Main program

Process the different types of defect images. Generates and trains convolutional networks for defect detection from the processed defect images

Authors: Antonis Kantounias - Eleutherios Kantounias, Email: antonis.kantounias@gmail.com, Date: 2022.12.29

```
unprocessedImageDirectory = "C:\Users\Antonis Kantounias\Documents\ergasies\inteligentMachiningSystems\excersise3\Codes\Data";
```

Create all possible data sets

```
% Possible processes
optionNames                 =   {
                                'process_imadjust', 'process_average',  'process_imbinarize',   'process_filter2laplacian', 'process_filter2prewitt',   'process_bwareopen',    'process_imfill'
                                };

% Combinations of possible processes
optionValuesCombinations     =      [
                                    false,              false,             false,                false,                              false,                         false,                    false   % Filter1
                                    true,               false,             false,                false,                              false,                         false,                    false   % Filter2
                                    false,              false,             true,                 false,                              false,                         false,                    false   % Filter3
                                    false,              false,             false,                true,                               false,                         false,                    false   % Filter4
                                    false,              false,             true,                 true,                               false,                         false,                    false   % Filter5
                                    false,              false,             false,                false,                              true,                          false,                    false   % Filter6
                                    false,              false,             false,                false,                              true,                          false,                    false   % Filter7
                                    false,              true,              true,                 false,                              false,                         false,                    false   % Filter8
                                    false,              true,              true,                 true,                               false,                         false,                    false   % Filter9
                                    false,              true,              true,                 false,                              true,                          false,                    false   % Filter10
                                    false,              false              true,                 false,                              false,                         false,                    false   % Filter11
                                    false,              false,             false,                true,                               false,                         false,                    false   % Filter12
                                    false,              false,             false,                false,                              true,                          false,                    false   % Filter13
                                    false,              true,              false,                true,                               false,                         false,                    false   % Filter14
                                    false,              true,              false,                false,                              true,                          false,                    false   % Filter15
                                ];

% Equivalent names for each process combination
aliasses        =   {
                    'Filter1'
                    'Filter2'
                    'Filter3'
                    'Filter4'
                    'Filter5'
                    'Filter6'
                    'Filter7'
                    'Filter8'
                    'Filter9'
                    'Filter10'
                    'Filter11'
                    'Filter12'
                    'Filter13'
                    'Filter14'
                    'Filter15'
                    };
```

```
% Generate dataset
for iDataCombination = 12:15%1:length(aliasses)
    % Generate the option values for current combination
    optionValues    = optionValuesCombinations(iDataCombination,:);
    alias           = aliasses{iDataCombination};
    % Generate varagin file
    varargin = cell(1,2*length(optionNames));
    for iOption  = 1:length(optionNames)
        varargin{2*iOption-1}   = optionNames{iOption};
        varargin{2*iOption}     = optionValues(iOption);
    end
    % Generate processed image data base
    processDBImages(unprocessedImageDirectory,alias,varargin{:})
end
```

Generate all possible neural networks

```
% Network possible layers
networkLayerCombinations    =   {
                                [16,32,64,16]
                                [16,64,128,32]
                                [16,32,64]
                                [16,64,32]
                                };
% Network equivalent names
networkAliasses             =   {
                                'Network1'
                                'Network2'
                                'Network3'
                                'Network4'
                                };

% Generate and train the networks
for iNetworkCombination = 1:length(networkAliasses)
    for iDataCombination = 1:length(aliasses)
        networkAlias                = networkAliasses{iNetworkCombination};
        networkLayers               = networkLayerCombinations{iNetworkCombination};
        processedImageDirectory     = join([unprocessedImageDirectory,"Processed_",string(aliasses{iDataCombination})],"");
        createDeepLearningNetwork(processedImageDirectory,networkLayers,networkAlias)
    end
end
```

Generate all possible neural networks

```matlab
% Network possible layers
networkLayerCombinations    =   {
                                [16,32,64,16]
                                [16,64,128,32]
                                [16,32,64]
                                [16,64,32]
                                };
% Network equivalent names
networkAliasses             =   {
                                'Network1'
                                'Network2'
                                'Network3'
                                'Network4'
                                };

% Generate and train the networks
for iNetworkCombination = 1:length(networkAliasses)
    for iDataCombination = 1:length(aliasses)
        networkAlias                = networkAliasses{iNetworkCombination};
        networkLayers               = networkLayerCombinations{iNetworkCombination};
        processedImageDirectory     = join([unprocessedImageDirectory,"Processed_",string(aliasses{iDataCombination})],"");
        createDeepLearningNetwork(processedImageDirectory,networkLayers,networkAlias)
    end
end
```

createDeepLearningNetwork

```
function [networkResult,dirNameResult] = createDeepLearningNetwork(datasetPath,networkLayers,networkAlias)
```

createDeepLearningNetwork

Creates and trains a convolutional neural network for image recognition. The network is saved at the datasetPath folder.

