 ) {
85         throw std::invalid_argument("Expected stride and padding parameters to be bigger than or equal to
       0!");
86     }
87
88     if(!this->computationalGraph.empty()) {
```

```
89          std::array<int, 3> output_of_the_previous_layer =
        this->computationalGraph.back()->get_output_size();
90          if(output_of_the_previous_layer[0] != num_of_inputs || output_of_the_previous_layer[1] !=
        input_size
91              || output_of_the_previous_layer[2] != input_size) {
92              throw std::invalid_argument("Expected to match the output size of the previous layer with "
93                                          "the input size of the maxpooling layer!");
94          }
95      }
96      int output_size = calculate_output_size(input_size, filter_size, stride, padding);
97      this->computationalGraph.push_back(new Maxpooling(input_size, num_of_inputs, output_size,
        num_of_inputs,
98              filter_size, stride, padding));
99      return std::make_pair(output_size, num_of_inputs);
100 }
```

Here is the call graph for this function:



6.10.2.8 predict()

```
int Network::predict (
              std::vector< Eigen::MatrixXd > & input )
```

Definition at line 402 of file Network.cpp.

```
402                                                 {
403     Eigen::MatrixXd prediction;
404     prediction = forward_pass(input, 1);
405     Eigen::Index max, min;
406     prediction.maxCoeff(&max, &min);
407     int result = 1;
408     if(max == 0) result = 0;
409     return result;
410 }
```

6.10.2.9 relu()

```
std::array< int, 3 > Network::relu (
              int num_of_inputs,
              int node_size,
              int node_size2 )
```

A function to add the relu activation function to the network.

Parameters

num_of_inputs	an integer argument, the depth of the input variable.
node_size	an integer argument, the height of the input.
node_size2	an integer argument, the width of the input.

Returns

respectively, the output height, the output width, and the depth of the output is going to be returned.

Definition at line 102 of file Network.cpp.

```
102                                                                                                                 {
103
104     if(!this->computationalGraph.empty()) {
105         std::array<int, 3> output_of_the_previous_layer =
        this->computationalGraph.back()->get_output_size();
106         if(output_of_the_previous_layer[0] != num_of_inputs || output_of_the_previous_layer[1] !=
        node_size
107             || output_of_the_previous_layer[2] != node_size2) {
108              throw std::invalid_argument("Expected to match the output size of the previous layer with "
109                                          "the input size of the relu activation layer!");
110         }
111     }
112
113
114     this->computationalGraph.emplace_back(new ReLUActivation(num_of_inputs, node_size, node_size2));
115     std::array<int, 3> output = {node_size, node_size2, num_of_inputs};
116     return output;
117 }
```

6.10.2.10 save_weights()

```
void Network::save_weights (
            const std::string & file_path )
```

A function to save the weights and biases a file. This function can be invoked by user from Python files.

Parameters

file_path	a string argument, a path to the location of the txt file where the weights and biases will be loaded. The file does not need to be exists.

Definition at line 192 of file Network.cpp.

```
192                                                 {
193     std::ofstream out(file_path, std::ios::out | std::ios::binary | std::ios::trunc);
194     for(auto operation : this->computationalGraph ) {
195         operation->write_binary(out);
196     }
197
198     out.close();
199 }
```

6.10.2.11 sigmoid()

```
std::array< int, 3 > Network::sigmoid (
            int num_of_inputs,
            int node_size,
            int node_size2 )
```

A function to add the sigmoid activation function to the network.

Parameters

num_of_inputs	an integer argument, the depth of the input variable.
node_size	an integer argument, the height of the input.
node_size2	an integer argument, the width of the input.

Returns

respectively, the output height, the output width, and the depth of the output is going to be returned.

Definition at line 154 of file Network.cpp.

```
154                                                                                                                 {
155     if(node_size2 != 1 || num_of_inputs != 1) {
156         throw std::invalid_argument("Sigmoid function cannot be used in the convolution layers.");
157     }
158
159     if(!this->computationalGraph.empty()) {
160         std::array<int, 3> output_of_the_previous_layer =
        this->computationalGraph.back()->get_output_size();
161         if(output_of_the_previous_layer[0] != num_of_inputs || output_of_the_previous_layer[1] !=
        node_size
162             || output_of_the_previous_layer[2] != node_size2) {
163             throw std::invalid_argument("Expected to match the output size of the previous layer with "
164                                         "the input size of the sigmoid activation layer!");
165         }
166     }
167
168     this->computationalGraph.emplace_back(new SigmoidActivation(node_size));
169     std::array<int, 3> output = {node_size, node_size2, num_of_inputs};
170     return output;
171 }
```

6.10.2.12 softmax()

```
std::array< int, 3 > Network::softmax (
            int num_of_inputs,
            int node_size,
            int node_size2 )
```

A function to add the softmax activation function to the network.

Parameters

num_of_inputs	an integer argument, the depth of the input variable.
node_size	an integer argument, the height of the input.
node_size2	an integer argument, the width of the input.

Returns

respectively, the output height, the output width, and the depth of the output is going to be returned.

Definition at line 135 of file Network.cpp.

```
135                                                                                                                 {
136     if(node_size2 != 1 || num_of_inputs != 1) {
137         throw std::invalid_argument("Softmax function cannot be used in the convolution layers.");
138     }
139
140     if(!this->computationalGraph.empty()) {
141         std::array<int, 3> output_of_the_previous_layer =
        this->computationalGraph.back()->get_output_size();
142         if(output_of_the_previous_layer[0] != num_of_inputs || output_of_the_previous_layer[1] !=
        node_size
143             || output_of_the_previous_layer[2] != node_size2) {
144             throw std::invalid_argument("Expected to match the output size of the previous layer with "
145                                         "the input size of the softmax activation layer!");
146         }
147     }
148
149     this->computationalGraph.emplace_back(new SoftmaxActivation(node_size));
150     std::array<int, 3> output = {node_size, node_size2, num_of_inputs};
151     return output;
152 }
```

6.10.2.13 test()

```
double Network::test (
            std::vector< std::vector< Eigen::MatrixXd >> & input,
            std::vector< Eigen::MatrixXd > & target )
```

A function to test the network.

Parameters

input	a vector of vectors of Eigen matrices, testing samples.
target	a vector of Eigen matrices, ground truth testing labels.

Returns

the accuracy of the given samples' test results.

Definition at line 412 of file Network.cpp.

```
412                                                                                                                 {
413
414
415     int correct_guesses_total = 0;
416
417     for(size_t i = 0; i < input.size(); i++) {
418         Eigen::MatrixXd prediction;
419         std::vector<Eigen::MatrixXd> input_to_forward;
420         input_to_forward = input[i];
421         prediction = forward_pass(input_to_forward, 1);
422
423         Eigen::Index max, min;
424         prediction.maxCoeff(&max, &min);
425
426         Eigen::Index max_t, min_t;
427         target[i].maxCoeff(&max_t, &min_t);
428
429         if(max == max_t && min == min_t) {
430             correct_guesses_total ++;
431         }
432     }
433
434     return correct_guesses_total;
435 }
```

6.10.2.14 train()

```
std::vector< double > Network::train (
            std::vector< std::vector< Eigen::MatrixXd >> samples,
            std::vector< Eigen::MatrixXd > targets,
            std::string error_func,
            int epoch,
            int batch_size,
            double learning_rate,
            double learning_rate_decay )
```

A function for training of the network.

Parameters

samples	a vector of vectors of Eigen matrices, training samples.
targets	a vector of Eigen matrices, ground truth training labels.

Parameters

error_func	a string argument, the name of the error function [leastSquaresError or crossEntropyError]
epoch	an integer argument, number of epoch
batch_size	an integer argument, the number of batch size.
learning_rate	a double argument, the learning rate.
learning_rate_decay	a double argument, the learning rate decay.

Returns

vector of double values, it will return the batch cost, and the batch accuracy

Definition at line 318 of file Network.cpp.

```
321                                                 {
322
323
324     int correct_guesses = 0 ;
325     int correct_guesses_per_epoch = 0 ;
326     double cost = 0.0;
327     std::vector<double> output_vals;
328     Eigen::MatrixXd input;
329     std::vector<Eigen::MatrixXd> inputs;
330
331
332
333     for(int sample_idx = 0; sample_idx < samples.size(); sample_idx++) { // one batch
334         inputs = samples[sample_idx];
335
336         input = forward_pass(inputs);
337
338         Eigen::Index max, min;
339         input.maxCoeff(&max, &min);
340
341
342         Eigen::Index max_t, min_t;
343         targets[sample_idx].maxCoeff(&max_t, &min_t);
344
345         if(max == max_t && min == min_t && input(0,0) != input(1,0)) {
346             correct_guesses_per_epoch ++;
347             correct_guesses ++;
348         }
349         cost += backprop(input, targets[sample_idx], error_func);
350         inputs.clear();
351
352         //learning_rate -= learning_rate_decay * learning_rate;
353
354     }
355
356     std::vector<Eigen::MatrixXd>().swap(inputs);
357     input.resize(0, 0);
358
359     output_vals.push_back(cost/float(batch_size));
360     output_vals.push_back(correct_guesses * 100 /float(batch_size));
361     set_parameters(learning_rate, batch_size);
362
363     return output_vals;
364 }
```

6.10.2.15 validation()

```
double Network::validation (
            std::vector< std::vector< Eigen::MatrixXd >> samples,
            std::vector< Eigen::MatrixXd > targets,
            int batch_size )
```

A function to validate the network during the training phase.

Parameters

input	a vector of vectors of Eigen matrices, validation samples.
target	a vector of Eigen matrices, ground truth validation labels.
batch_size	an integer argument, the number of elements in one batch.

Returns

the accuracy of the given samples' validation results.

Definition at line 367 of file Network.cpp.

