Faculty of Mechanical Engineering and Robotics

Department of Robotics and Mechatronics

Basics of AI and Deep Learning

Course for Mechatronic Engineering with English as instruction language

Instruction 11:

Deep Convolutional Neural Network

You will learn how to build your own Deep Convolutional Neural Network from scratch – and how to configure it for classification of raw images. You will also compare this solution with the approach from last week and you will also try different tasks and configurations showing the range of possibilities you might want to dive into later.

Additional materials:

Course supervisor:
Ziemowit Dworakowski, zdw@agh.edu.pl

Instruction authors:
Ziemowit Dworakowski, zdw@agh.edu.pl

Introduction

The aim of this exercise is to perform an operation analogous to the previous one (i.e. letter classification), but this time using deep learning. The same data and the same tasks will be used (the **individual** task is defined in exactly the same way as in the previous laboratory), but the data source will now be folders containing raw image files.

Most likely, you already have access to the dataset, which only needs to be unpacked (*DeepLearningData* and *DeepLearningDataCaptcha*). If not, it can be built from the structure used in the previous class by using the script provided as an attachment to this instruction.

imageDatastore

In deep learning, datasets are often so large that it would be impossible to store all of them in RAM. Therefore, the standard approach in deep learning is to work with images stored directly on disk. In this exercise, we will use a very small dataset, but the underlying principle remains the same. The *imageDatastore* object can be created in the following way:

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

Here, we loaded two paths: one pointing to the folder containing the letter "**A**" and the other to the folder containing the letter "**D**". Any number of data sources (i.e., different folders) can be provided here as subsequent elements in the *Path* vector. It is important to remember that the names of the folders become the labels of the objects — so if objects of the same category are located in different places, the names of their parent folders must be identical. Let's check how some example images from our dataset look. We will display twenty random images (Figure 1):

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

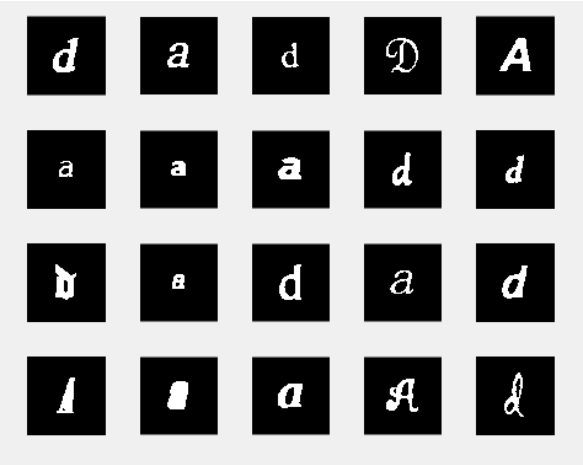

Figure 1 – random images from our database

Next, we will check how many objects of each class are present in the dataset:

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

As before, we will randomly split the data into training and validation subsets. This time, however, we do not need to do it manually — we will simply use a built-in function:

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

We will use 400 objects for training, and the remaining 200 will go into the validation set.

Components of a deep network

We will construct our network using a *layers* vector, employing the following components:

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

The input layer must match the size of the image.

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

This is a convolutional layer. The first argument defines the filter size (in this case: 3x3), and the second specifies the number of filters (i.e., how many convolutional layers operate in parallel — here, 8). The last two arguments ensure that the size of the resulting feature map matches the input size. Convolution is usually used "in a bundle" with normalization and a nonlinear activation function.

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

This layer implements the *MaxPooling* algorithm, which reduces the size of the resulting feature maps.

```
fullyConnectedLayer(2)
```

This is a fully connected layer. If it serves as a hidden layer, the number of neurons is defined in the parentheses. If it serves as the output layer the number of neurons must match the number of classes being classified.

```
softmaxLayer
classificationLayer
```

These final two layers convert the output of the last fully connected layer into a proper classification.

First self-constructed (still relatively shallow) network

Let's begin by building a network similar to the one used previously — that is, with two hidden layers, each containing 10 neurons. We will call this network **MLP1**. This time, we will use the ReLU (Rectified Linear Unit) activation function:

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

This network still needs to be configured for training — for that, we will use the following options vector:

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

Finally, the network can be trained. The results will be presented in Figure 2:

