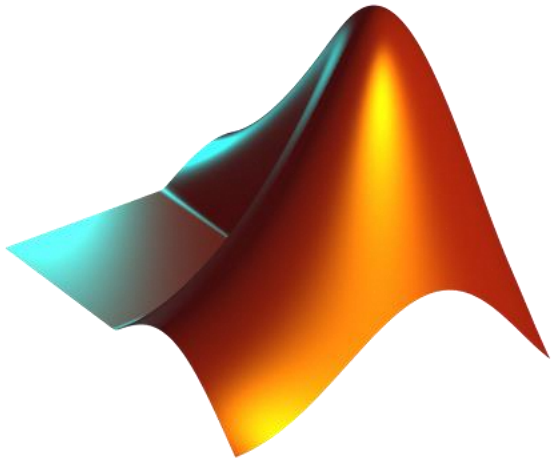
# Programowanie w zastosowaniach inżynierskich



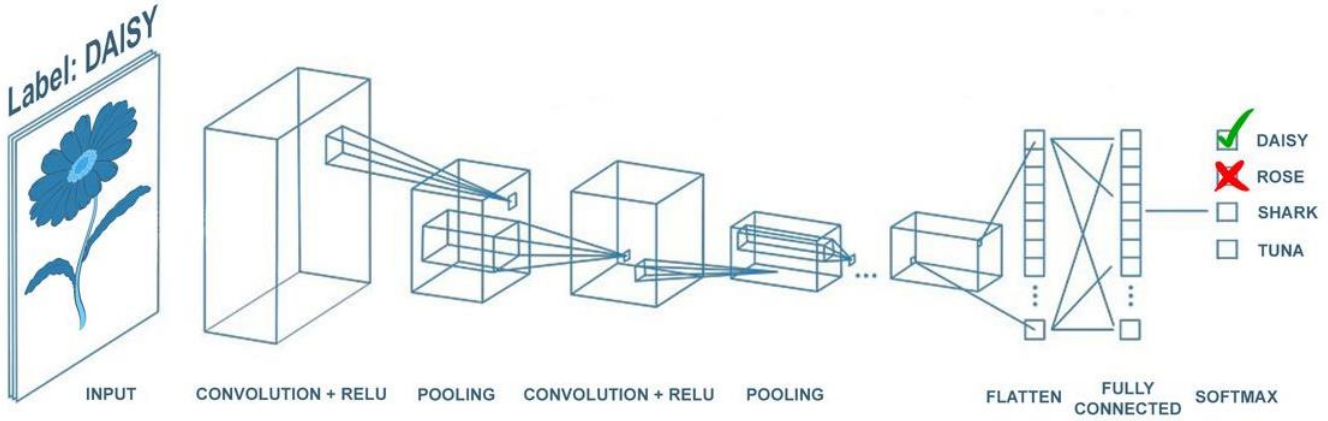
## Sieci neuronowe
## Deep Learning Toolbox

# Deep Learning Toolbox

**Deep Learning Toolbox** to rozszerzenie środowiska MATLAB o funkcje do projektowania, implementacji, wizualizacji i symulacji sieci neuronowych.

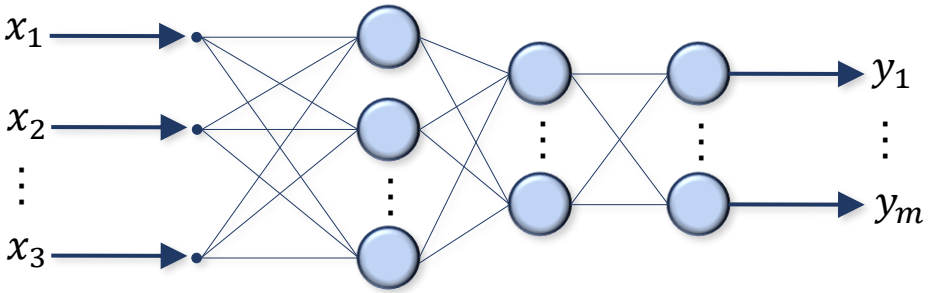
## Najważniejsze cechy modułu

- Wsparcie dla sieci neuronowych o różnych typach i architektach.
- Graficzny interfejs użytkownika do tworzenia i symulowania sieci neuronowych.
- Wsparcie dla obliczeń równoległych oraz z wykorzystaniem GPU.
- Zestaw funkcji do przygotowania danych dla sieci neuronowej.
- Modularna reprezentacja sieci, pozwalająca na zadawanie dowolnej liczby warstw wejściowych i dowolnej liczby połączeń między warstwami.
- Zbiór bloków Simulink do symulowania sieci neuronowych.



