Wydział Inżynierii Mechanicznej i Robotyki

Katedra Robotyki i Mechatroniki

Przetwarzanie sygnałów i identyfikacja w sterowaniu urządzeń mechatronicznych

Temat: Zastosowanie naturalnych metod obliczeniowych w klasyfikacji danych wizyjnych

Cel ćwiczenia: Porównanie uczenia głębokiego i płytkiego, klasyfikacja liter z zastosowaniem głębokiej konwolucyjnej sieci neuronowej

Zagadnienia: ImageDatastore, Budowanie głębokich sieci neuronowych, Sieć konwolucyjna,

Prowadzący: dr inż. Ziemowit Dworakowski

Wprowadzenie

Celem niniejszego ćwiczenia będzie wykonanie operacji analogicznej jak poprzednio (tj. klasyfikacji liter) lecz z wykorzystaniem uczenia głębokiego. W ramach ćwiczenia wykorzystane zostaną te same dane i te same zadania (zadanie **własne** zdefiniowane jest tak samo jak w poprzednim ćwiczeniu) - lecz źródłem danych będą tym razem foldery z obrazami zapisanymi bezpośrednio.

Najprawdopodobniej mają Państwo już dostęp do bazy danych którą wystarczy rozpakować (*DeepLearningData)* - jeśli nie, ze struktury wykorzystywanej na poprzednich zajęciach zbudować ją można korzystając ze skryptu dostępnego w załączniku do niniejszej instrukcji.

imageDatastore

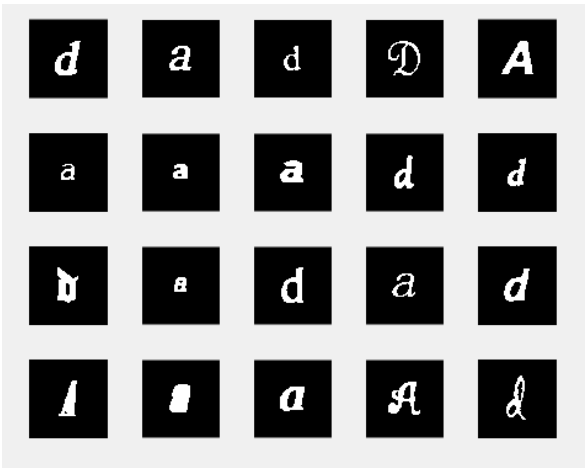
W uczeniu głębokim danych jest często tak dużo, że nie byłoby możliwości przechowywać ich wszystkich w pamięci RAM. W związku z tym standardem w przypadku uczenia głębokiego jest korzystanie z obrazów umieszczonych bezpośrednio na dysku. My w ramach ćwiczenia korzystać będziemy z bardzo małego zbioru danych - ale zasada działania pozostaje ta sama. Obiekt *imageDatastore* utworzyć możemy w następujący sposób:

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

Tutaj wczytaliśmy dwie ścieżki: do folderu zawierającego litery "**A**" oraz do folderu zawierającego litery "**D**". Źródeł danych (tzn. różnych folderów) można tutaj podać dowolną ilość - w ramach kolejnych pól w wektorze *Path* - należy jedynie pamiętać, że nazwy folderów stają się etykietami obiektów - jeśli więc obiekty tej samej kategorii znajdują się w kilku miejscach, nazwy ich folderów nadrzędnych muszą być tożsame.

Sprawdźmy, jak wyglądają przykładowe obrazy z naszego zbioru. Wyświetlimy dwadzieścia losowych obrazów:

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



A następnie sprawdzimy ile obiektów jakiej klasy znajduje się w zbiorze:

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

Analogicznie jak poprzednio, podzielimy dane losowo na podzbiór uczący i walidacyjny. Tym razem jednak nie musimy robić tego ręcznie, skorzystamy jedynie z gotowej funkcji:

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

Wykorzystamy 400 obiektów do uczenia, pozostałe (200) trafią do zbioru walidacyjnego.