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

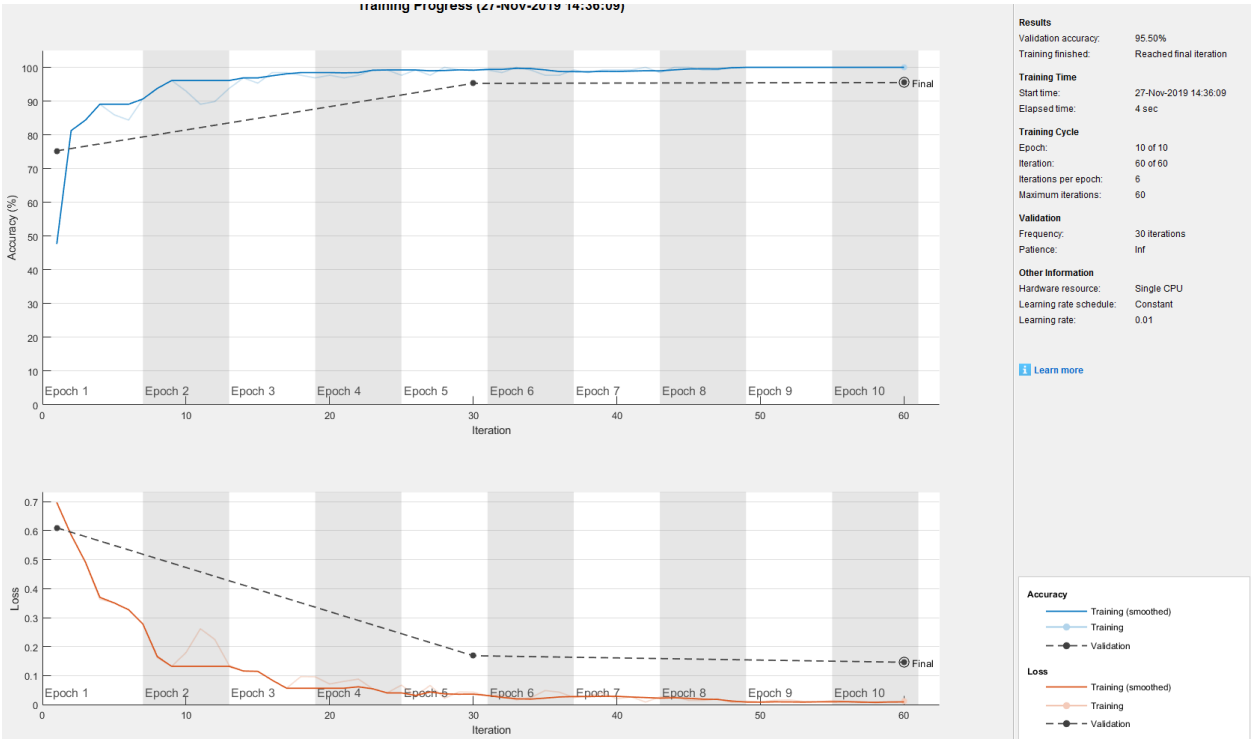

Figure 2 – results of training of our network

In our case, we achieved a validation accuracy of 95.5% during training — which seems like an excellent result, considering that we are using a relatively simple network and are not performing any feature extraction! Let's take a moment to reflect on why this might be happening.

Problem for reflection:
(Think carefully about each of the following questions — answers will likely be discussed at the end of the session):

1 - A high classification accuracy was achieved without any preprocessing or feature extraction. In this context, what is the significance of the fact that the datasets were initially split completely at random? Does this provide any guarantee of generalization?

2 - What constitutes the "world" of the trained neural network? What are the only pieces of information it received? Is it possible, in our case, to construct simple classification rules based on the values of specific image pixels?

3 - What percentage of the dataset consists of truly problematic and non-standard data? How many letters appear to significantly overlap with one another?

Task 11.1: Adapt the neural network according to the above scheme (MLP1) for classification of your **individual** task. What classification accuracy was achieved? Is the result repeatable? What is its variability? Store the results in the table at the end of the instruction

Validation of decision systems on a subset randomly selected from the original dataset carries a number of risks — the most important being the possibility of hidden overfitting. This refers to a situation in which the test data, although formally different from the training data (e.g., coming from a different file), are in practice identical in terms of the information they contain. A practical example of such a situation could be the classification of traffic signs. Imagine we take 100 photos of the same traffic sign that we want to classify. Each photo differs slightly in position, but all of them include a yellow background, because the sign is mounted on a yellow wall. If we now divide these images into training and test sets, and the algorithm learns that the easiest way to recognize this specific sign is by identifying the background it appears against, we may obtain 100% detection accuracy on the so-called "independent" validation set — but it will not translate to reliable performance in practice for other signs of the same type placed in different locations.

For this reason, we will now evaluate our network using a genuinely independent test set. To do this, we will use the "W2" dataset. First, let's load the data:

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

Next, we test the network on this data:

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

In our case, the achieved accuracy is 93% — so it turns out that the problem we worked so hard to solve in the previous session is actually much simpler than we initially assumed.

Task 11.2: Test the performance of the network trained to solve your individual task using the "W2" dataset. What accuracy was achieved? Does retraining the network from scratch and testing it again change the result? Record the obtained results in the table located at the end of the instruction.