Inputs: datasetPath Directory where the processed image data are located [string] networkAlias Network archietecture related naming [string]

Output: network Contains the trained network file and the accuracy result [structure] datasetPath Trained network file location [string]

Authors: Antonis Kantounias - Eleutherios Kantounias, Email: antonis.kantounias@gmail.com, Date: 2022.12.29

Constant data

```
PERCENTAGEOFTRAINFILES = 0.80;
```

Load image data

```
% Load sample data as an image datastore
imageData = imageDatastore( datasetPath,...
                            'IncludeSubFolders',true,...
                            'LabelSource','foldernames');

% Specify the size of the images in the input layer
imageExample                = readimage(imageData,1);
[resolutionX, resolutionY]  = size(imageExample);
resolutionZ                 = 1; % 2d image

% Specify the categorical label number
labelCount          = countEachLabel(imageData);
[numOfLabels,~]     = size(labelCount);
```

Specify training and validation sets

```
% Split the homogenous datastore into the train data store and the validation datastore randomly
[imageDataTrain,imageDataValidation] = splitEachLabel(imageData,PERCENTAGEOFTRAINFILES,'randomize');
```

Define network architecture (generate network's layers)

```
layers =    imageInputLayer([resolutionX resolutionY resolutionZ]);

for iLayer = 1:length(networkLayers)
    layers(end+1:end+4,1) =      [
                                convolution2dLayer(3,networkLayers(iLayer),'Padding','same')
                                batchNormalizationLayer
                                reluLayer
                                maxPooling2dLayer(2,'Stride',2)
                                ];
end

layers(end:end+2,1) =      [
                           fullyConnectedLayer(numOfLabels)
                        softmaxLayer
                        classificationLayer
                        ];
```

Specify training options

```
options =   trainingOptions(...
                            'sgdm', ...
                            'InitialLearnRate',0.01, ...
                            'MaxEpochs',30, ...
                            'Shuffle','every-epoch', ...
                            'ValidationData',imageDataTrain, ...
                            'ValidationFrequency',15, ...
                            'Verbose',false, ...
                            'Plots','training-progress'...
                            );
```

Train the network

```
networkTrained =       trainNetwork(imageDataTrain,layers,options);
```

Compute the accuracy of the network

```
classificationPredicted =        classify(networkTrained,imageDataValidation);
classificationReal      =        imageDataValidation.Labels;
networkAccuracy         =   sum(classificationPredicted == classificationReal)/numel(classificationReal);
```

Save network training results

```matlab
networkResult.networkTrained     =   networkTrained;
networkResult.networkAccuracy   =   networkAccuracy;

% Create result directory
dirNameSplit                    =   split(datasetPath,string(filesep));
dirNameResult                   =   join([dirNameSplit(1:end-1)',"Results"],string(filesep));
resultName                      =   dirNameSplit(end);

if ~exist(dirNameResult, 'dir')
    mkdir(dirNameResult)
end

% Create result figure directory
dirNameResultsFigures            =   join([dirNameResult,"Figures"],string(filesep));

if ~exist(dirNameResultsFigures, 'dir')
    mkdir(dirNameResultsFigures)
end

% Create result network directory
dirNameResultsNetworks           =   join([dirNameResult,"Networks"],string(filesep));

if ~exist(dirNameResultsNetworks, 'dir')
    mkdir(dirNameResultsNetworks)
end

% Save generated figure
FigList                         =       findobj(allchild(0), 'flat', 'Type', 'figure');
FigHandle                       = FigList(1);
FigHandle.Name                  = resultName;
savefig(FigHandle, join([dirNameResultsFigures,"\", resultName, string(networkAlias), ".fig"],""));

% Save generated network structure
save(join([dirNameResultsNetworks,"\",resultName, string(networkAlias),".mat"],""),'networkResult')

end
```

processDBImages

```
function processDBImages(unprocessedImageDirectory,alias,varargin)
```

processDBImages

Loads all image files from a specific directory, process the images and save them to an equivalent, processed directory

Inputs:	unprocessedImagesDirectory	Directory where the unprocessed data are stored, full name	[string]
	alias	Name extension for the processed data	[string]

Authors: Antonis Kantounias - Eleutherios Kantounias, Email: antonis.kantounias@gmail.com, Date: 2022.12.29