Elementy składowe głębokiej sieci

Naszą sieć tworzyć będziemy za pomocą wektora *layers*, korzystając z następujących komponentów:

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

Warstwa wejściowa musi być równa rozmiarowi obrazu

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

Warstwa konwolucyjna, pierwszy argument odpowiada za rozmiar filtra (tu: 3x3), drugi za ilość filtrów (tzn. ile równolegle połączonych warstw konwolucyjnych ma działać w sieci - tutaj 8), dwa ostatnie argumenty zapewniają, że wielkość otrzymanej mapy jest równa wielkości obszaru wejściowego.

Konwolucja działa zwykle "w pakiecie" z normalizacją i nieliniową funkcją aktywacji

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

Warstwa odpowiadająca za realizację algorytmu *MaxPooling*, zmniejszającego rozmiar otrzymanych map cech

```
fullyConnectedLayer(2)
```

Warstwa w pełni połączona - jeśli ma pełnić rolę warstwy ukrytej, ilość jej neuronów określamy w nawiasie. Jeśli warstwy wyjściowej (przed softmax) - ilość neuronów musi być równa ilości klasyfikowanych obiektów.

```
softmaxLayer
classificationLayer
```

Dwie finalne warstwy zamieniające wyjście z ostatniej w pełni połączonej warstwy na własciwą klasyfikację.

Pierwsza samodzielnie zbudowana (dość płytka jeszcze) sieć

Spróbujmy na początek zbudować sieć analogiczną do tej wykorzystywanej poprzednio - tzn. posiadającą dwie warstwy ukryte po 10 neuronów w każdej. Nazwiemy tą sieć MLP1 Jako funkcję aktywacji

wykorzystamy tym razem funkcję ReLU (Rectified Linear Unit):

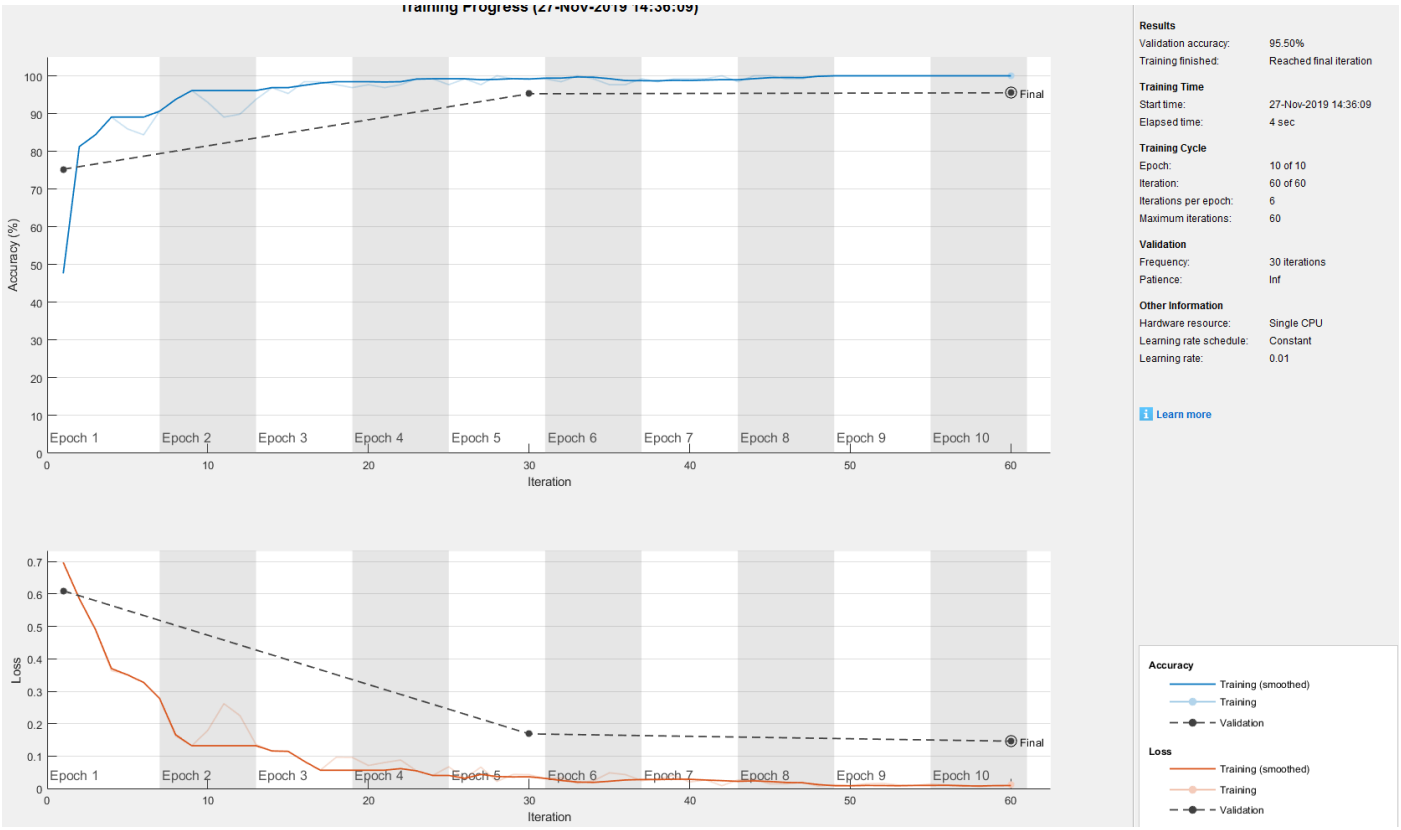
```
layers = [
    imageInputLayer([150 150 1])
    fullyConnectedLayer(10)
    reluLayer
    fullyConnectedLayer(10)
    reluLayer
    fullyConnectedLayer(2)
    softmaxLayer
    classificationLayer];
```

