Wydział Inżynierii Mechanicznej i Robotyki

Katedra Robotyki i Mechatroniki

Signal processing and identification in control of mechatronic devices

Signal processing and identification in monitoring of mechatronic devices

Topic: Deep learning in vision data interpretation

Aim: Learning image classification using deep learning approach

Scope: Image datastores; Assembly of a deep learning net; Credit assigmnent path; Various layers of CNN; *'sgdm'*, *'rmsprop'* and *'adam'* optimizers; Validation on separate datasets; Confusion matrices

Leading assistant: dr inż. Ziemowit Dworakowski

Introduction

The aim of this exercise is to solve the same problem as before (i.e. letter classification) but this time: using deep learning. In scope of the exercise we'll use the same data and the same **individual** tasks as before – but this time we'll use raw image data. It is likely that you already have access to *DeepLearningData* archive. If not – you can build it from *StudentData* archive from previous class using script provided in attachment to this instruction.

imageDatastore

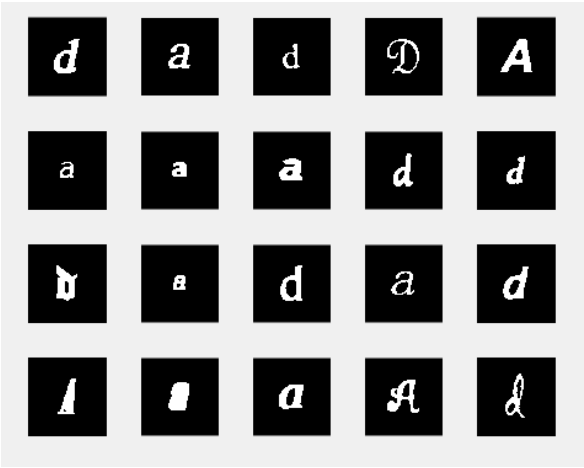
In deep learning we often use so much data that it is impossible to store that in RAM memory. Because of that normally deep learning systems access data straight from disk. In scope of this exercise we will use rather small set of data, but this principle will still be met. *imageDatastore* object can be created as follows:

```
Path = {    'DeepLearningData/Normal/A',...
            'DeepLearningData/Normal/D'};
imds = imageDatastore(Path,'IncludeSubfolders',true,'LabelSource','foldernames');
```

We've used two paths: To folders containing **"A"** and **"D"** letters, respectively. We can provide here any number of sources of data – as consecutive fields in "Path" vector. We only need to remember that folder names will became object labels – so if objects of the same class are in different folders in various locations, folder names have to be identical.

Let us check how do our dataset look like. We'll plot here 20 random images:

```
figure;
perm = randperm(length(imds.Files),20);
for i = 1:20
    subplot(4,5,i);
    imshow(imds.Files{perm(i)});
end
```



Next, we'll check how many objects of each class we have:

```
labelCount = countEachLabel(imds)
```

Similarily as before, we'll divide data into training and validation subsets. This time, however, we don't need to do that manually. We'll use a built-in function instead:

```
numTrainFiles = 400;
[imdsTrain,imdsValidation] = splitEachLabel(imds,numTrainFiles,'randomize');
```

We'll use 400 samples for training and remainder (200) for validation.

Elements building up the convolutional neural net

Our net will be designed using a *layers* vector, that will have various components of the following kinds:

```
imageInputLayer([150 150 1])
```

Input layer – it defines size of the input data (letter image in our case).

```
convolution2dLayer(3,8,'Padding','same')
batchNormalizationLayer
reluLayer
```

Convolution layer. First argument refers to the filter size (here: 3x3), second to the number of parallel filters used (here: 8). Two latter arguments ensure that the output of this layer has the same size as input. Convolution is usually followed by normalization and nonlinear activation (here: ReLU activation).

```
maxPooling2dLayer(2,'Stride',2)
```

A layer that performs *MaxPooling* operation – reducing the size of our data.

```
fullyConnectedLayer(2)
```

A fully-connected layer – if it is used as a hidden layer, we define its neuron numbers in brackets. If it is just before softmax layer – number of neurons needs to be equal to number of classes.

```
softmaxLayer
classificationLayer
```

Two final layers that map output of the last layer of the fully connected net to actual classification space.