```
368                         {
369
370
371     int correct_guesses = 0 ;
372     int correct_guesses_per_epoch = 0 ;
373     Eigen::MatrixXd input;
374
375
376
377     for(int sample_idx = 0; sample_idx < samples.size(); sample_idx++) { // one batch
378         //inputs = samples[sample_idx];
379
380         input = forward_pass(samples[sample_idx], 1);
381
382         Eigen::Index max, min;
383         input.maxCoeff(&max, &min);
384
385
386         Eigen::Index max_t, min_t;
387         targets[sample_idx].maxCoeff(&max_t, &min_t);
388
389         if(max == max_t && min == min_t) {
390             correct_guesses_per_epoch ++;
391             correct_guesses ++;
392         }
393
394         //learning_rate -= learning_rate_decay * learning_rate;
395
396     }
397
398     input.resize(0, 0);
399     return correct_guesses * 100 / float(samples.size());
400 }
```

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

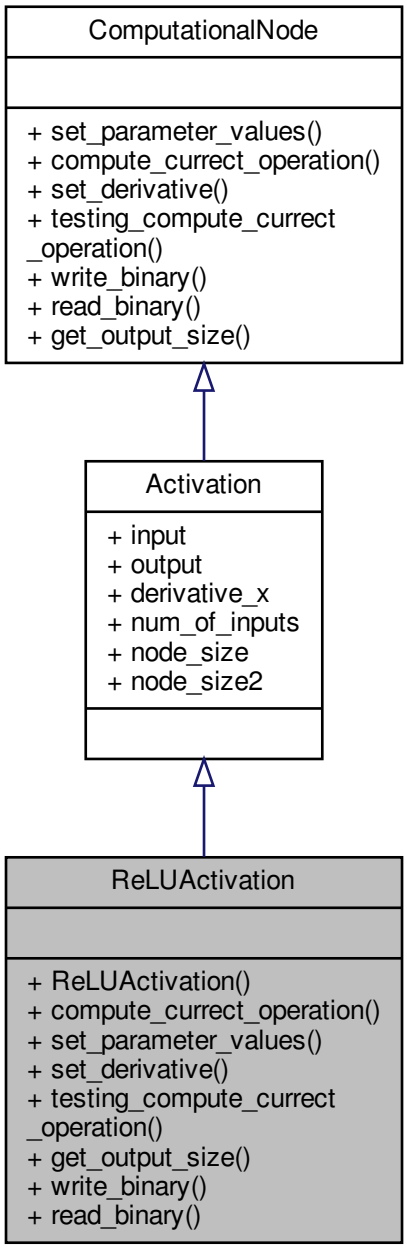
- /home/nehil/dlfsC++/libdl/headers/Network.h
- /home/nehil/dlfsC++/libdl/src/Network.cpp

6.11 ReLUActivation Class Reference

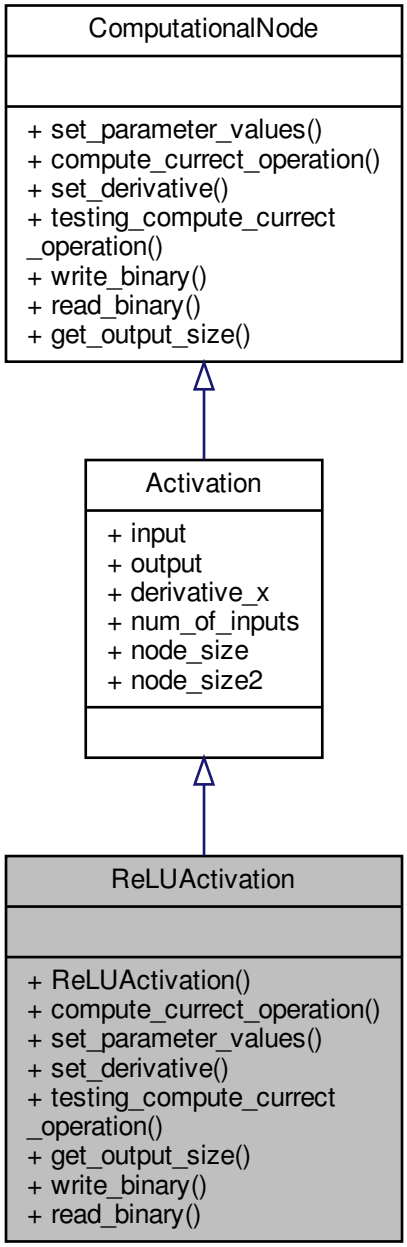
A class for the Rectified Linear Units (RELU) Activation function.

`#include <Activation.h>`

Inheritance diagram for ReLUActivation:



Collaboration diagram for ReLUActivation:



Public Member Functions

- ReLUActivation (int num_of_inputs, int num_nodes1, int num_nodes2)
- std::vector< Eigen::MatrixXd > compute_currect_operation (std::vector< Eigen::MatrixXd > tmp_input) override
- void set_parameter_values (double learning_rate, int batch_size) override
- std::vector< Eigen::MatrixXd > set_derivative (std::vector< Eigen::MatrixXd > prev_derivative) override

- std::vector< Eigen::MatrixXd > testing_compute_currect_operation (std::vector< Eigen::MatrixXd > tmp_input) override
- std::array< int, 3 > get_output_size () override
- void write_binary (std::ofstream &out) override
- void read_binary (std::ifstream &in) override

Additional Inherited Members

6.11.1 Detailed Description

A class for the Rectified Linear Units (RELU) Activation function.

Definition at line 99 of file Activation.h.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 ReLUActivation()

```
ReLUActivation::ReLUActivation (
            int num_of_inputs,
            int num_nodes1,
            int num_nodes2 )  [explicit]
```

A constructor for RELU activation operation. This class can be used for both fully connected layer, and the convolution layers.

Parameters

num_of_inputs	an integer argument, the number of feature maps.
num_nodes1	an integer argument, refers to the height input matrix.
num_nodes2	an integer argument, refers to width of the input matrix.

initialization of the inputs, outputs, and and the derivative vector with respect to the input

Definition at line 59 of file Activation.cpp.

```
59                                                                                {
60
64      this->num_of_inputs = num_of_inputs;
65      this->node_size = num_nodes1;
66      this->node_size2 = num_nodes2;
67      for (int i = 0; i < num_of_inputs; ++i) {
68          this->input.emplace_back(Eigen::MatrixXd::Zero(num_nodes1,num_nodes2));
69          this->output.emplace_back(Eigen::MatrixXd::Zero(num_nodes1,num_nodes2));
70          this->derivative_x.emplace_back(Eigen::MatrixXd::Zero(num_nodes1,num_nodes2));
71      }
72  }
```

6.11.3 Member Function Documentation

6.11.3.1 compute_currect_operation()

```
std::vector< Eigen::MatrixXd > ReLUActivation::compute_currect_operation (
            std::vector< Eigen::MatrixXd > tmp_input )  [override], [virtual]
```

The function is to calculate the the current operation of the RELU node. The RELU activation function is applied to the given input. RELU operation kills half of the neurons, because it a value is smaller than 0, then the output will be zero. Else it will be the value itself.

Parameters

tmp_input	a vector of Eigen matrices, an input coming from the previous computational node.

Returns

a vector of Eigen matrices, returns the values of the input after applying RELU activation function.

< looping through all the input feature maps

< RELU calculation

Implements ComputationalNode.

Definition at line 75 of file Activation.cpp.

```
75                                                                                                                  {
76     this->input = tmp_input;
77     this->output = tmp_input;
78     for(size_t idx = 0; idx < tmp_input.size(); idx ++) {
79         for(size_t i = 0; i < tmp_input[idx].rows(); i++) {
80             for(size_t j = 0; j < tmp_input[idx].cols(); j++) {
81                 if(tmp_input[idx](i, j) <= 0) this->output[idx](i, j) = 0.0;
82             }
83         }
84     }
85
86     return this->output;
87 }
```

6.11.3.2 get_output_size()

```
std::array<int, 3> ReLUActivation::get_output_size ( )  [inline], [override], [virtual]
```

A function to return the output size of this computational node.

Returns

respectively, number of outputs, height of the output, and the width of the output.

Implements ComputationalNode.

Definition at line 154 of file Activation.h.

```
154                                             { return std::array<int, 3> { this->num_of_inputs,
       this->node_size,
155                                                                           this->node_size2};}
```

6.11.3.3 read_binary()

```
void ReLUActivation::read_binary (
             std::ifstream & in )  [inline], [override], [virtual]
```

A virtual function which will be override by the child classes.

- This function is for loading the weights and the biases before the training of the network starts.

Parameters

in	an ifstream object

Implements ComputationalNode.

Definition at line 158 of file Activation.h.

```
158 {};
```

6.11.3.4 set_derivative()

```
std::vector< Eigen::MatrixXd > ReLUActivation::set_derivative (
            std::vector< Eigen::MatrixXd > prev_derivative )  [override], [virtual]
```

A function to calculate the derivative of the RELU operation. Derivative of the previous operations in the network are received as the input paramater. The result is the multiplication of the current derivative of the operation and the input down flow derivative.

Parameters

prev_derivative	a vector of Eigen matrices, down flow derivative through this RELU activation function.

Returns

a vector of Eigen matrices, the multiplication of the current derivative of the operation and the input down flow derivative.

< The derivative of the RELU with respect to the input is set back zero.

< because, in the mini batch calculation, we don't call the update function and this is the only way to set the derivatives back.

Implements ComputationalNode.

Definition at line 114 of file Activation.cpp.

```
114                                                                                                {
115
116     for(size_t i = 0; i < prev_derivative.size(); i++) {
117         derivative_x[i].setZero();
118     }
120
121     for(size_t i = 0; i < prev_derivative.size(); i++) {
122         for (int j = 0; j < prev_derivative[i].rows(); j++) {
123             for (int k = 0; k < prev_derivative[i].cols(); k++) {
124                 if(this->output[i](j,k) == 0) this->derivative_x[i](j, k) = 0.0;
125                 else this->derivative_x[i](j, k) = 1 * prev_derivative[i](j, k);
126             }
127         }
128     }
129     return this->derivative_x;
130 }
```

6.11.3.5 set_parameter_values()

```
void ReLUActivation::set_parameter_values (
            double learning_rate,
            int batch_size )  [override], [virtual]
```

A function to set weights and biases by using the derivatives, which are calculated during the backward pass, and to set input, output and derivative values to zero. Since there is no weight or bias in the activation function layers, it is used only for setting all the input, output and, derivatives to zero.

Parameters

learning_rate	a double argument, learning rate.
batch_size	an integer argument, the number of elements in one batch.

Implements ComputationalNode.

Definition at line 105 of file Activation.cpp.