Tak przygotowaną sieć należy jeszcze skonfigurować do treningu - do tego posłuży nam następujący wektor opcji:

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

I ostatecznie wytrenować:

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



W naszym przypadku uzyskaliśmy w treningu skuteczność na zbiorze walidacyjnym sięgającą **95.5%** - co wydaje się być wspaniałym wynikiem, biorąc pod uwagę, że korzystamy tutaj ze stosunkowo prostej sieci i nie wydobywamy żadnych cech! Zastanówmy się nad tym dlaczego tak się dzieje?

Problem "do przemyślenia": *(Zastanów się przez chwilę nad każdym pytaniem - na końcu zajęć prawdopodobnie pojawią się odpowiedzi):*

1 - Nie stosując wstępnego przetwarzania danych i nie wydobywając cech uzyskano wysoką skuteczność klasyfikacji. Jakie znaczenie ma w tym kontekście to, że zbiory początkowo były dzielone zupełnie losowo? Czy mamy dzięki temu gwarancję generalizacji?
2 - Z czego się składa "świat" uczonej sieci neuronowej? Jakie są jedyne informacje jakie otrzymała? Czy da się zbudować proste reguły klasyfikacyjne oparte o wartości konkretnych pikseli obrazu w naszym przypadku?
3 - Jaki procent danych w zbiorze stanowią dane rzeczywiście problematyczne i niestandardowe? Jak dużo liter wydaje się na siebie nakładać?

Zadanie 1: Przystosuj sieć neuronową według powyższego schematu (MLP1) do klasyfikacji **własnego** zadania. Jaką skuteczność uzyskano? Czy wynik jest powtarzalny? Jaka jest jego zmienność?

Walidacja systemów decyzyjnych na zbiorze wydzielonym zupełnie losowo z początkowego zbioru danych niesie za sobą szereg zagrożeń z których podstawowym jest możliwość pojawienia się ukrytego przeuczenia - tzn. sytuacji, w której dane testowe są wprawdzie różne od danych uczących (np. poprzez fakt pobrania ich z innego pliku), ale w praktyce są im tożsame w kontekście zawieranej informacji. Przykładem praktycznym takiej sytuacji może być klasyfikacja znaków drogowych. Wyobraźmy sobie, że wykonujemy zdjęcie jednego znaku który chcemy sklasyfikować 100 razy. Każde zdjęcie różni się nieco położeniem, ale wszystkie zawierają żółte tło, bo znak znajdował się na tle żółtej ściany. Jeśli teraz te zdjęcia podzielimy na uczące i testowe a algorytm nauczy się, że ten konkretny znak najłatwiej rozpoznać poprzez tło na jakim się znajduje uzyskamy 100% skuteczności detekcji również na "niezależnym" zbiorze walidacyjnym - ale nie przełoży się to na skuteczność w praktyce dla wszystkich innych znaków z tej samej kategorii lecz umieszczonych w różnych miejscach.