First (not so) deep net

Let us start with a net similar to the one used before, that is, having two hidden layers with 10 neurons in each. Let us name it “MLP1”. As an activation function we'll use ReLU activation this time:

```
layers = [
    imageInputLayer([150 150 1])
    fullyConnectedLayer(10)
    reluLayer
    fullyConnectedLayer(10)
    reluLayer
    fullyConnectedLayer(2)
    softmaxLayer
    classificationLayer];
```

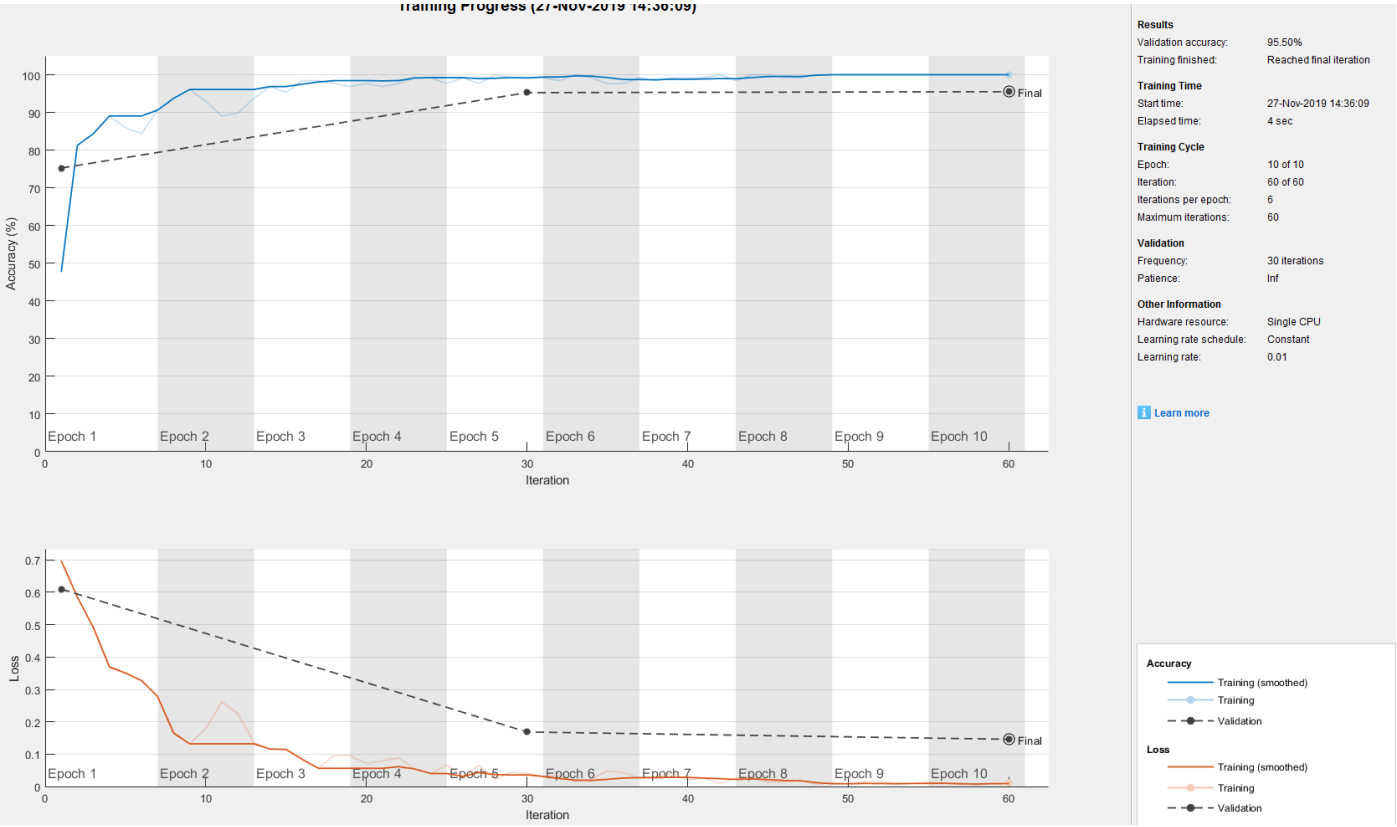
This net should now be configured for training, using the following configuration vector:

```
options = trainingOptions('sgdm', ...
    'InitialLearnRate',0.01, ...
    'MaxEpochs',10, ...
    'Shuffle','every-epoch', ...
    'ValidationData',imdsValidation, ...
    'ValidationFrequency',30, ...
    'Verbose',false, ...
```

```
    'Plots','training-progress');
```