```
105                                                                                 {
106     for (int i = 0; i < this->input.size(); ++i) {
107         this->derivative_x[i].setZero();
108         this->input[i].setZero();
109         this->output[i].setZero();
110     }
111 }
```

6.11.3.6 testing_compute_currect_operation()

```
std::vector< Eigen::MatrixXd > ReLUActivation::testing_compute_currect_operation (
            std::vector< Eigen::MatrixXd > tmp_input )  [override], [virtual]
```

A function to run the forward pass of the relu activation operation during the testing phase.

Parameters

tmp_input	a vector of Eigen matrices, an input coming from the previous computational node.

Returns

a vector of Eigen matrices, returns the values of the input after the relu activation operation.

Implements ComputationalNode.

Definition at line 90 of file Activation.cpp.

```
91                                                 {
92     this->input = tmp_input;
93     this->output = tmp_input;
94     for(size_t idx = 0; idx < tmp_input.size(); idx ++) {
95         for(size_t i = 0; i < tmp_input[idx].rows(); i++) {
96             for(size_t j = 0; j < tmp_input[idx].cols(); j++) {
97                 if(tmp_input[idx](i, j) <= 0) this->output[idx](i, j) = 0.0;
98             }
99         }
100     }
101
102     return this->output;
103 }
```

6.11.3.7 write_binary()

```
void ReLUActivation::write_binary (
            std::ofstream & out )  [inline], [override], [virtual]
```

A virtual function which will be override by the child classes. This function is for saving the weights and the biases after the training of the network.

Parameters

out	an ofstream object

Implements ComputationalNode.

Definition at line 157 of file Activation.h.

```
157 {};
```

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

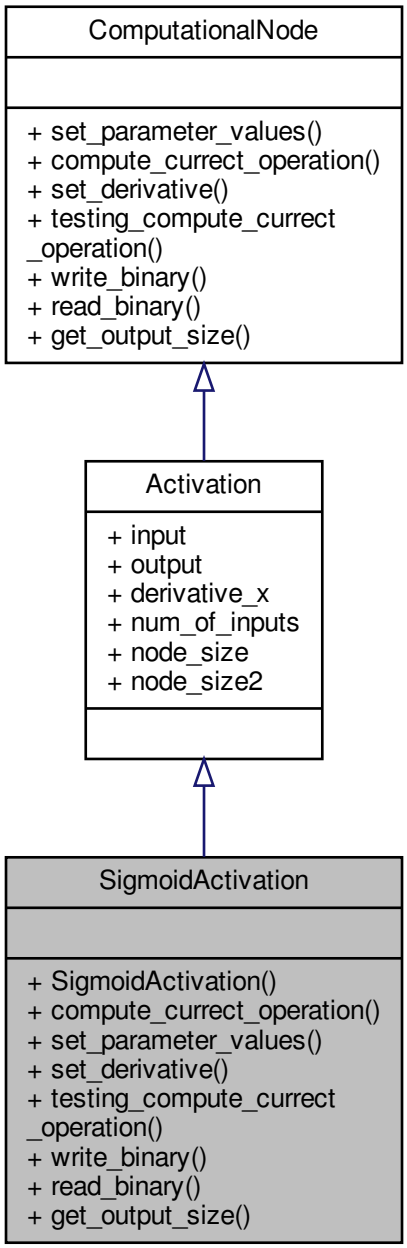
- /home/nehil/dlfsC++/libdl/headers/Activation.h
- /home/nehil/dlfsC++/libdl/src/Activation.cpp

6.12 SigmoidActivation Class Reference

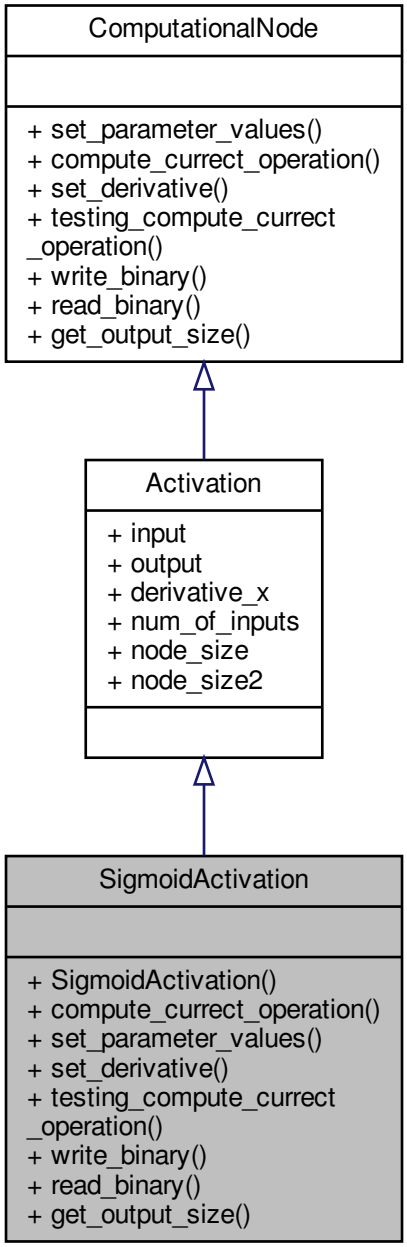
A class for the Sigmoid Activation function.

```
#include <Activation.h>
```

Inheritance diagram for SigmoidActivation:



Collaboration diagram for SigmoidActivation:



Public Member Functions

- SigmoidActivation (int num_nodes)
- std::vector< Eigen::MatrixXd > compute_currect_operation (std::vector< Eigen::MatrixXd > tmp_input) override
- void set_parameter_values (double learning_rate, int batch_size) override
- std::vector< Eigen::MatrixXd > set_derivative (std::vector< Eigen::MatrixXd > prev_derivative) override

- std::vector< Eigen::MatrixXd > testing_compute_currect_operation (std::vector< Eigen::MatrixXd > tmp_input) override
- void write_binary (std::ofstream &out) override
- void read_binary (std::ifstream &in) override
- std::array< int, 3 > get_output_size () override

Additional Inherited Members

6.12.1 Detailed Description

A class for the Sigmoid Activation function.

Definition at line 33 of file Activation.h.

6.12.2 Constructor & Destructor Documentation

6.12.2.1 SigmoidActivation()

```
SigmoidActivation::SigmoidActivation (
            int num_nodes )  [explicit]
```

A constructor for the Sigmoid activation operation. This class is designed only to be used in fully connected layers.

Parameters

num_nodes	an integer argument, which is for the number of input neurons of the activation function layer.

< input of the sigmoid layer, Eigen matrix has the size of num_nodes x 1

< output of the sigmoid layer, Eigen matrix has the size of num_nodes x 1

< derivative of the sigmoid operation wrt. input value,

< Eigen matrix has the size of num_nodes x 1

Definition at line 9 of file Activation.cpp.

```
9                                                {
10      this->input.emplace_back(Eigen::MatrixXd::Zero(num_nodes,1));
11      this->output.emplace_back(Eigen::MatrixXd::Zero(num_nodes,1));
12      this->derivative_x.emplace_back(Eigen::MatrixXd::Zero(num_nodes,1));
13      this->num_of_inputs = 1;
14      this->node_size = num_nodes;
15      this->node_size2 = 1;
17  }
```

6.12.3 Member Function Documentation

6.12.3.1 compute_currect_operation()

```
std::vector< Eigen::MatrixXd > SigmoidActivation::compute_currect_operation (
            std::vector< Eigen::MatrixXd > tmp_input )  [override], [virtual]
```

A function is to calculate the the current operation of the sigmoid node. The Sigmoid activation function is applied to the given input. Sigmoid operation will give an output between 0 and 1. Sigmoid function is a good output activation function in the case of the existance of only two classes.

Parameters

tmp_input	a vector of Eigen matrices, an input coming from the previous computational node.

Returns

a vector of Eigen matrices, returns the values of the input after applying sigmoid activation function.

< The incoming input value is assigned to the input variable of the sigmoid class object, for the calculation of backpropagation.

< The output value also assigned with the same incoming input for now, to make the following calculations easier.

< going through all the neurons of the output layer

< calculate the sigmoid function

Implements ComputationalNode.

Definition at line 20 of file Activation.cpp.

```
20                                                                                                  {
21     this->input = tmp_input;
22     this->output = tmp_input;
23     for(size_t i = 0 ; i < this->output[0].rows(); i++){
24         this->output[0](i) = 1.0/(1 + exp(-1 * this->output[0](i)));
25     }
26     return this->output;
27 }
```

6.12.3.2 get_output_size()

```
std::array<int, 3> SigmoidActivation::get_output_size ( )  [inline], [override], [virtual]
```

A function to return the output size of this computational node.

Returns

respectively, number of outputs, height of the output, and the width of the output.

Implements ComputationalNode.

Definition at line 90 of file Activation.h.

```
90                                          { return std::array<int, 3> { this->num_of_inputs,
       this->node_size,
91                                                                        this->node_size2};}
```

6.12.3.3 read_binary()

```
void SigmoidActivation::read_binary (
             std::ifstream & in )  [inline], [override], [virtual]
```

A virtual function which will be override by the child classes.

- This function is for loading the weights and the biases before the training of the network starts.

Parameters

in	an ifstream object

Implements ComputationalNode.

Definition at line 84 of file Activation.h.

```
84 {};
```

6.12.3.4 set_derivative()

```
std::vector< Eigen::MatrixXd > SigmoidActivation::set_derivative (
            std::vector< Eigen::MatrixXd > prev_derivative )  [override], [virtual]
```

A function to calculate the derivatie of the Sigmoid operation. Derivative of the previous operations in the network are received as the input paramater. The result is the multiplication of the current derivative of the operation and the input down flow derivative.

Parameters

prev_derivative	a vector of Eigen matrices, down flow derivative through this Sigmoid activation function.

Returns

a vector of Eigen matrices, the multiplication of the current derivative of the operation and the input down flow derivative.

< For the sake of the new derivative calculation we need to set the value of the derivative of the input to zero.

< looping through the output neurons of the sigmoid layer.

< calculation of the derivative of sigmoid.

Implements ComputationalNode.

Definition at line 50 of file Activation.cpp.

```
50                                                                                                  {
51     derivative_x[0].setZero();
52     for(size_t i = 0 ; i < this->output[0].rows(); i++) {
53         this->derivative_x[0](i) = this->output[0](i) * (1 - this->output[0](i)) * prev_derivative[0](i);

54     }
55
56     return this->derivative_x;
57 }
```

6.12.3.5 set_parameter_values()

```
void SigmoidActivation::set_parameter_values (
            double learning_rate,
            int batch_size )  [override], [virtual]
```

A function to set weights and biases by using the derivatives, which are calculated during the backward pass, and to set input, output and derivative values to zero. Since there is no weight or bias in the activation function layers, it is used only for setting all the input, output and, derivatives to zero.

Parameters

learning_rate	a double argument, learning rate.
batch_size	an integer argument, the number of elements in one batch.

The input, output variables and the derivative of the computation with respect to the input is set to 0.

Implements ComputationalNode.

Definition at line 40 of file Activation.cpp.