## Sztuczna sieć neuronowa

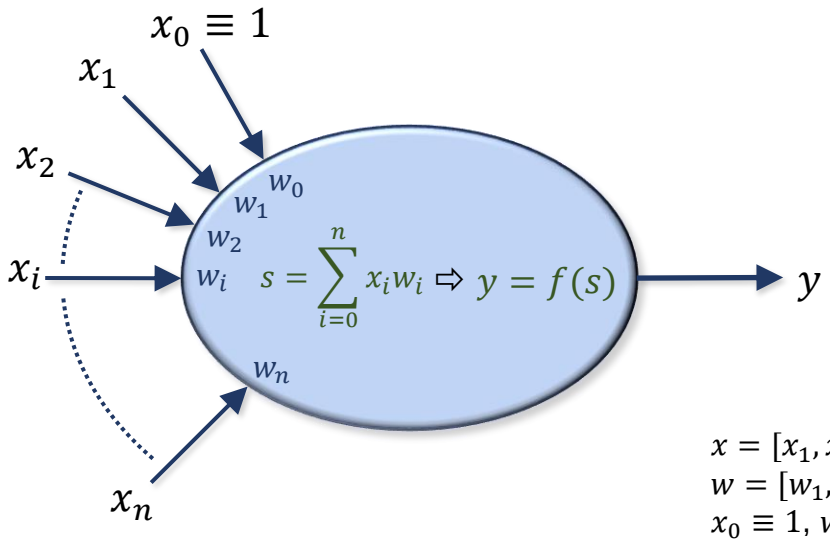
to system przetwarzający informacje, skonstruowany z połączonych elementów, nazywanych neuronami, którego budowa i zasada działania została oparta na fragmentach biologicznego systemu nerwowego.




## Cechy sztucznych sieci neuronowych

1. Zdolność uczenia na podstawie przykładów.
2. Automatyczne uogólnianie zdobytej wiedzy (generalizacja).
3. Umiejętność rozwiązywania problemów bez ich formalizacji.
4. Brak konieczności przyjmowania założeń dotyczących rozwiązywanego problemu.

**Neuron** to podstawowy składnik sieci neuronowej, element przetwarzający informacje w sposób wzorowany na funkcjonowaniu biologicznej komórki neuronowej.




$x = [x_1, x_2, \ldots, x_n]$ – wektor sygnałów wejściowych
$w = [w_1, w_2, \ldots, w_n]$ – wektor wag
$x_0 \equiv 1$, $w_0$ – wyraz wolny, tzw. bias neuronu
$s$ – sumaryczne pobudzenie komórki
$f$ – funkcja aktywacji, $y$ – sygnał wyjściowy

## Etapy obliczeń

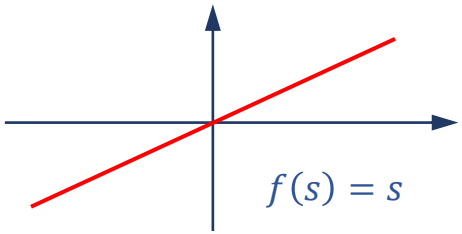
1. Agregacja sygnałów wejściowych $x$ z uwzględnieniem wag $w$ i wyznaczenie sumarycznego pobudzenia neuronu $s$.
2. Wygenerowanie sygnału wyjściowego przez funkcję aktywacji $f$.

## Funkcja aktywacji

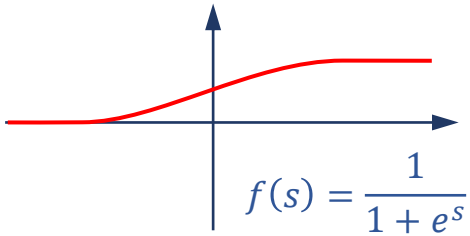
wyznacza wartość sygnału wyjściowego neuronu na podstawie wartości sumarycznego pobudzenia (suma iloczynu wejść i wag).

## Typowe funkcje aktywacji

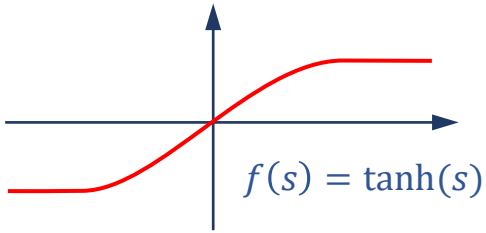
liniowa



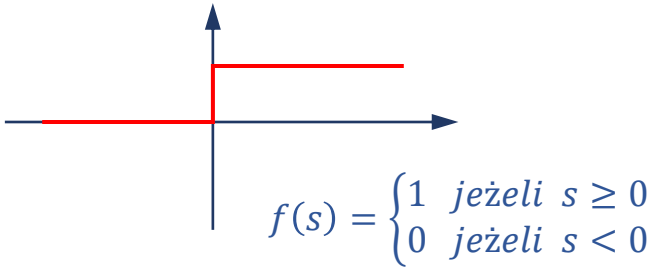

sigmoidalna (`logsig`)



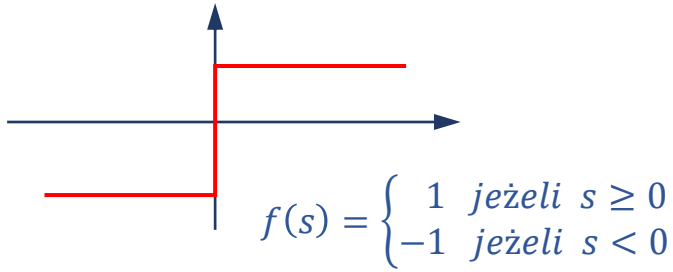

tangensoidalna (`tansig`)



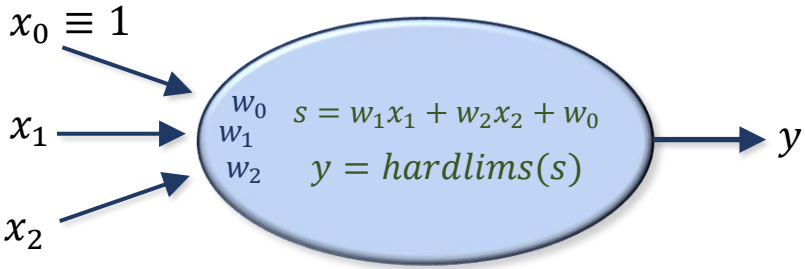

unipolarny skok (`hardlim`)




bipolarny skok (`hardlims`)