Convolutional Neural Network

Deep learning would not be meaningful without methods dedicated to extracting low-level features. One such method is the use of convolutional layers. Let's try to expand our neural network to enable even more effective performance. The extended network will be called **CNN1**. Let the layers of the network 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 is not a particularly deep network (its **Credit Assignment Path Depth** is only 3 — just one more than in the previous example), but it allowed us to achieve a validation accuracy of **98.5%**, which means that the error rate in our case has been more than halved. Let's now try using an even deeper network (we will call it **C2**):

```
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];
```

Will this version bring any further improvement? What about the performance on the test dataset?

Task 11.3: Test the performance of the network trained to solve your **individual** task using the "W2" dataset. What accuracy was achieved? Does retraining the network from scratch and testing it again change the result? Record the obtained results in the table located at the end of the instruction.

Network performance on a challenging classification problem

Task 11.4: Examine how the networks perform in classifying your **individual** task — this time using the raw dataset that includes background. Load the data from the *DeepLearningDataCaptcha* database and once again compare the performance of the three tested networks for the two-letter classification problem. Record the obtained results in the table located at the end of the instruction.

If you do not have access to the DeepLearningDataCaptcha database, you can easily generate it by uncommenting the relevant sections in Attachment 1 and using the function provided in Attachment 2.

Działanie sieci dla problemu wieloklasowego

Task 11.5: Let's now see how our networks perform in a multi-class classification task. Load all letters from the "Normal" class available in the captcha dataset (A, D, F, H, K, L, M, N, T, and Y). Remember to change the size of the final layer in the network to match the number of classes (now 10). What validation and independent test accuracy do the previously configured networks — MLP1, CNN1, and CNN2 — achieve? Record the obtained results in the table located at the end of the instruction.

To determine which classes are most problematic for the networks, generate a Confusion Matrix, for example, using the following command:

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

Training algorithm configuration

To configure the training algorithm, a number of parameters can be selected. Up to this point, we have been using the default settings without any modifications. Let's now delve deeper into this topic. Currently, we have three solvers to choose from: '*sgdm*', *'rmsprop'*, and *'adam'*. For each of them, we can adjust specific parameters related to momentum, regularization, learning rate, learning rate decay, etc.

Let's begin by testing one of these — the **learning rate**:

Task 11.6: For the classification problem from Task 6, and using the network that achieved the best result, test the influence of the *LearningRate* parameter. Consider which values should be measured, and then try to identify values of *LearningRate* at which "training breaks down." Present the results of your experiment in graphical form.

Distortion detection in letters

Task 11.7:

Implement a task focused on detecting "waviness" in letters — that is, distinguishing letters from the *Normal* dataset from those in the *W1* and *W2* datasets. To achieve this, you will need to modify the data loading process to construct the imageDatastore object accordingly.

Load images from both categories simultaneously, for example:

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

Then manually assign categories in the order in which the data was loaded, for example:

```
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 classification accuracy was achieved by each of the three networks? Why did you obtain that particular result? Does adding more letters to both datasets (e.g., the "N" set includes undistorted letters "F," "K," "M," "N," and the "W" set includes the same letters in distorted versions) make the task easier or harder? Is it possible to train a network to predict whether a letter that was not previously seen in the dataset is distorted or not?

Additional tasks

Task 11.8: For a problem proposed by the instructor, perform an optimization of at least three selected training metaparameters of the network. Remember that this optimization must have statistical significance — the results should be presented in the form of boxplots, as used in the previous instruction. Descriptions of training parameters (to help you select three metaparameters for optimization) can be found here:

https://www.mathworks.com/help/deeplearning/ref/trainingoptions.html

Task 11.9: Take your convolutional net trained to recognize captcha letters. Then take a number of correctly classified examples (e.g. 10) from our testing dataset and use them for further tests. Try to modify these examples first by gradually added noise and then by more advanced challenges (e.g. rotation or progressively adding different objects to the image). Check at which point the network starts to misclassify these examples.

Task 11.10: Train your deep network to calculate features for your images. Let the input be the binary image of a letter and output consist of number of features that you used to classify your data in your previous laboratory. Then check how well your network operates and whether it can calculate features for unseen letter samples. In order to perform this task you will need to replace final layers of your net with a *regressionLayer* and define targets as features.

Result tables

Individual task, classification of letters: ..... and			
		Validation	**Independent test**
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	**Independent test**
MLP1	**-**		
CNN1	**-**		
CNN2	**-**		

“Captcha”, Individual task, classification of letters: ..... and			
		Validation	**Independent test**
MLP1	**-**		
CNN1	**-**		
CNN2	**-**		
" Captcha”, All the letters:			
		Validation	**Independent test**
MLP1	**-**		
CNN1	**-**		
CNN2	**-**		

Attachment 1

Kod zamieniający strukturę *StudentData* na znormalizowane obiekty pogrupowane w foldery nadające się do wczytania jako *imageDatastore:*

```
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

Funkcja *Captcha* tworząca "popsutą" bazę:

```
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
```