```
40                                                                              {
44     this->derivative_x[0].setZero();
45     this->input[0].setZero();
46     this->output[0].setZero();
47 }
```

6.12.3.6 testing_compute_currect_operation()

```
std::vector< Eigen::MatrixXd > SigmoidActivation::testing_compute_currect_operation (
            std::vector< Eigen::MatrixXd > tmp_input )  [override], [virtual]
```

A function to run the forward pass of the sigmoid activation operation during the testing phase.

Parameters

tmp_input	a vector of Eigen matrices, an input coming from the previous computational node.

Returns

a vector of Eigen matrices, returns the values of the input after the sigmoid activation operation.

Implements ComputationalNode.

Definition at line 29 of file Activation.cpp.

```
30                                                  {
31     this->input = tmp_input;
32     this->output = tmp_input;
33     for(size_t i = 0 ; i < this->output[0].rows(); i++){
34         this->output[0](i) = 1.0/(1 + exp(-1 * this->output[0](i)));
35     }
36     return this->output;
37 }
```

6.12.3.7 write_binary()

```
void SigmoidActivation::write_binary (
            std::ofstream & out )  [inline], [override], [virtual]
```

A virtual function which will be override by the child classes. This function is for saving the weights and the biases after the training of the network.

Parameters

out	an ofstream object

Implements ComputationalNode.

Definition at line 83 of file Activation.h.

```
83 {};
```

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

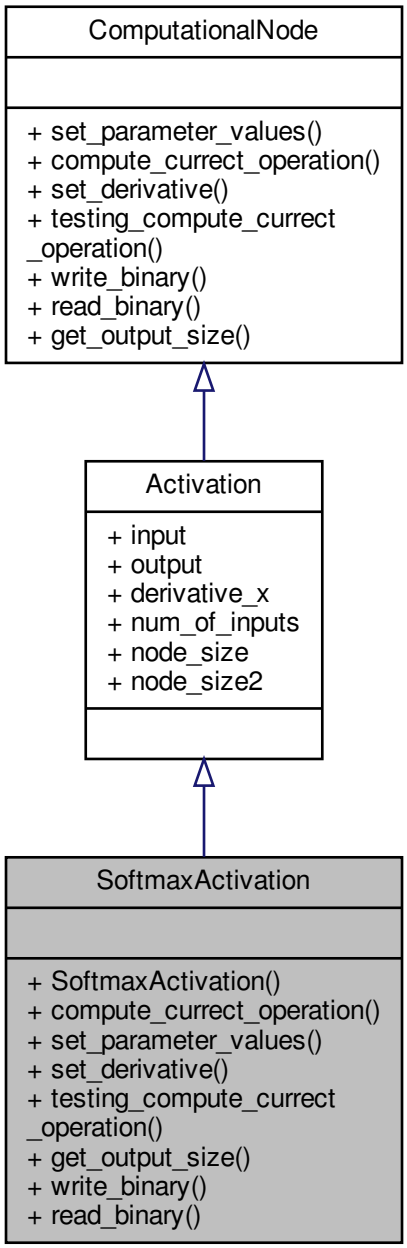
- /home/nehil/dlfsC++/libdl/headers/Activation.h
- /home/nehil/dlfsC++/libdl/src/Activation.cpp

6.13 SoftmaxActivation Class Reference

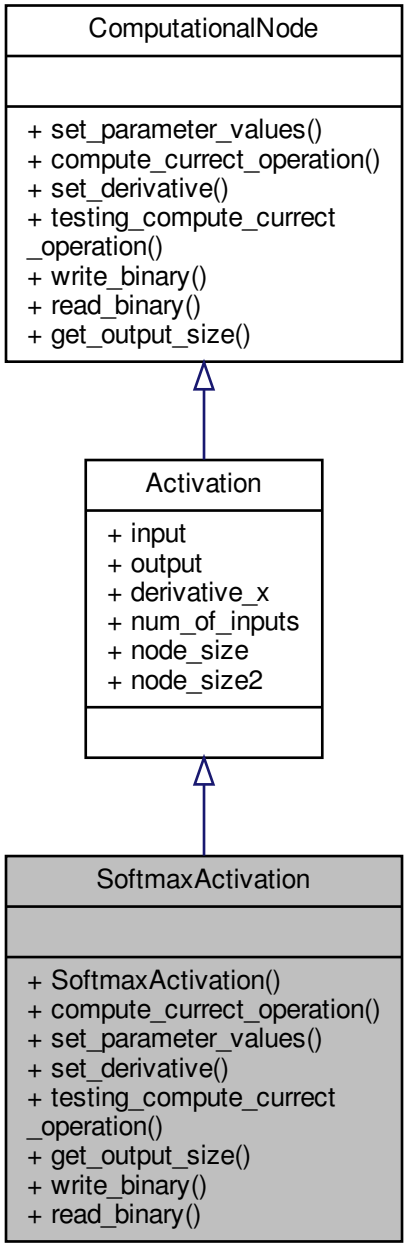
A class for the Softmax Activation function.

```
#include <Activation.h>
```

Inheritance diagram for SoftmaxActivation:



Collaboration diagram for SoftmaxActivation:



Public Member Functions

- SoftmaxActivation (int num_nodes)
- std::vector< Eigen::MatrixXd > compute_currect_operation (std::vector< Eigen::MatrixXd > tmp_input) override
- void set_parameter_values (double learning_rate, int batch_size) override
- std::vector< Eigen::MatrixXd > set_derivative (std::vector< Eigen::MatrixXd > prev_derivative) override

- std::vector< Eigen::MatrixXd > testing_compute_currect_operation (std::vector< Eigen::MatrixXd > tmp_input) override
- std::array< int, 3 > get_output_size () override
- void write_binary (std::ofstream &out) override
- void read_binary (std::ifstream &in) override

Additional Inherited Members

6.13.1 Detailed Description

A class for the Softmax Activation function.

Definition at line 231 of file Activation.h.

6.13.2 Constructor & Destructor Documentation

6.13.2.1 SoftmaxActivation()

```
SoftmaxActivation::SoftmaxActivation (
            int num_nodes )  [explicit]
```

A constructor for the Softmax activation operation. This class is designed only to be used in fully connected layers.

Parameters

num_nodes	an integer argument, which is for the number of input neurons of the activation function layer.

Definition at line 205 of file Activation.cpp.

```
205                                                 {
206
207     this->num_of_inputs = 1;
208     this->node_size = num_nodes;
209     this->node_size2 = 1;
210     this->input.emplace_back(Eigen::MatrixXd::Zero(num_nodes,1));
211     this->output.emplace_back(Eigen::MatrixXd::Zero(num_nodes,1));
212     this->derivative_x.emplace_back(Eigen::MatrixXd::Zero(num_nodes,1));
213 }
```

6.13.3 Member Function Documentation

6.13.3.1 compute_currect_operation()

```
std::vector< Eigen::MatrixXd > SoftmaxActivation::compute_currect_operation (
            std::vector< Eigen::MatrixXd > tmp_input )  [override], [virtual]
```

A function is to calculate the the current operation of the softmax node. The softmax activation function is applied to the given input. Softmax operation will give an output between 0 and 1. Softmax function is a good output activation function in the case of the existance of multiple classes.

Parameters

tmp_input	a vector of Eigen matrices, an input coming from the previous computational node.

Returns

a vector of Eigen matrices, returns the values of the input after applying sigmoid activation function.

Implements ComputationalNode.

Definition at line 217 of file Activation.cpp.

```
217
      {
218
219     this->input = tmp_input;
220     double max_val = tmp_input[0].maxCoeff(); // finds the maximum value in the vector
221     tmp_input[0] -= max_val * Eigen::MatrixXd::Ones(tmp_input[0].rows(), 1); // this line is required to
      prevent the numerical issues.
222     double sum = 0.0;
223     for(size_t i = 0; i < tmp_input[0].rows(); i++) {
224         tmp_input[0](i, 0) = exp(tmp_input[0](i, 0));
225         sum += tmp_input[0](i, 0);
226     }
227     this->output[0] = tmp_input[0] / sum;
228
229     return this->output;
230 }
```

6.13.3.2 get_output_size()

```
std::array<int, 3> SoftmaxActivation::get_output_size ( )  [inline], [override], [virtual]
```

A function to return the output size of this computational node.

Returns

respectively, number of outputs, height of the output, and the width of the output.

Implements ComputationalNode.

Definition at line 282 of file Activation.h.

```
282                                             { return std::array<int, 3> { this->num_of_inputs,
      this->node_size,
283                                                                           this->node_size2};}
```

6.13.3.3 read_binary()

```
void SoftmaxActivation::read_binary (
            std::ifstream & in )  [inline], [override], [virtual]
```

A virtual function which will be override by the child classes.

- This function is for loading the weights and the biases before the training of the network starts.

Parameters

in	an ifstream object

Implements ComputationalNode.

Definition at line 288 of file Activation.h.

```
288 {};
```

6.13.3.4 set_derivative()

```
std::vector< Eigen::MatrixXd > SoftmaxActivation::set_derivative (
            std::vector< Eigen::MatrixXd > prev_derivative )  [override], [virtual]
```

A function to calculate the derivative of the Softmax operation. Derivative of the previous operations in the network are received as the input paramater. The result is the multiplication of the current derivative of the operation and the input down flow derivative.

Parameters

prev_derivative	a vector of Eigen matrices, down flow derivative through this Softmax activation function.

Returns

a vector of Eigen matrices, the multiplication of the current derivative of the operation and the input down flow derivative.

Implements ComputationalNode.

Definition at line 253 of file Activation.cpp.