## Neoruon o dwóch wejściach z biasem (wyraz wolny, waga $w_0$)

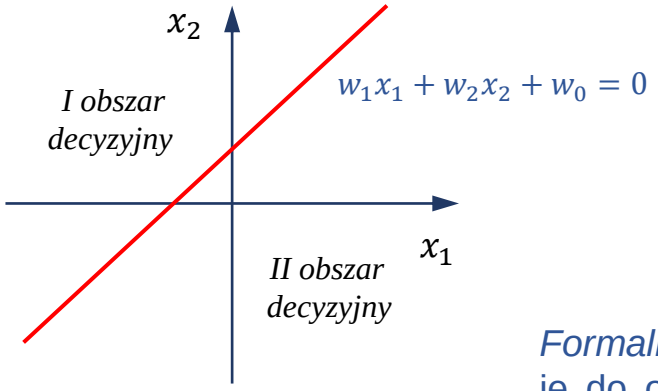



Pobudzenie neuronu:

$$s = w_1x_1 + w_2x_2 + w_0$$

Powierzchnia decyzyjna:

$$w_1x_1 + w_2x_2 + w_0 = 0$$



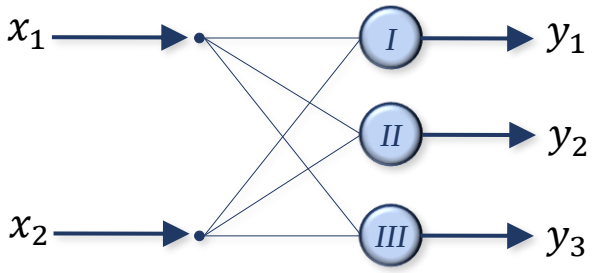

Zbiór punktów, dla których $s = 0$ wyznacza powierzchnię decyzyjną (w tym przypadku prostą) dzielącą przestrzeń wejść na dwa obszary decyzyjne o wartościach $s > 0$ i $s < 0$. Funkcja aktywacji przypisuje sygnał wyjściowy (w tym przypadku 1 lub $-1$) sygnałom wejściowym $(x_1, x_2)$ leżącym w odpowiednim obszarze.

*Formalnie*: neuron klasyfikuje sygnały wejściowe $(x_1, x_2)$ przypisując je do odpowiedniej klasy (w tym przypadku 1 lub $-1$). Położenie powierzchni decyzyjnej (sposób klasyfikacji) zależy od doboru wag.

*Uwaga: w przypadku ogólnym ($n$ sygnałów wejściowych) powierzchnia decyzyjna jest hiperpłaszczyzną w $n$ wymiarowej przestrzeni wejść.*

## Sieć jednowarstwowa o dwóch wejściach i trzech neuronach

Neurony z biasem, funkcje aktywacji `hardlims`

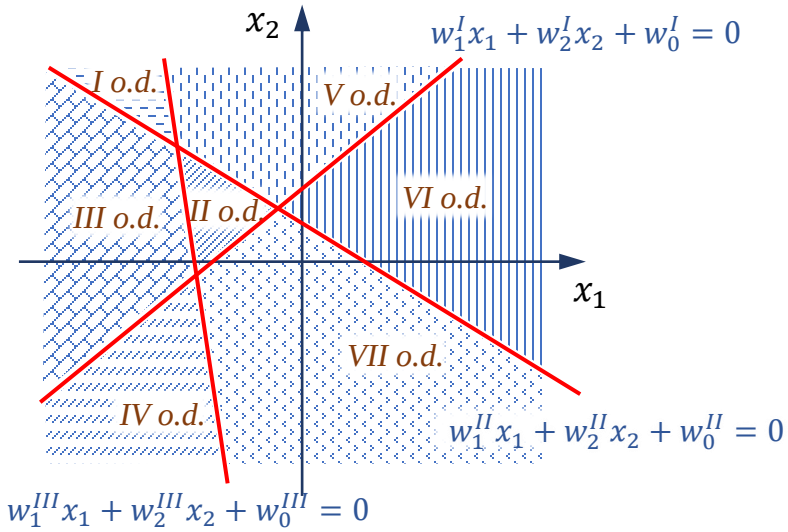



Wagi neuronów: $w^I = [w_0^I, w_1^I, w_2^I]$,

$w^{II} = [w_0^{II}, w_1^{II}, w_2^{II}]$, $w^{III} = [w_0^{III}, w_1^{III}, w_2^{III}]$

Sygnały wyjściowe: $[1,1,1]$, $[1,1,-1]$, $[1,-1,1]$, ...