And finally train our net:

```
    net = trainNetwork(imdsTrain,layers,options);
```



In our case we've obtained **95.5%** accuracy on a validation subset – which seems to be suspiciously high, when we take into account the fact that we use pretty simple net and don't even have any data preprocessing! Lets think on why this happened?

A problem "to think through": *(Try to answer the following questions right now. The answers will probably be provided by the LA by the end of the class):*

1 – Using no data preprocessing and no feature extraction we have very good classification accuracy. May it be caused by the fact that we initially divided our data fully randomly? Does it ensure that no overfitting occurred?
2 – What does the net have in its "world"? What is the only knowledge it did receive? Are we able to build efficient classification rules based on particular pixel positions in this problem?
3 – How many data in our dataset are truly "difficult"? How many letters are basically copies of each other?

Task 1: According to the above codes prepare a MLP1 net to solve your **individual** task. What efficiency did it receive? Is the result repeatable?

Validation of decision systems on validation subset that is taken randomly from the initial data set may prove to be risky. The most important threat is a hidden overfitting phenomenon, that is, a situation in which testing data are in fact taken from different source (e.g. different folder) but are too similar to the ones used

in training (contain the same information). A practical example of this situation may be a road sign classification task in which we'd like to distinguish two speed limit signs from each other: each with different speed. Lets assume that we've taken 100 images of sign “speed limit 70 kph” and 100 images of “speed limit 50 kph”. Let us further assume that for the “70 kph” sign all of the images were acquired with clear sky as a background while for the “50kph” sign all of the images were acquired with a green forest as a background. Now, we divide the data randomly into training and testing datasets. The algorithm can easily learn to distinguish signs from each other based only on the background of the signs. If we use a standard testing procedure based on our randomly selected dataset – we won't learn that the classification accuracy is not general.

Because of that we'll test our net based on an independent set of data. We'll use a “W2” dataset (with distorted images) to this end. Let us build image datastore:

```
Path = {    'DeepLearningData/W2/A',...
            'DeepLearningData/W2/D'};
imdsTest = imageDatastore(Path,'IncludeSubfolders',true,'LabelSource','foldernames');
```

And now test our net:

```
YPred = classify(net,imdsTest);
YValidation = imdsTest.Labels;
accuracy = sum(YPred == YValidation)/numel(YValidation)
```

In this case we've obtained 93% efficiency – so it appears that the problem that we worked so hard to solve during previous instruction – is much more straightforward that we've anticipated!

Task 2: Check the accuracy of the net in your **individual** task – when tested on “W2” dataset. Does training and testing again changes this result? Save the obtained accuracy values in the table provided in the last part of this instruction.