```
253                                                                                                                {
254
255     derivative_x[0].setZero();
256
257     for(size_t output_idx = 0; output_idx < this->output[0].rows(); output_idx++) {
258         for(size_t input_idx = 0; input_idx < this->input[0].rows(); input_idx++) {
259             if(output_idx == input_idx) {
260                 this->derivative_x[0](input_idx, 0) += prev_derivative[0](output_idx, 0) *
      this->output[0](output_idx, 0) * (1 - this->output[0](output_idx, 0));
261             }
262             else {
263                 this->derivative_x[0](input_idx, 0) -= prev_derivative[0](output_idx, 0) *
      this->output[0](output_idx, 0) * this->output[0](input_idx, 0);
264             }
265         }
266     }
267
268     return this->derivative_x;
269 }
```

6.13.3.5 set_parameter_values()

```
void SoftmaxActivation::set_parameter_values (
            double learning_rate,
            int batch_size )  [override], [virtual]
```

A function to set weights and biases by using the derivatives, which are calculated during the backward pass, and to set input, output and derivative values to zero. Since there is no weight or bias in the activation function layers, it is used only for setting all the input, output and, derivatives to zero.

Parameters

learning_rate	a double argument, learning rate.
batch_size	an integer argument, the number of elements in one batch.

Implements ComputationalNode.

Definition at line 247 of file Activation.cpp.

```
247                                                                                {
248     this->derivative_x[0].setZero();
249     this->input[0].setZero();
250     this->output[0].setZero();
251 }
```

6.13.3.6 testing_compute_currect_operation()

```
std::vector< Eigen::MatrixXd > SoftmaxActivation::testing_compute_currect_operation (
            std::vector< Eigen::MatrixXd > tmp_input )  [override], [virtual]
```

A function to run the forward pass of the softmax activation operation during the testing phase.

Parameters

tmp_input	a vector of Eigen matrices, an input coming from the previous computational node.

Returns

a vector of Eigen matrices, returns the values of the input after the softmax activation operation.

< finds the maximum value in the vector

< this line is required to prevent the numerical issues.

Implements ComputationalNode.

Definition at line 232 of file Activation.cpp.

```
233                                                 {
234     this->input = tmp_input;
235     double max_val = tmp_input[0].maxCoeff();
236     tmp_input[0] -= max_val * Eigen::MatrixXd::Ones(tmp_input[0].rows(), 1);
237     double sum = 0.0;
238     for(size_t i = 0; i < tmp_input[0].rows(); i++) {
239         tmp_input[0](i, 0) = exp(tmp_input[0](i, 0));
240         sum += tmp_input[0](i, 0);
241     }
242     this->output[0] = tmp_input[0] / sum;
243 
244     return this->output;
245 }
```

6.13.3.7 write_binary()

```
void SoftmaxActivation::write_binary (
            std::ofstream & out )  [inline], [override], [virtual]
```

A virtual function which will be override by the child classes. This function is for saving the weights and the biases after the training of the network.

Parameters

out	an ofstream object

Implements ComputationalNode.

Definition at line 285 of file Activation.h.

```
285 {};
```

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

- /home/nehil/dlfsC++/libdl/headers/Activation.h
- /home/nehil/dlfsC++/libdl/src/Activation.cpp

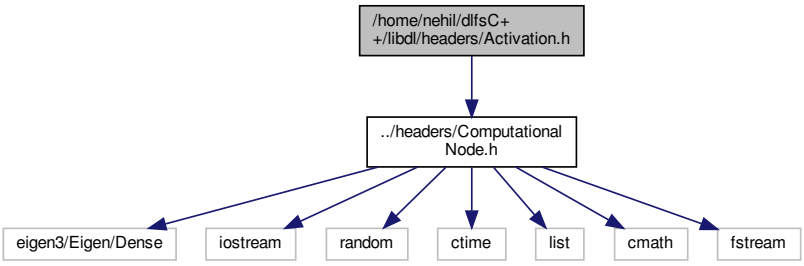
Chapter 7

File Documentation

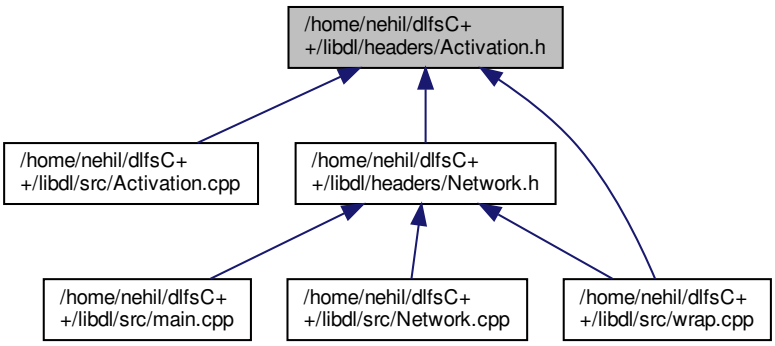
7.1 /home/nehil/dlfsC++/libdl/headers/Activation.h File Reference

```
#include "../headers/ComputationalNode.h"
```

Include dependency graph for Activation.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Activation

  An abstract class, which is the parent of all the activation functions.
- class SigmoidActivation

  A class for the Sigmoid Activation function.
- class ReLUActivation

  A class for the Rectified Linear Units (RELU) Activation function.
- class LeakyReLUActivation

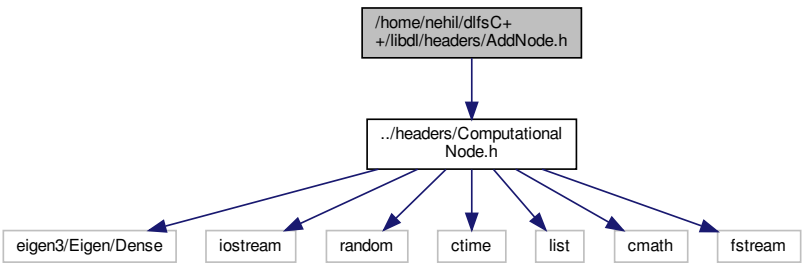
  A class for the Leaky Rectified Linear Units (RELU) Activation function.
- class SoftmaxActivation

  A class for the Softmax Activation function.

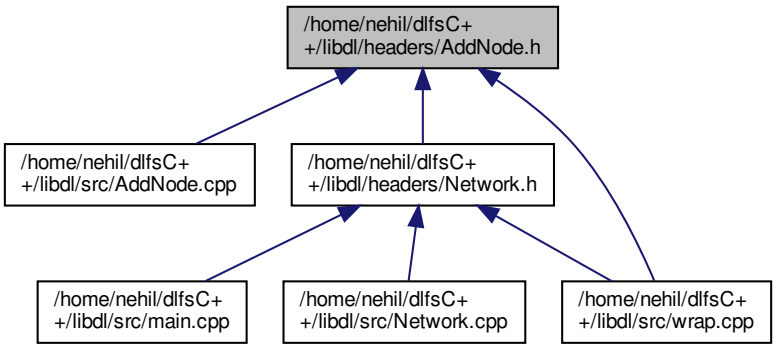
7.2 /home/nehil/dlfsC++/libdl/headers/AddNode.h File Reference

```
#include "../headers/ComputationalNode.h"
```

Include dependency graph for AddNode.h:



This graph shows which files directly or indirectly include this file:



Classes

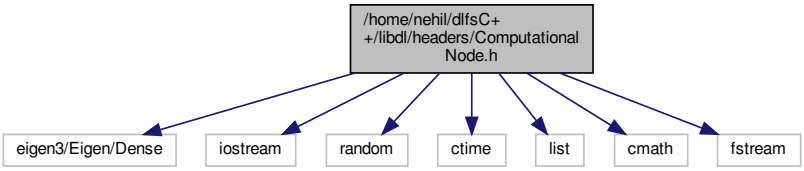
- class AddNode

  A class which implements the add bias operation in the network. This only called in the fully connected layers.

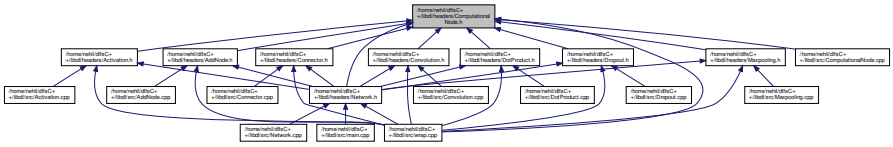
7.3 /home/nehil/dlfsC++/libdl/headers/ComputationalNode.h File Reference

```
#include <eigen3/Eigen/Dense>
#include <iostream>
#include <random>
#include <ctime>
#include <list>
#include <cmath>
#include <fstream>
```

Include dependency graph for ComputationalNode.h:



This graph shows which files directly or indirectly include this file:



Classes

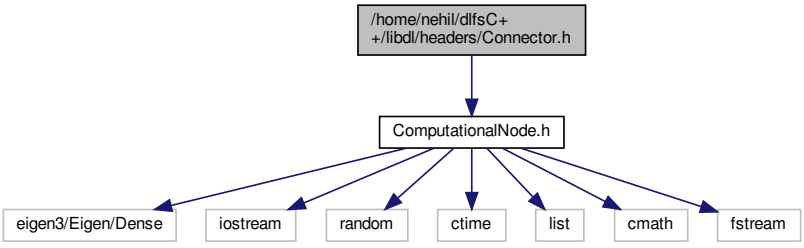
- class ComputationalNode

  An abstract class, which is the parent of all the computation operations in the network.

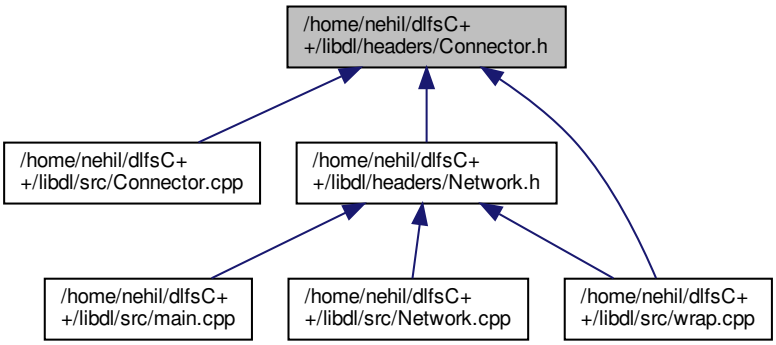
7.4 /home/nehil/dlfsC++/libdl/headers/Connector.h File Reference