## Przestrzeń wejść i powierzchnie decyzyjne

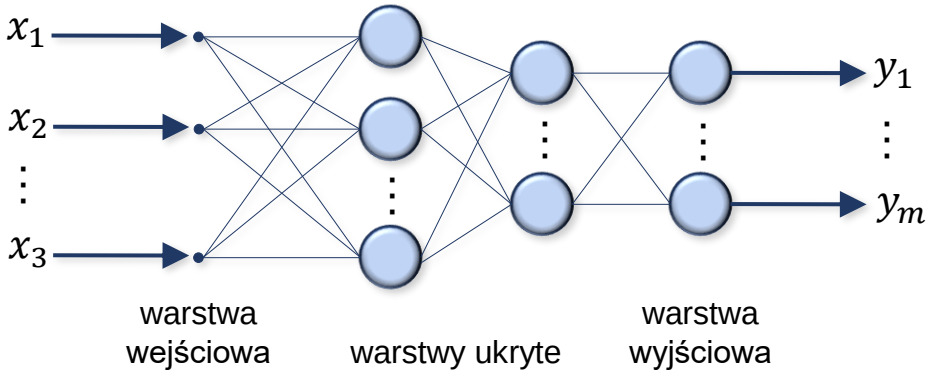



- Każdy neuron tworzy jedną powierzchnię decyzyjną
- Powierzchnie decyzyjne wyznaczają obszary decyzyjne (klasy obiektów)
- Każdemu obszarowi decyzyjnemu (klasie) odpowiada unikalna sekwencja sygnałów wyjściowych $y = [y_1, y_2, y_3]$

# Sieci wielowarstwowe (jednokierunkowe)

**Sieć wielowarstwowa** jest zbudowana z kilku warstw neuronów. Cechy:

- neurony z tej samej warstwy nie są ze sobą połączone,
- każdy neuron warstwy poprzedniej jest połączony ze wszystkimi neuronami warstwy następnej,
- kolejne warstwy neuronów dokonują klasyfikacji wyników warstwy poprzedniej.



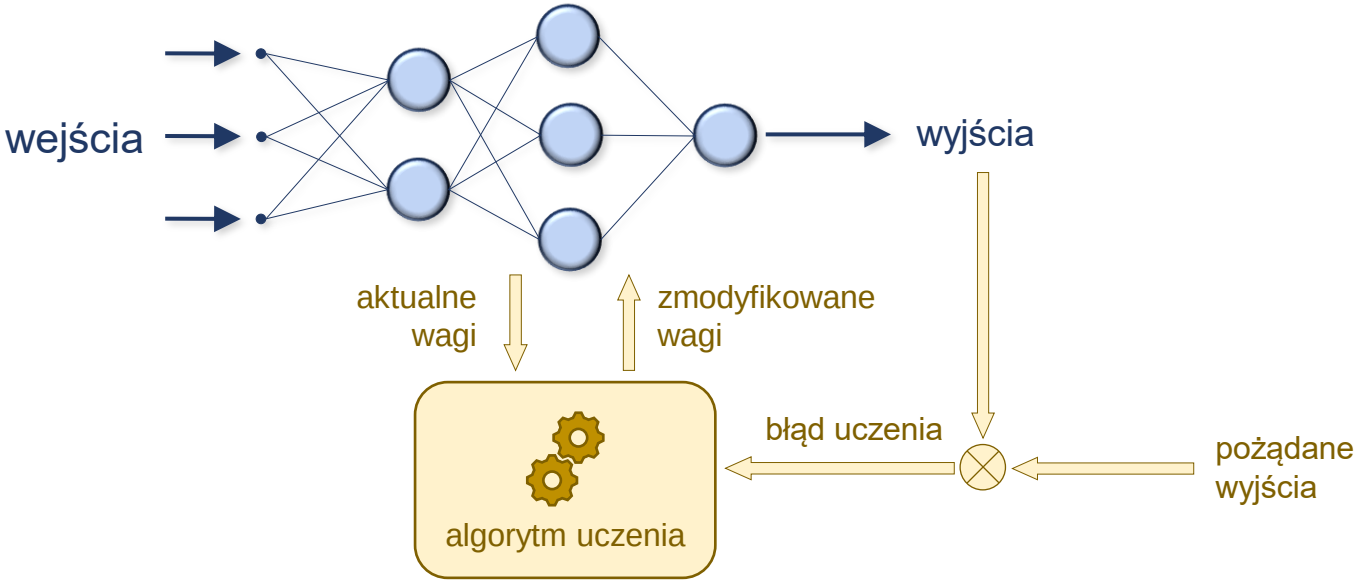

Warstwa wejściowa rozprowadza sygnały wejściowe do neuronów pierwszej warstwy ukrytej, nie bierze bezpośredniego udziału w wypracowaniu odpowiedzi sieci.

Warstwa wyjściowa określa sygnał wyjściowy (odpowiedź) całej sieci.

Warstwy ukryte przetwarzają dane wejściowe (poszukują zależności w danych wyjściowych) w taki sposób, żeby uzyskane wyniki były przydatne do opracowania odpowiedzi na warstwie wyjściowej.

# Uczenie sieci neuronowej

**Uczenie sieci** polega na prezentacji pewnego zbioru przykładów prawidłowych rozwiązań rozpatrywanego problemu (tzw. danych uczących) i dopasowaniu sieci (zazwyczaj dobraniu wag neuronów składowych) w taki sposób, aby jej odpowiedź była jak najbardziej zbliżona do odpowiedzi wzorcowej.