Z tego względu wykonamy dla naszej sieci test oparty na niezależnym zbiorze danych. Wykorzystamy w tym celu zbiór "W2". Wczytajmy na początek dane:

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

A następnie przetestujmy sieć na ich podstawie:

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

W naszym przypadku uzyskana skuteczność to 93% - okazuje się więc, że problem który z takim wysiłkiem rozwiązywaliśmy na poprzednim spotkaniu okazuje się dużo prostszy niż założyliśmy.

Zadanie 2: Sprawdź skuteczność uzyskaną przez sieć wytrenowaną do realizowania **własnego** zadania na podstawie zbioru "W2" Jaką skuteczność uzyskano? Czy uruchomienie treningu od nowa i ponowne przetestowanie zmieni ten wynik? Zapisz uzyskane wyniki w tabeli znajdującej się na końcu instrukcji.

Konwolucyjna sieć neuronowa

Uczenie głębokie nie miałoby sensu bez istnienia metod dedykowanych wydobywaniu cech niskiego poziomu. Jedną z takich metod jest stosowanie warstw konwolucyjnych. Spróbujmy rozbudować naszą sieć neuronową tak, aby umożliwić jej jeszcze skuteczniejsze działanie. Rozbudowaną sieć nazwiemy sieć CNN1

Niech warstwy sieci zorganizowane zostaną następująco:

```
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];
```

Nie jest to sieć bardzo głęboka (wartość *Credit Assignment Path Depth* to jedynie 3 - czyli zaledwie o 1 więcej niż w poprzednim przykładzie - ale pozwoliło na uzyskanie skuteczności na zbiorze walidacyjnym na poziomie **98,5% -** czyli błąd w naszym przypadku spadł ponad dwukrotnie. Zastosowanie sieci jeszcze głębszej (nazwijmy ją sieć 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];
```

Czy w tym przypadku uda się uzyskać jakąś poprawę? Co w przypadku zbioru testowego?

Zadanie 3: Sprawdź skuteczność uzyskaną przez sieć wytrenowaną do realizowania **własnego** zadania na podstawie zbioru "W2" Jaką skuteczność uzyskano? Czy uruchomienie treningu od nowa i ponowne przetestowanie zmieni ten wynik? Zapisz uzyskane wyniki w tabeli znajdującej się na końcu instrukcji.

Działanie sieci dla trudnego problemu klasyfikacyjnego

Zadanie 4: Sprawdź jak poradzą sobie sieci w klasyfikacji **<u>własnego</u>** zadania - tym razem z bazy "surowej", zawierającej tło. Wczytaj dane z bazy *DeepLearningDataCaptcha* i ponownie porównaj ze sobą trzy testowane sieci. Zapisz otrzymane wyniki w tabeli znajdującej się na końcu instrukcji.

Jeśli nie posiadasz bazy DeepLearningDataCaptcha możesz ją łatwo zbudować poprzez odkomentowanie fragmentów z załącznika 1 i wykorzystanie funkcji z załącznika 2

Działanie sieci dla problemu wieloklasowego

Zadanie 5: Zobaczmy jak poradzą sobie nasze sieci w przypadku problemu wieloklasowego. Wczytajmy wszystkie litery z klasy Normal dostępne w bazie (A,D,F,H,K,L,M,N,T oraz Y). Należy pamiętać, aby zmienić rozmiar ostatniej warstwy sieci na równy ilości klas (teraz 10). Jaką skuteczność w walidacji i niezależnym teście uzyskały skonfigurowane wcześniej sieci MLP1, CNN1 oraz CNN2? Zapisz otrzymane wyniki w tabeli znajdującej się na końcu instrukcji.

Aby określić które klasy sprawiają sieci największy problem, wyznacz *Confusion Matrix* np w ten sposób:

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

Konfiguracja algorytmu uczącego

W celu skonfigurowania algorytmu uczącego wybrać można szereg różnych wartości. Do tej pory stosowaliśmy wersję *default* - bez modyfikowania czegokolwiek. Spróbujmy teraz wgryźć się w ten problem nieco bliżej. Do wyboru mamy jak na razie trzy solvery: *'sgdm'*, *'rmsprop'* oraz *'adam'*. Dla każdego z nich możemy wybrać wartości konkretnych parametrów powiązanych z momentem, regularyzacją, krokiem optymalizacji *(Learning rate)*, spadkiem kroku optymalizacji w czasie id.

Przetestujmy jeden z nich, tj. krok optymalizacji:

Zadanie 6: Dla problemu rozwiązywanego w ramach zadania 6, dla sieci która osiągnęła najlepszy rezultat przetestuj wartości parametru *LearningRate* z następującego zbioru: [0.1 0.02, 0.01, 0.005, 0.001] - jak zmiana parametru wpływa na czas treningu? Jak zmiana parametru wpływa na ostatecznie uzyskiwany wynik? Czy te wyniki są powtarzalne? Czy powtarzalność zależy od wartości parametru *LearningRate*?

Wykrywanie zniekształceń w literach

Zadanie 7: Zrealizuj zadanie wykrywania zafalowań w literach - tzn. odróżniania liter ze zbioru *Normal* od liter ze zbioru *W1* i *W2*. W tym celu zmodyfikować będzie trzeba sposób pobierania danych aby zbudować obiekt *imageDatastore*:

Wczytaj obrazy z obu kategorii jednocześnie, np tak:

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

A następnie ręcznie przypisz kategorie w kolejności wczytanych danych, np. tak:

```
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'})
```

Jaką skuteczność udało się uzyskać każdą z trzech sieci? Dlaczego właśnie taką? Czy dodanie większej ilości liter do obu zbiorów (tzn. zbiór "N" zawiera np. niezafalowane litery "F","K","M","N" a zbiór "W" zawiera te same litery w wersji zafalowanej) ułatwia czy utrudnia zadanie? Czy da się nauczyć sieć przewidywania czy litera niewidziana wcześniej w bazie jest zafalowana czy nie?

Zadanie dodatkowe

Zadanie 8: Dla zaproponowanego przez prowadzącego problemu wykonaj optymalizację przynajmniej trzech wybranych metaparametrów treningu sieci - pamiętaj o tym, aby ta optymalizacja miała sens statystyczny (uzyskiwane wyniki przedstaw w formie wykresów pudełkowych znanych z poprzedniej instrukcji)

Opis parametrów treningu (potrzeby w celu wybrania trzech metaparametrów do optymalizacji) znaleźć można tu: *https://www.mathworks.com/help/deeplearning/ref/trainingoptions.html*

Zadanie 9: Sprawdź jak poradzą sobie sieci w klasyfikacji wszystkich liter dostępnych w bazie (A,D,F,H,K,L,M,N,T oraz Y) - tym razem z bazy "surowej", zawierającej tło. Wczytaj dane z bazy *DeepLearningDataCaptcha* i ponownie porównaj ze sobą trzy testowane sieci. Niech wykonane badania mają znaczenie statystyczne: zweryfikuj powtarzalność wszystkich trzech sieci.

Tabele z wynikami

Własne zadanie, klasyfikacja liter ..... oraz			
		Walidacja	**Niezależny test**
MLP1	**Próba 1**		
	Próba 2		
	Próba 3		
CNN1	**Próba 1**		
	Próba 2		
	Próba 3		
CNN2	**Próba 1**		
	Próba 2		
	Próba 3		

"Wszystkie litery"			
		Walidacja	**Niezależny test**
MLP1	**-**		
CNN1	**-**		
CNN2	**-**		

"Problem Captcha", własne zadanie: Klasyfikacja liter ..... oraz			
		Walidacja	**Niezależny test**
MLP1	**-**		
CNN1	**-**		
CNN2	**-**		

"Problem Captcha, wszystkie litery"			
		Walidacja	**Niezależny test**
MLP1	**-**		
CNN1	**-**		
CNN2	**-**		

Załącznik 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
```

Załącznik 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
```