```
#include "ComputationalNode.h"
```

Include dependency graph for Connector.h:



This graph shows which files directly or indirectly include this file:



Classes

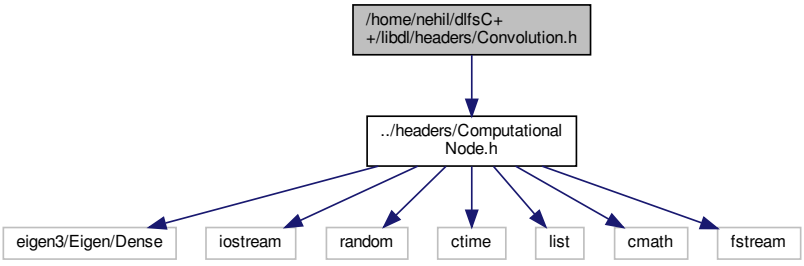
- class Connector

  A class which refers to the operation for flattening the output out convolution or max pooling layers into a vector. In this way, the input will not be matrix anymore, and it will be suitable for the fully connected layers.

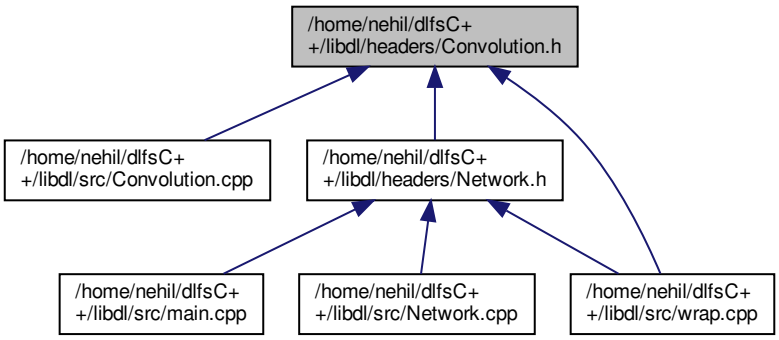
7.5 /home/nehil/dlfsC++/libdl/headers/Convolution.h File Reference

```
#include "../headers/ComputationalNode.h"
```

Include dependency graph for Convolution.h:



This graph shows which files directly or indirectly include this file:



Classes

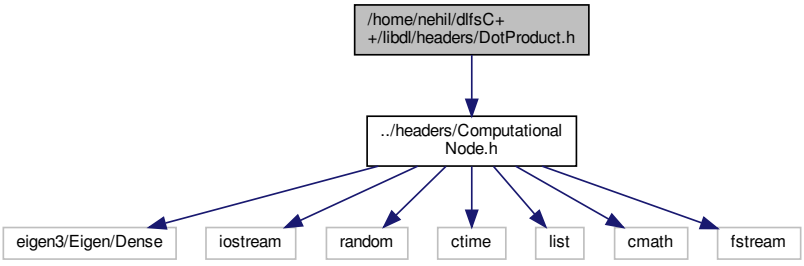
- class Convolution

  A class which implements the convolution operation in the network.

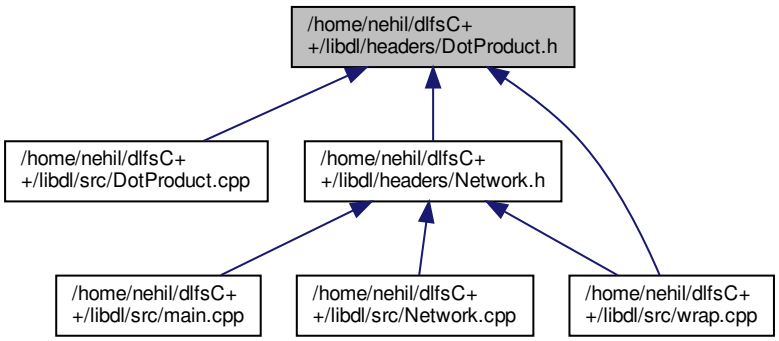
7.6 /home/nehil/dlfsC++/libdl/headers/DotProduct.h File Reference

```
#include "../headers/ComputationalNode.h"
```

Include dependency graph for DotProduct.h:



This graph shows which files directly or indirectly include this file:



Classes

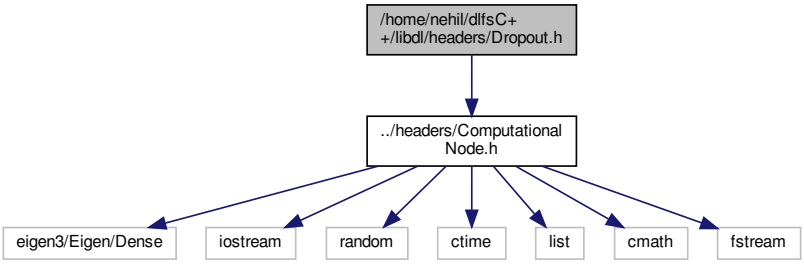
- class DotProduct

  A class which implements the dot product operation in the network.

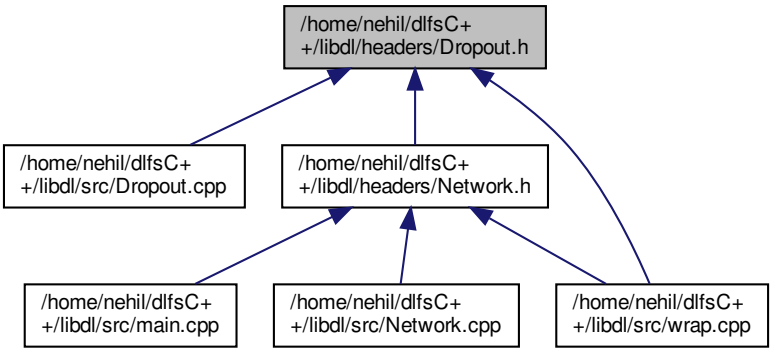
7.7 /home/nehil/dlfsC++/libdl/headers/Dropout.h File Reference

```
#include "../headers/ComputationalNode.h"
```

Include dependency graph for Dropout.h:



This graph shows which files directly or indirectly include this file:



Classes

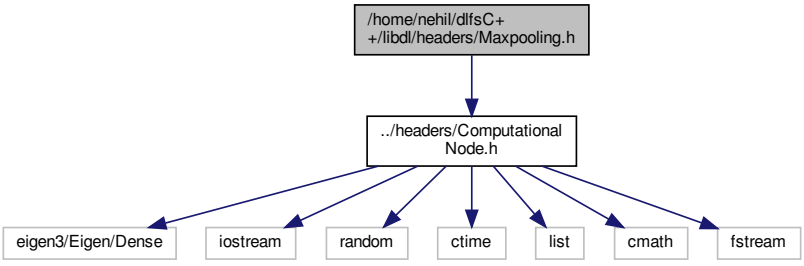
- class Dropout

  A class which implements the dropout computation in the network.

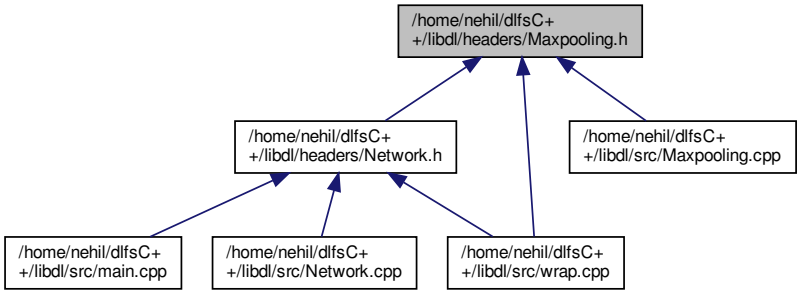
7.8 /home/nehil/dlfsC++/libdl/headers/Maxpooling.h File Reference

```
#include "../headers/ComputationalNode.h"
```

Include dependency graph for Maxpooling.h:



This graph shows which files directly or indirectly include this file:



Classes

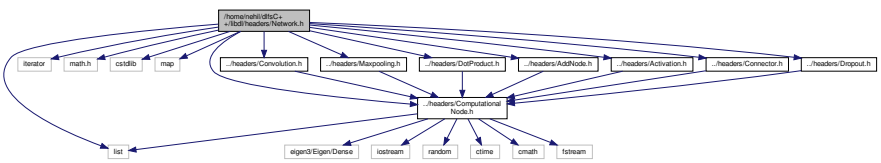
- class Maxpooling

  A class which implements the maxpooling operation in the network.

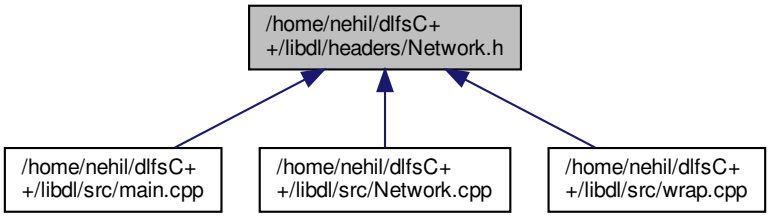
7.9 /home/nehil/dlfsC++/libdl/headers/Network.h File Reference

```
#include <list>
#include <iterator>
#include <math.h>
#include <cstdlib>
#include <map>
#include "../headers/ComputationalNode.h"
#include "../headers/Convolution.h"
#include "../headers/Maxpooling.h"
#include "../headers/DotProduct.h"
#include "../headers/AddNode.h"
#include "../headers/Activation.h"
#include "../headers/Connector.h"
#include "../headers/Dropout.h"
```

Include dependency graph for Network.h:



This graph shows which files directly or indirectly include this file:



Classes

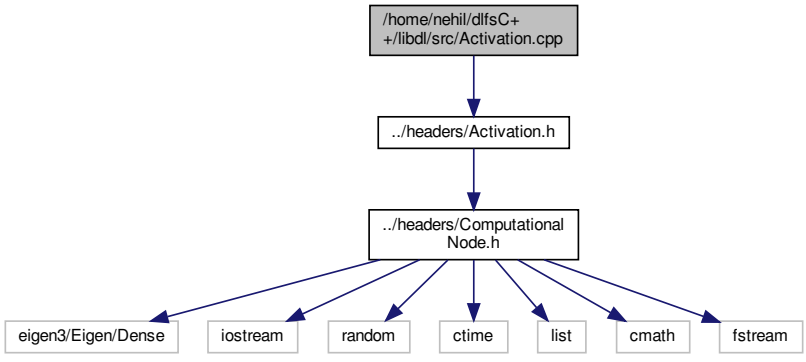
- class Network

  A class which contains all functions to build the network, do training, validation and testing.

7.10 /home/nehil/dlfsC++/libdl/src/Activation.cpp File Reference