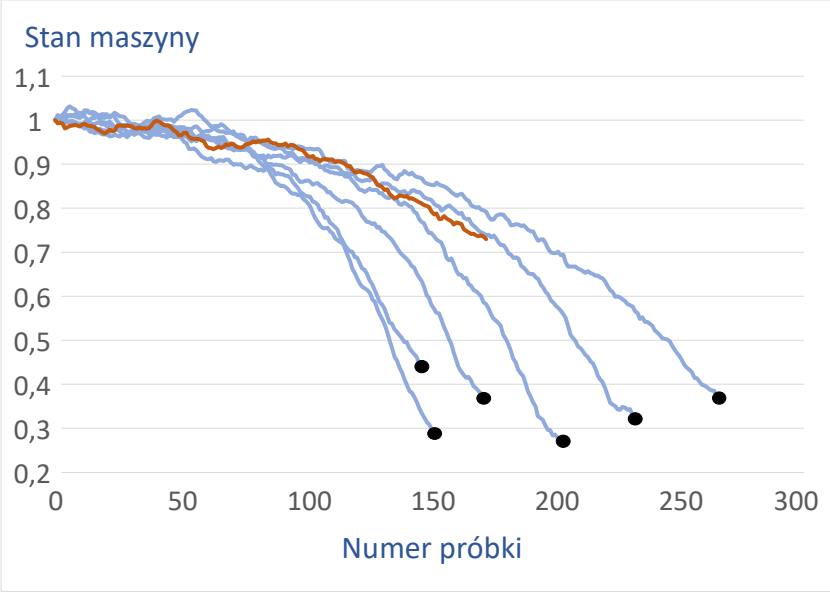

Uczenie jest procesem iteracyjnym (wielokrotnie powtarzanym). W każdym kroku dokonywana jest drobna korekta wag, która prowadzi do lepszego dopasowania sieci. Metoda zmiany wag podczas procesu uczenia nazywana jest algorytmem uczenia.

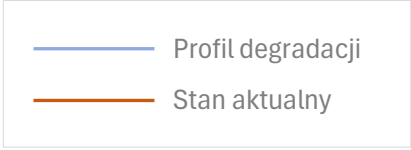
# Przykład – predykcja awarii

**RUL (Remaining Useful Life)** to pozostały czas użytkowania, wskaźnik określa żywotność maszyn i jest wykorzystywany w systemie predykcyjnego utrzymania ruchu.

**Podejście tradycyjne** (model podobieństwa)

- Podczas pracy rejestrowane są odczyty czujników, które określają stan maszyny.
- Zmiany stanów poprzedzające awarię przechowywane są jako profile degradacji
- Oszacowanie wartości wskaźnika RUL jest przeprowadzone przez znalezienie najbardziej podobnego profilu degradacji (możliwe wykorzystanie sieci neuronowej).




Przykład na podstawie:

Mądry K., Pająk I., *Zastosowanie uczenia maszynowego w predykcyjnym utrzymaniu ruchu*, Inżynieria produkcji, Badania w inżynierii mechanicznej, vol. XX, Wydawnictwo Naukowe Instytutu Inżynierii Mechanicznej Uniwersytetu Zielonogórskiego, Zielona Góra 2023

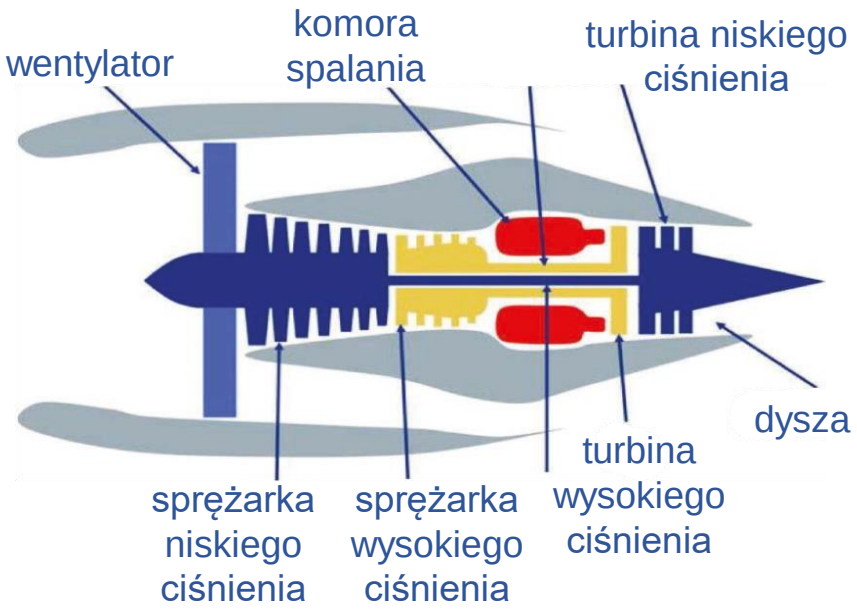
https://iim.uz.zgora.pl/badania/monografie/inzynieria-produkcji

# Predykcja awarii – dane

## NASA Turbofan Jet Engine Data Set

Zbiór zawiera 4 podzbiory danych: FD001, FD002, FD003 i FD004 opisujące degradację silnika turbowentylatorowego.

Dane zostały wygenerowane za pomocą opracowanego przez NASA symulatora C-MAPSS (Commercial Modular Aero-Propulsion System Simulation).