Convolutional neural net

Deep learning would not have much sense without layers that allow for low-level feature extraction. One of the mechanisms for that purpose is a convolution layer. Let us incorporate it to our network. The modified net we'll call CNN1

Let the layers be organized as follows:

```
layers = [
    imageInputLayer([150 150 1])
    convolution2dLayer(3,8,'Padding','same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride',2)
    convolution2dLayer(3,16,'Padding','same')
    batchNormalizationLayer
    reluLayer
    fullyConnectedLayer(2)
    softmaxLayer
    classificationLayer];
```

This net is not particularly deep *(Credit Assignment Path Depth* is only 3 in here, so only 1 more than before) – but in our example it allowed for accuracy equal to **98,5% -** Below we can find a net even deeper (let us call it CNN2):

```
layers = [
    imageInputLayer([150 150 1])
    convolution2dLayer(3,8,'Padding','same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride',2)
    convolution2dLayer(3,16,'Padding','same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride',2)
    convolution2dLayer(3,16,'Padding','same')
    batchNormalizationLayer
    fullyConnectedLayer(10)
    reluLayer
    fullyConnectedLayer(2)
    softmaxLayer
    classificationLayer];
```

Do this net provide even better result?

Task 3: In solving your **individual** task train the CNN1 and CNN2 nets based on Normal data. Test them now on "W2" dataset. What accuracy did you obtain? Does starting training from scratch and testing the nets again change the results much? Save the results in table at the end of this instruction.

Testing the nets in multiclass problems

Task 4: Check the performance of the nets in a multiclass problem. Load all the letters from Normal dataset (A,D,F,H,K,L,M,N,T and Y). Remember to modify the size of output layer so it would be equal to the new class number (10). What is the validation accuracy and what is the testing accuracy on a separate dataset (W2)? Save the results in the table present at the end of this instruction.

In order to find out which classes cause the net the most trouble, calculate a *Confusion Matrix*:

```
plotconfusion(YValidation,YPred)
```

Testing the nets in a more difficult classification task

Task 5: Check the performance of the three nets in your **individual** task – this time using raw data from database that contain backgrounds (*DeepLearningDataCaptcha).* Of course the nets should be both trained and tested on respective subsets of new dataset, so "Normal" data from *Captcha* dataset for training and validation and "W2" data from *Captcha* dataset for final testing.

If you don't have DeepLearningDataCaptcha dataset, you can easily build it by uncommenting lines of code in 2nd attachment.

Configuration of a training algorithm

Up until now we were just using deep nets in their default condition. Right now we'll investigate this a bit deeper. For instance, we could modify an optimizer (*'sgdm'*, *'rmsprop'* oraz *'adam'*) used for net training changing its type, learning rate, way of decrease of learning rate over time or regularization.

For now, let us test just one of them and lets modify a *LearningRate* parameter:

Task 6: For Task 4 problem test *LearningRate* values from this set: [0.1 0.02, 0.01, 0.005, 0.001] – how the change affects the learning time? How does it affect convergence? How does is affect the final accuracy obtained? Are the result repeatable? Is the repeatability influenced by *LearningRate* value?

Additional tasks:

Task 7: Solve a task of "Waviness detection", that is: classification whether the particular letter comes from *Normal, W1* or *W2* datasets. To this end you'll need to build a new *imageDatastore*:

Load images simultaneously from both categories, e.g. like that:

```
 Path_a = {    'DeepLearningData/Normal/A',...
           'DeepLearningData/W2/A'};
 imdsDistortion =
imageDatastore(Path_a,'IncludeSubfolders',true,'LabelSource','none');
```

Now, 'manually' adjust labels for all the samples based on order of their loading, like that:

```
 LB1 = categorical(zeros(600,1));  for k = 1:600;    LB1(k) = 'N'; end
 LB2 = categorical(zeros(600,1));  for k = 1:600;    LB2(k) = 'W'; end
 imdsDistortion.Labels = [LB1;LB2];
 imdsDistortion.Labels = setcats(imdsDistortion.Labels,{'N','W'})
```

What is the accuracy obtained for all the three nets? Why that? Does adding more letters to the sets (e.g. 'N' dataset contain 'Normal' "F","K","M" and "N" letters, 'W' dataset contains the same letters but taken from 'W1' folder) makes the task easier or harder? Is it possible to determine whether the letter unseen during training is "wavy" or not?