```
#include "../headers/Activation.h"
```

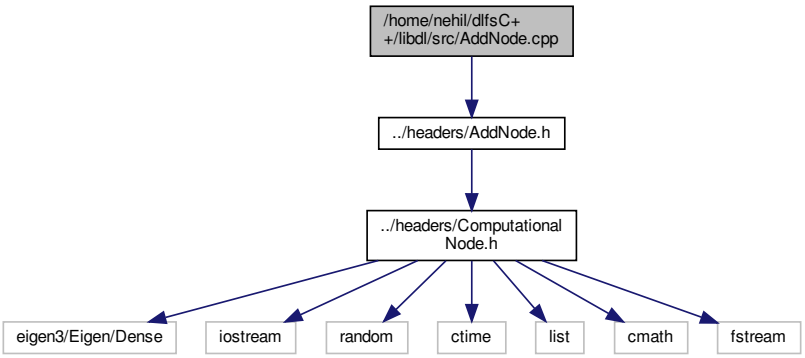
Include dependency graph for Activation.cpp:



7.11 /home/nehil/dlfsC++/libdl/src/AddNode.cpp File Reference

```
#include "../headers/AddNode.h"
```

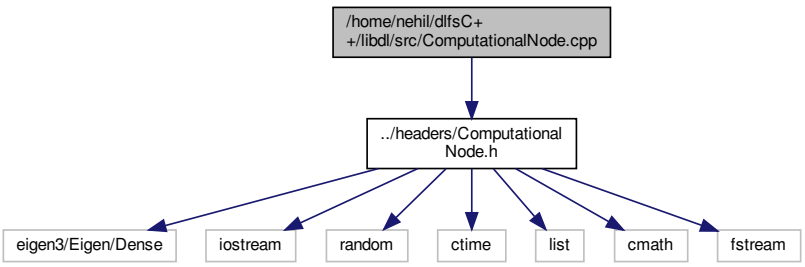
Include dependency graph for AddNode.cpp:



7.12 /home/nehil/dlfsC++/libdl/src/ComputationalNode.cpp File Reference

```
#include "../headers/ComputationalNode.h"
```

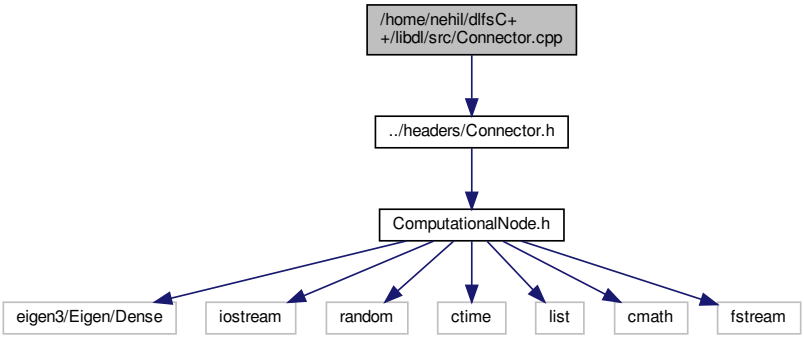
Include dependency graph for ComputationalNode.cpp:



7.13 /home/nehil/dlfsC++/libdl/src/Connector.cpp File Reference

```
#include "../headers/Connector.h"
```

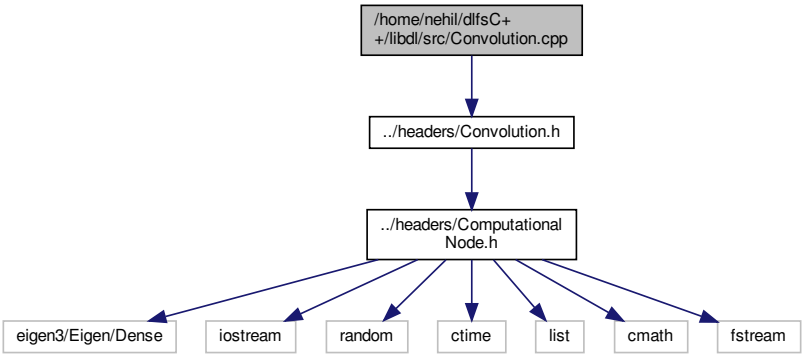
Include dependency graph for Connector.cpp:



7.14 /home/nehil/dlfsC++/libdl/src/Convolution.cpp File Reference

```
#include "../headers/Convolution.h"
```

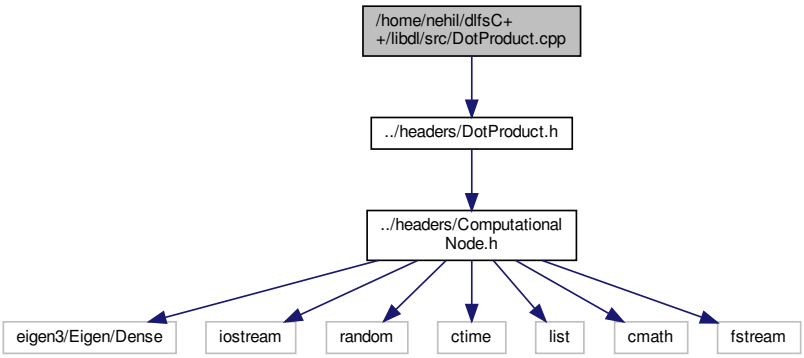
Include dependency graph for Convolution.cpp:



7.15 /home/nehil/dlfsC++/libdl/src/DotProduct.cpp File Reference

```
#include "../headers/DotProduct.h"
```

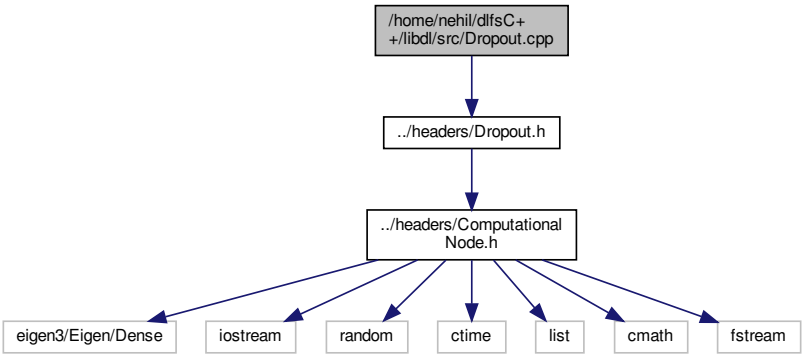
Include dependency graph for DotProduct.cpp:



7.16 /home/nehil/dlfsC++/libdl/src/Dropout.cpp File Reference

```
#include "../headers/Dropout.h"
```

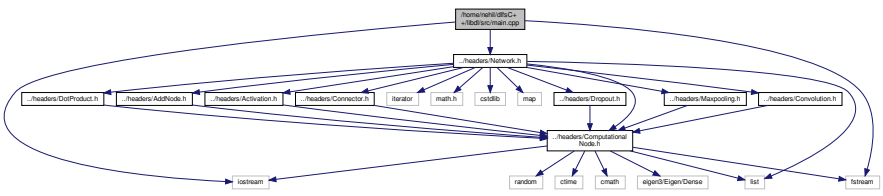
Include dependency graph for Dropout.cpp:



7.17 /home/nehil/dlfsC++/libdl/src/main.cpp File Reference

```
#include <iostream>
#include <fstream>
#include "../headers/Network.h"
```

Include dependency graph for main.cpp:



Functions

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

7.17.1 Function Documentation

7.17.1.1 main()

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

The main entry point of the program

Definition at line 8 of file main.cpp.