## Załadowanie bazy

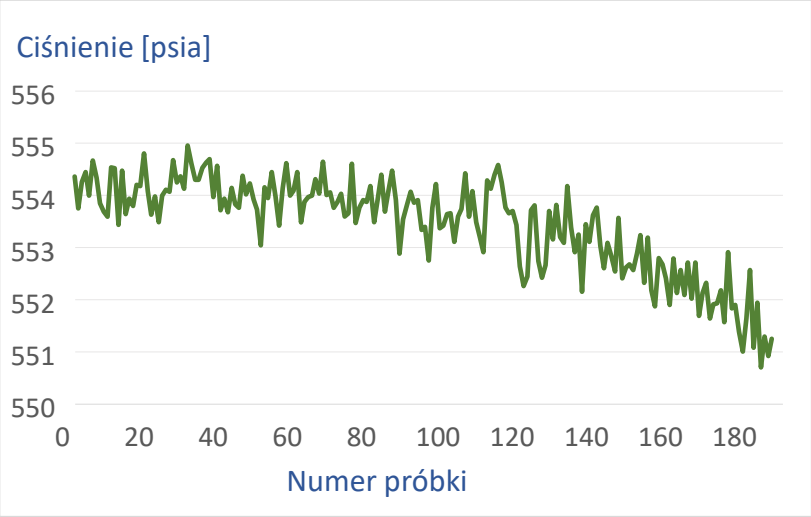
```
fname = matlab.internal.examples.downloadSupportFile( ...
  "nnet","data/TurbofanEngineDegradationSimulationData.zip");
unzip(fname, "")
```

*Uwaga: po załadowaniu dane zostały uzupełnione o dwie dodatkowe kolumny: wartość wskaźnika RUL oraz flagę bRUL opisującą stan zagrożenia awarią (przyjęto, że bRUL = (RUL <= 30)), uzupełniony zbiór FD001 został zapisany w pliku FD001.mat.*

## NASA Turbofan Jet Engine Data Set – podzbiór FD001

Oryginalny zbiór FD001 zawiera dane 200 silników dla których symulowana była usterka, w plikach train_FD001.txt i test_FD001.txt (podzbiory do uczenia i testowania) zapisane są:

- identyfikator symulowanego silnika,
- numer próbki,
- 3 parametry opisujące warunki operacyjne,
- sygnały z 21 czujników monitorujących temperaturę, ciśnienie, prędkość turbiny oraz wyciek paliwa z silnika.




| id | nr | warunki operacyjne | | | sygnały z czujników | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | op1 | op2 | op3 | s1 | s2 | … | s20 | s21 |
| 1 | 1 | -0,0007 | -0,0004 | 100 | 518,67 | 641,82 | | 39,06 | 23,4190 |
| 1 | 2 | 0,0019 | -0,0003 | 100 | 518,67 | 642,15 | | 39 | 23,4240 |
| 1 | 3 | -0,0043 | 0,0003 | 100 | 518,67 | 642,35 | | 38,95 | 23,3440 |
| 1 | 4 | 0,0007 | 0 | 100 | 518,67 | 642,35 | | 38,88 | 23,3740 |
| | | | | | | | | | |
| 100 | 200 | -0,0032 | -0,0005 | 100 | 518,67 | 643,85 | | 38,37 | 23,0522 |

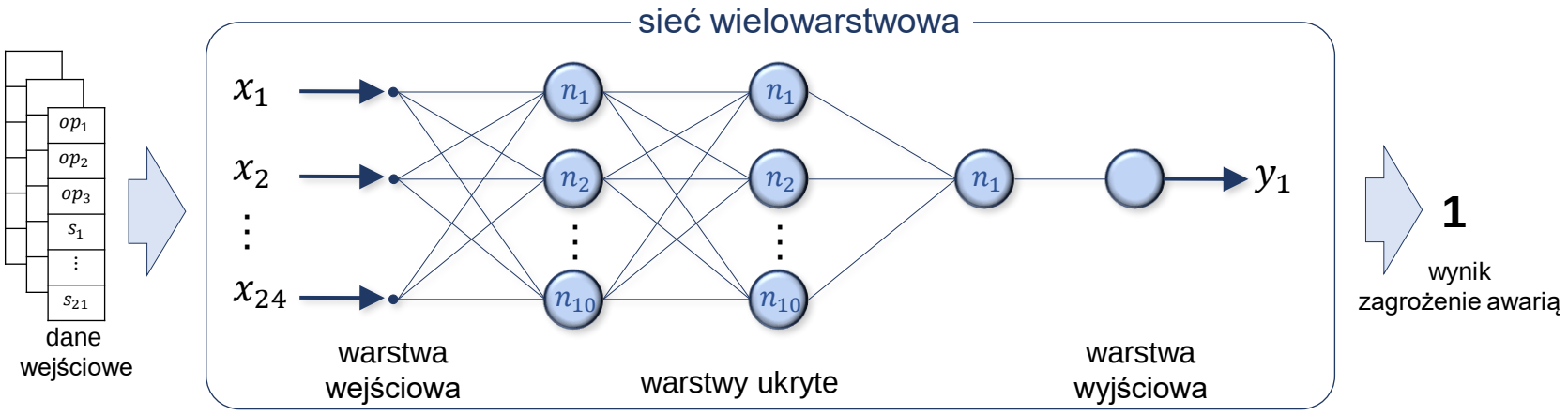
# Deep Learning Toolbox – sieć wielowarstwowa

Jednokierunkową wielowarstwową sieć neuronową w Deep Learning Toolbox konstruuje się wykorzystując funkcję feedforwardnet. Najważniejszym parametrem funkcji jest tablica określająca rozmiar warstw ukrytych sieci. Liczba wejść sieci i liczba neuronów warstwy wyjściowej jest określana w oparciu o zbiór uczący podawany albo podczas konfiguracji sieci (z wykorzystaniem funkcji configure) albo na etapie wywołania procesu uczenia sieci (z wykorzystaniem funkcji train).