Task 8: For the task chosen by the LA do optimization of at least three chosen metaparameters of the net. Remember that the methods are non-deterministic – the optimization needs to be statistics-based (e.g. using boxplots)

Description of the training parameters can be found e.g. in here:
https://www.mathworks.com/help/deeplearning/ref/trainingoptions.html

Task 9: Do task 4 again, this time for the letters in *DeepLearningDataCaptcha* database. Let the evaluation be statistically-significant.

Result tables

Individual task: Classification of …. and …. letters		
		Validation (200 samples)
MLP1	Try 1	
	Try 2	
	Try 3	
CNN1	Try 1	
	Try 2	
	Try 3	
CNN2	Try 1	
	Try 2	
	Try 3	

"All the letters"		
		Validation (200 samples)
MLP1	-	
CNN1	-	
CNN2	-	

"Captcha", individual task: Classification of …. and …. letters		
		Validation (200 samples)
MLP1	-	
CNN1	-	
CNN2	-	

"Captcha", all the letters:		
		Validation (200 samples)
MLP1	-	
CNN1	-	
CNN2	-	

Attachment 1

Code that builds folders for *imageDatastore* based on *StudentData:*

```
Letters = {'A','D','F','H','K','L','M','N','T','Y'};
for k = 1:length(Letters)
    Letter = Letters{k}
    BS = load(strcat('StudentData/Features',Letter));
    mkdir(strcat('DeepLearningData/Normal/',Letter));
    for k = 1:length(BS.Data);           Name = strcat('Image',num2str(k),'.png'); ...
        I = BS.Data(k).Normal.Image;         S = size(I);
        offY = floor((150 - S(1))/2);    offX = floor((150 - S(2))/2);
        New = logical(zeros(150,150));   New(offY:offY+S(1)-1,offX:offX+S(2)-1) = I;
        % New = Captcha(New);
        imwrite(New,strcat('DeepLearningData/Normal/',Letter,'/',Name));
    end
    mkdir(strcat('DeepLearningData/W1/',Letter));
    for k = 1:length(BS.Data);           Name = strcat('Image',num2str(k),'.png'); ...
        I = BS.Data(k).W1.Image;         S = size(I);
        offY = floor((150 - S(1))/2);    offX = floor((150 - S(2))/2);
        New = logical(zeros(150,150));   New(offY:offY+S(1)-1,offX:offX+S(2)-1) = I;
        % New = Captcha(New);
        imwrite(New,strcat('DeepLearningData/W1/',Letter,'/',Name));
    end
    mkdir(strcat('DeepLearningData/W2/',Letter));
    for k = 1:length(BS.Data);           Name = strcat('Image',num2str(k),'.png'); ...
        I = BS.Data(k).W2.Image;         S = size(I);
        offY = floor((150 - S(1))/2);    offX = floor((150 - S(2))/2);
        New = logical(zeros(150,150));   New(offY:offY+S(1)-1,offX:offX+S(2)-1) = I;
        % New = Captcha(New);
        imwrite(New,strcat('DeepLearningData/W2/',Letter,'/',Name));
    end
end
```

Attachment 2

Captcha function that builds base with backgrounds:

```
function [ImageRes] = Captcha(ImgIn)
    ImgBG = rgb2gray(imread('D:\NAUKA\__DYDAKTYKA\2019_2020 (PhD 5)\Instrukcje\Database\Exported\BG.png'));
    Image1 = uint8(100*ImgIn);
    Image1 = imgaussfilt(Image1);
    ImgBG = ImgBG.*0.8;
    Cx = randi(size(ImgBG,1)-151);
    Cy = randi(size(ImgBG,2)-151);
    Image2 = ImgBG(Cx:Cx+149,Cy:Cy+149);
    ImageRes = Image1+Image2;
end
```