```
9 {
10
11     Eigen::MatrixXd inputs(4,2);
12     Eigen::VectorXd targets(4);
13     inputs(0,0) = 0; inputs(0,1) = 1; targets(0) = 1;
14     inputs(1,0) = 1; inputs(1,1) = 0; targets(1) = 1;
15     inputs(2,0) = 0; inputs(2,1) = 0; targets(2) = 0;
16     inputs(3,0) = 1; inputs(3,1) = 1; targets(3) = 0;
17
18
19     Eigen::MatrixXd test_inputs(4,2);
20     test_inputs(0,0) = 0; test_inputs(0,1) = 0;
21     test_inputs(3,0) = 0; test_inputs(3,1) = 1;
22     test_inputs(2,0) = 1; test_inputs(2,1) = 0;
23     test_inputs(1,0) = 1; test_inputs(1,1) = 1;
24
25     std::vector<int> architecture = {2, 4, 1};
26
27     /*auto *xor_net = new Network(architecture); // All the layers are being initialized.
28     xor_net->train(inputs, targets);
29     xor_net->test(test_inputs);*/
30
31     Eigen::MatrixXd input;
32     input = Eigen::MatrixXd::Ones(28, 28);
33
34
35     std::vector<std::string> activation_functions = {"Sigmoid", "Sigmoid", "None"};
36
37     auto *net = new Network(); // All the layers are being initialized.
38
39
40
41
42
43
44     return 0;
45 }
```

7.18 /home/nehil/dlfsC++/libdl/src/malaria_detection.py File Reference

Namespaces

- malaria_detection

Functions

- def malaria_detection.read_images ()
- def malaria_detection.train_test_set (files_df)
- def malaria_detection.load_images (train_files, train_labels, val_files, test_files, mean, hdf5_file, IMG_DIMS)
- def malaria_detection.discover_dataset (train_files)

Variables

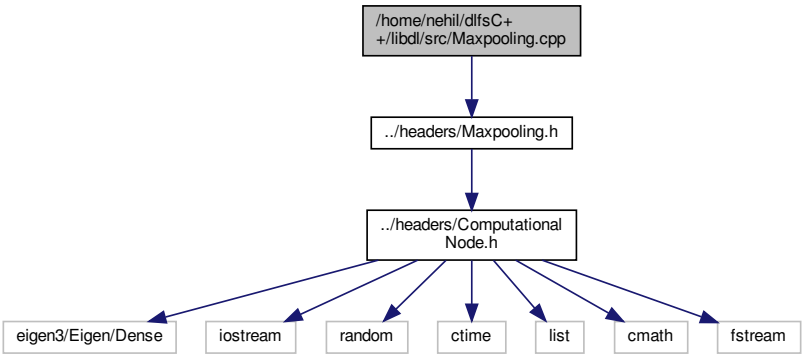
- string malaria_detection.datapath = '../Malaria_Dataset/'
- bool malaria_detection.SUBTRACT_MEAN = False
- string malaria_detection.path_train = "../Malaria_Dataset/cell_images_28/train/"
- string malaria_detection.path_val = "../Malaria_Dataset/cell_images_28/validation/"
- string malaria_detection.path_test = "../Malaria_Dataset/cell_images_28/test/"
- tuple malaria_detection.IMG_DIM = (28, 28)
- malaria_detection.hdf5_file = None
- string malaria_detection.hdf5_datapath = '../Malaria_Dataset/data_64.hdf5'
- tuple malaria_detection.train_shape = (len(train_files), 28, 28 , 3)
- tuple malaria_detection.val_shape = (len(val_files), 28, 28, 3)
- tuple malaria_detection.test_shape = (len(test_files), 28, 28, 3)
- malaria_detection.le = LabelEncoder()
- malaria_detection.train_labels_enc = le.transform(train_labels)
- malaria_detection.val_labels_enc = le.transform(val_labels)
- malaria_detection.test_labels_enc = le.transform(test_labels)
- malaria_detection.mean = np.zeros(train_shape[1:], np.float32)
- malaria_detection.figsize
- int malaria_detection.n = 0
- malaria_detection.r = np.random.randint(0 , hdf5_file["train_img"].shape[0] , 1)
- malaria_detection.hspace
- malaria_detection.wspace
- int malaria_detection.BATCH_SIZE = 10
- int malaria_detection.EPOCH_SIZE = 5
- malaria_detection.data_num = hdf5_file["train_img"].shape[0]
- malaria_detection.validation_set_num = hdf5_file["val_img"].shape[0]
- malaria_detection.batches_list = list(range(int(ceil(float(data_num) / BATCH_SIZE))))
- malaria_detection.val_batches_list = list(range(int(ceil(float(validation_set_num) / BATCH_SIZE))))
- malaria_detection.net = dl.Network()
- malaria_detection.output_size_conv1
- malaria_detection.num_of_outputs_conv1
- malaria_detection.output_size_relu1_h
- malaria_detection.output_size_relu1_w
- malaria_detection.num_of_outputs_relu1
- malaria_detection.output_size_conv2
- malaria_detection.num_of_outputs_conv2
- malaria_detection.output_size_relu2_h
- malaria_detection.output_size_relu2_w
- malaria_detection.num_of_outputs_relu2
- malaria_detection.output_size_pooling2
- malaria_detection.num_of_outputs_pooling_2
- malaria_detection.output_size_conv3
- malaria_detection.num_of_outputs_conv3
- malaria_detection.output_size_relu3_h
- malaria_detection.output_size_relu3_w
- malaria_detection.num_of_outputs_relu3
- malaria_detection.output_size_conv4
- malaria_detection.num_of_outputs_conv4
- malaria_detection.output_size_relu4_h
- malaria_detection.output_size_relu4_w
- malaria_detection.num_of_outputs_relu4
- malaria_detection.output_size_pooling4
- malaria_detection.num_of_outputs_pooling_4
- malaria_detection.output_size_fully1_h

- malaria_detection.output_size_fully1_w
- malaria_detection.num_of_outputs_fully1
- malaria_detection.output_size_relu5_h
- malaria_detection.output_size_relu5_w
- malaria_detection.num_of_outputs_relu5
- malaria_detection.output_size_fully2_h
- malaria_detection.output_size_fully2_w
- malaria_detection.num_of_outputs_fully2
- malaria_detection.output_size_sigmoid5_h
- malaria_detection.output_size_sigmoid5_w
- malaria_detection.num_of_outputs_sigmoid5
- int malaria_detection.n_values = 2
- list malaria_detection.train_cost = [ ]
- list malaria_detection.train_cost10 = [ ]
- list malaria_detection.iterations = [ ]
- list malaria_detection.iterations10 = [ ]
- float malaria_detection.train_acc = 0.0
- float malaria_detection.val_acc = 0.0
- list malaria_detection.epoch_axis = [ ]
- list malaria_detection.train_acc_epoch = [ ]
- list malaria_detection.val_acc_epoch = [ ]
- malaria_detection.images = None
- int malaria_detection.iter = 0
- int malaria_detection.iter10 = 0
- float malaria_detection.batch_10_cost = 0.0
- int malaria_detection.i_s = i ∗ BATCH_SIZE
- malaria_detection.i_e = min([(i + 1) ∗ BATCH_SIZE, data_num])
- malaria_detection.labels = hdf5_file["train_labels"][i_s:i_e]
- malaria_detection.labels_one_hot = np.eye(n_values)[labels]
- malaria_detection.batch_cost
- malaria_detection.batch_accuracy = net.validation(images / 255., labels_one_hot, BATCH_SIZE)
- malaria_detection.label
- malaria_detection.loc

7.19 /home/nehil/dlfsC++/libdl/src/Maxpooling.cpp File Reference

```
#include "../headers/Maxpooling.h"
```

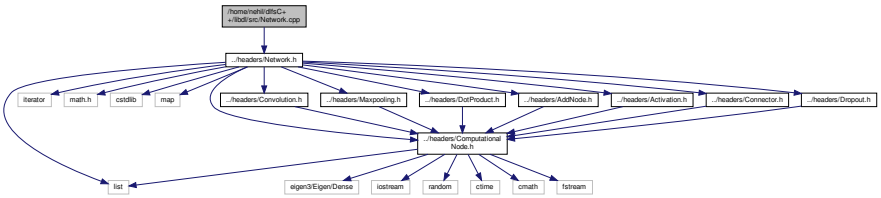
Include dependency graph for Maxpooling.cpp:



7.20 /home/nehil/dlfsC++/libdl/src/Network.cpp File Reference

```
#include "../headers/Network.h"
```

Include dependency graph for Network.cpp:



7.21 /home/nehil/dlfsC++/libdl/src/setup.py File Reference

Namespaces

- setup

Variables

- list setup.cpp_args = ['-std=c++11']
- list setup.ext_modules
- setup.name
- setup.version
- setup.author
- setup.author_email
- setup.description

7.22 /home/nehil/dlfsC++/libdl/src/wrap.cpp File Reference

```
#include <pybind11/pybind11.h>
#include <pybind11/stl.h>
#include <pybind11/complex.h>
#include <pybind11/functional.h>
#include <pybind11/chrono.h>
#include <eigen3/Eigen/Core>
#include <eigen3/Eigen/SparseCore>
#include <pybind11/eigen.h>
#include <eigen3/Eigen/Dense>
#include "../headers/Network.h"
#include "../headers/ComputationalNode.h"
#include "../headers/Convolution.h"
#include "../headers/Maxpooling.h"
#include "../headers/DotProduct.h"
#include "../headers/AddNode.h"
#include "../headers/Activation.h"
#include "../headers/Connector.h"
```

```
#include "../headers/Dropout.h"
```

Include dependency graph for wrap.cpp:

Functions

- PYBIND11_MODULE (dl, m)

7.22.1 Function Documentation

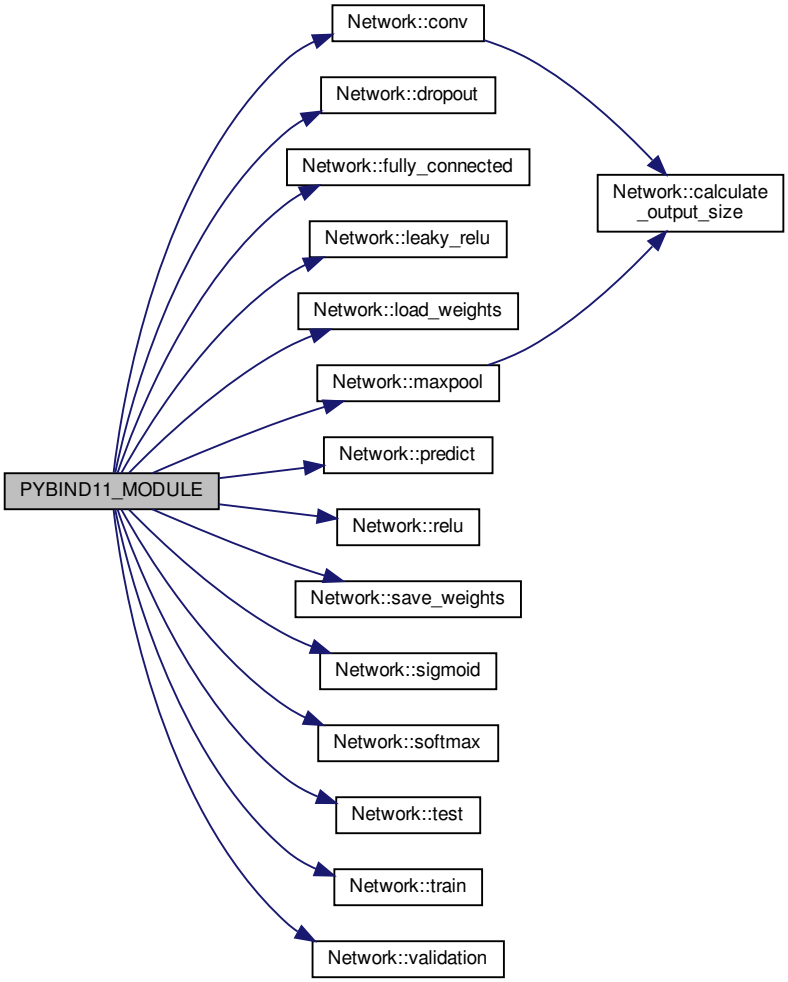
7.22.1.1 PYBIND11_MODULE()

```
PYBIND11_MODULE (
              dl ,
              m  )
```

Definition at line 28 of file wrap.cpp.

```
28                          {
29     m.doc() = "awesome dl library";
30     py::class_<Network>(m, "Network")
31     .def(py::init<>())
32     .def("train", &Network::train)
33     .def("test", &Network::test)
34     .def("validation", &Network::validation)
35     .def("loadWeights", &Network::load_weights)
36     .def("saveWeights", &Network::save_weights)
37     .def("conv", &Network::conv)
38     .def("maxpool", &Network::maxpool)
39     .def("fullyConnected", &Network::fully_connected)
40     .def("relu", &Network::relu)
41     .def("leakyRelu", &Network::leaky_relu)
42     .def("softmax", &Network::softmax)
43     .def("sigmoid", &Network::sigmoid)
44     .def("dropout", &Network::dropout)
45     .def("predict", &Network::predict);
46 }
```

Here is the call graph for this function:



Index