## Struktura sieci

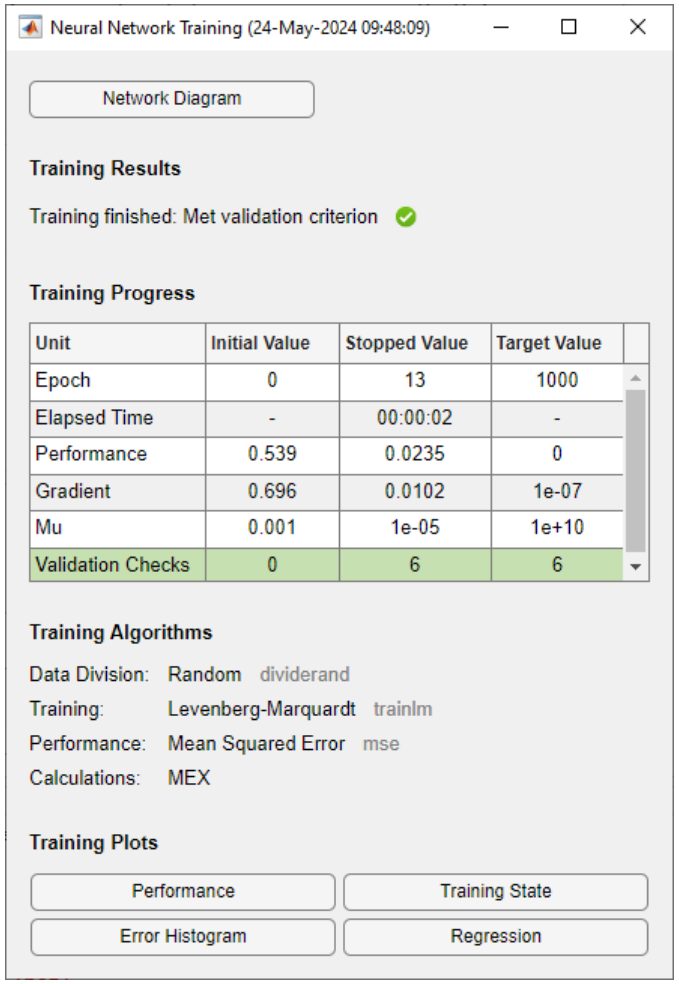
```
net = feedforwardnet([10, 10, 1]);

load('fd001.mat')
XTrain = dsTrain(:,3:26)';
YTrain = dsTrain(:,27)';

net = configure(net, XTrain, YTrain);
```

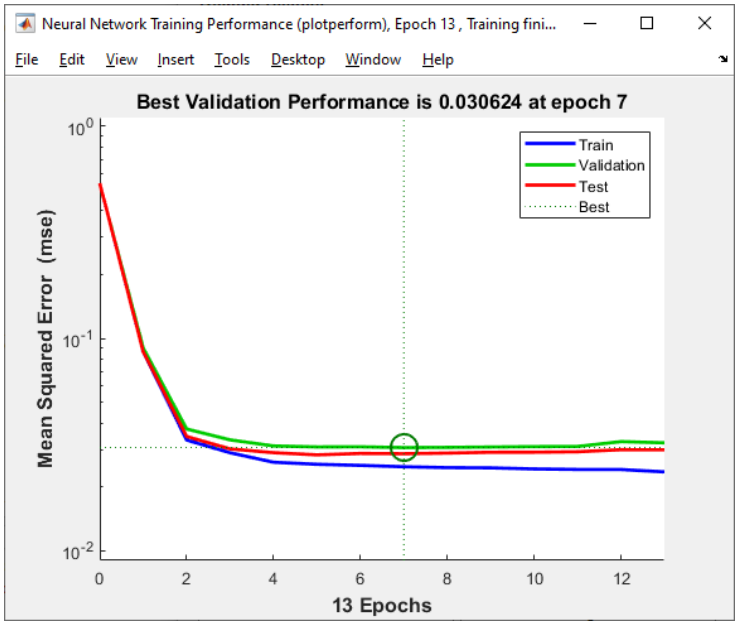
*Uwaga: neurony warstw ukrytych otrzymują funkcję aktywacji 'tansig', dodatkowo tworzona jest warstwa wyjściowa zbudowana z neuronów liniowych.*



# Deep Learning Toolbox – uczenie sieci

```
net = train(net, XTrain, YTrain);
```



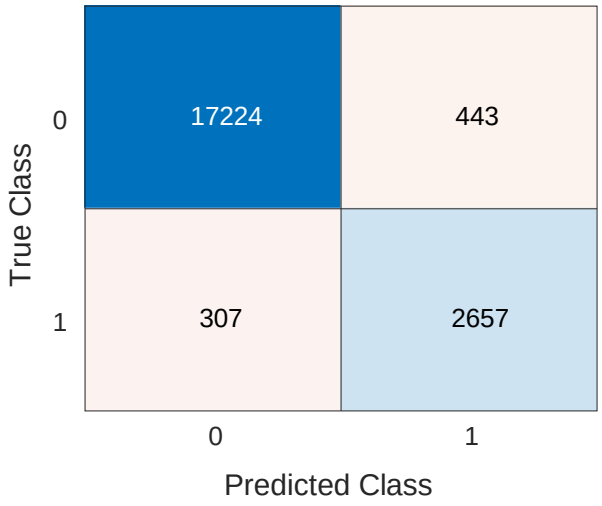




**Mean Squared Error** to błąd średniokwadratowy, stanowi ocenę jakości uczenia, jest średnią kwadratów błędów dla wszystkich wzorców wykorzystywanych podczas uczenia sieci.