Get a list of all files and folders in this folder.

```
unprocessedImageDir     = dir(unprocessedImageDirectory);
unprocessedImageFiles   =  unprocessedImageDir(~[unprocessedImageDir.isdir]);
```

Process and save the image files

```
for iFile = 1:length(unprocessedImageFiles)

    fileName             = unprocessedImageFiles(iFile).name;
    [~,~,fileExtension] = fileparts(fileName);

    if strcmp(fileExtension,'.bmp')
        imageName = fileName;
        loadProcessSaveImage(unprocessedImageDirectory,alias,imageName,varargin{:});
    end

end
```

loadProcessSaveImage

```
function [processedImageDirectory] = loadProcessSaveImage(unprocessedImageDirectory,alias,imageName,varargin)
```

loadProcessSaveImage

loadProcessSaveImage loads the images of the database folder. Process the images and saves them to a new database folder that will be used for network training and validation

Inputs: unprocessedImagesDirectory Directory where the unprocessed data are stored, full name [string] alias Name extension for the processed data [string]

Outpus: processedImagesDirectory Directory where the unprocessed data are stored, full name [string]

Authors: Antonis Kantounias - Eleutherios Kantounias, Email: antonis.kantounias@gmail.com, Date: 2022.12.29

Add parameters

```
p = inputParser;
p.addParameter('process_imadjust',          true);
p.addParameter('process_imbinarize',        true);
p.addParameter('process_filter2laplacian',     true);
p.addParameter('process_filter2prewitt',       false);
p.addParameter('process_bwareopen',         true);
p.addParameter('process_imfill',            true);

p.parse(varargin{:})
process_imadjust            = p.Results.process_imadjust;
process_imbinarize          = p.Results.process_imbinarize;
process_filter2prewitt      = p.Results.process_filter2prewitt;
process_filter2laplacian   = p.Results.process_filter2laplacian;
process_bwareopen           = p.Results.process_bwareopen;
process_imfill              = p.Results.process_imfill;
```

Load image

```
% Read image file
imageInitialName    = join([unprocessedImageDirectory,string(filesep),imageName],"");
imageFinal          = imread(imageInitialName);
```

Process image

```matlab
% Scale correction
imageFinal          = mat2gray(imageFinal);

% Adjust image intensity
if process_imadjust
    imageFinal = imadjust(imageFinal,[],[0.8,1]);
end

% Convert the image into binary using adaptive thresholding
if process_imbinarize
    imageFinal  = imbinarize(imageFinal,'adaptive','ForegroundPolarity','dark','Sensitivity',0.5);
end

% Perform filter operation to look for edges (2nd degree derivative detection)
if process_filter2laplacian
    imageFinal = filter2(fspecial('laplacian'),imageFinal);
end

% Perform filter operation to look for edges (1nd degree derivative detection)
if process_filter2prewitt
    imageFinal = filter2(fspecial('prewitt'),imageFinal);
    imageFinal = imadjust(imageFinal);
end

% Scale correction
imageFinal          = mat2gray(imageFinal);

% Convert the image into binary using adaptive thresholding
if process_imbinarize
    imageFinal  = imbinarize(imageFinal);
end

% Remove small objects from binary image
if process_bwareopen
    pixelSize  = 2;
    imageFinal = bwareaopen(imageFinal, pixelSize);
end

% Fill the holes
if process_imfill
    imageFinal(1,:)     = 1-imageFinal(1,:);
    imageFinal(end,:)   = 1-imageFinal(end,:);
    imageFinal(:,1)     = 1-imageFinal(:,1);
    imageFinal(:,end)   = 1-imageFinal(:,end);
    imageFinal = imfill(imageFinal, 'holes');
end
```

Save image

```
% Create the processed data base folder and file name
imageFinalNamesParts        = split(imageInitialName,string(filesep));

% Find the label of each image file
if contains(imageFinalNamesParts(end),'In')
    categoryName = "Inclusion";
elseif contains(imageFinalNamesParts(end),'Pa')
    categoryName = "Patch";
elseif contains(imageFinalNamesParts(end),'PS')
    categoryName = "Spot";
else
    error('Unknown image category')
end

% Insert category name folder
imageFinalNamesParts(end-1) = join([imageFinalNamesParts(end-1),"Processed","_",alias,"\",categoryName],"");

% Change image file type
imageFinalNamesParts(end)   = replace(imageFinalNamesParts(end),'.bmp','.png');

% Generate images final name
imageFinalName              = join(imageFinalNamesParts,"\");

% Create the folder in case it is not exists
processedImageDirectory = fileparts(imageFinalName);
if ~exist(processedImageDirectory, 'dir')
    mkdir(processedImageDirectory)
end


% Save processed image

imwrite(imageFinal,imageFinalName);
```