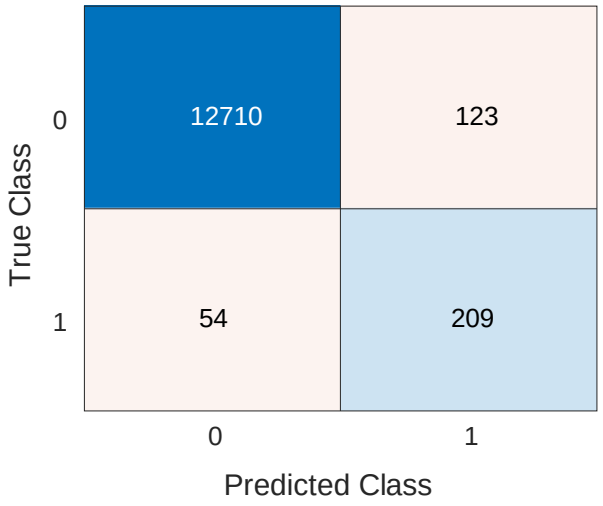
# Deep Learning Toolbox – ocena skuteczności

```
YPred = round(sim(net, XTrain));        % klasyfikacja stanów
correct = (YPred==YTrain);              % prawidłowo rozpoznane
accTrain = sum(correct)/numel(YTrain) % dokładność
figure; confusionchart(YPred, YTrain) % macierz pomyłek

YPred = round(sim(net, XTest));
correct = (YPred==YTest);
accTest = sum(correct)/numel(YTest)
figure; confusionchart(YPred, YTest)
```

zbiór: XTrain, YTrain



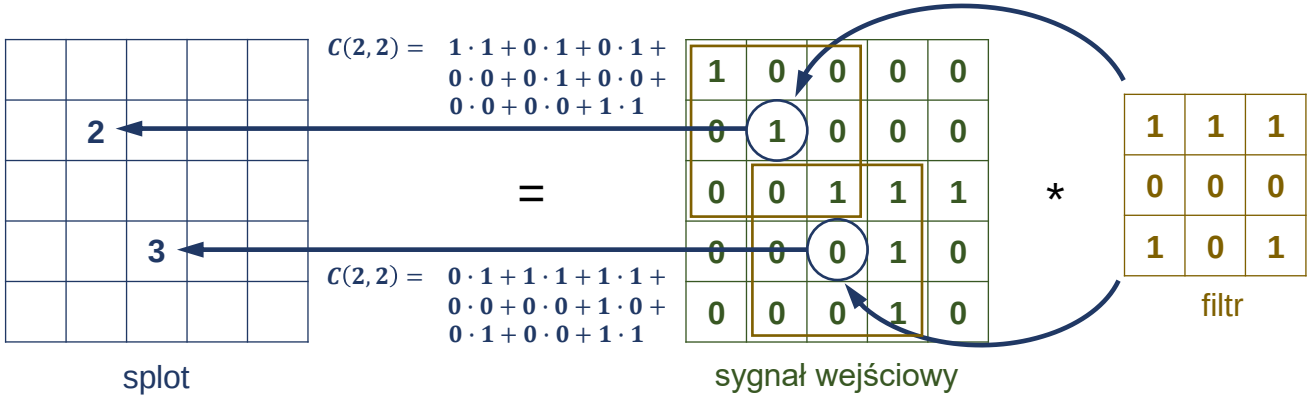

zbiór: XTest, YTest



**Konwolucyjna sieć neuronowa (CNN)** to sieć zawierająca co najmniej jedną warstwę konwolucyjną. Najważniejszą cechą sieci tego typu jest zdolność do wydobywania istotnych cech z surowych (nieprzetworzonych) danych wejściowych.

**Konwolucja (splot)** jest operacją matematyczną zdefiniowaną dla dwóch funkcji lub sygnałów, które te funkcje reprezentują. W konwolucyjnych sieciach neuronowych wykorzystuje się dyskretną implementację splotu, gdzie sygnał wejściowy sieci (np. wektor lub tablica) jest splatana z tzw. filtrem, którego postać jest dopasowana do cechy, która powinna zostać wydobyta z sygnału wejściowego.

**Implementacja splotu dla sygnałów dwuwymiarowych** (np. obraz)




*Uwaga: elementu splotu $C(i,j)$ powstaje przez "nałożenie" filtra na tablicę sygnału wejściowego i wyznaczenie sumy iloczynów odpowiadających sobie elementów. Komplety splot wyznacza się przesuwając filtr nad kolejnymi elementami sygnału $B$ i powtarzając operacje.*

# Konwolucja – przykład

## Filtr A – wykrywanie linii pionowych

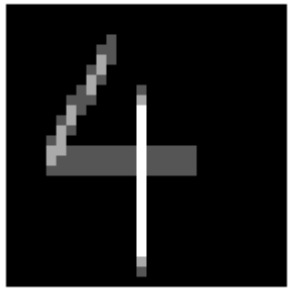
sygnał wejściowy

*

| | | |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 1 | 0 |
| 0 | 1 | 0 |

filtr A

=



splot

## Filtr B – wykrywanie linii poziomych

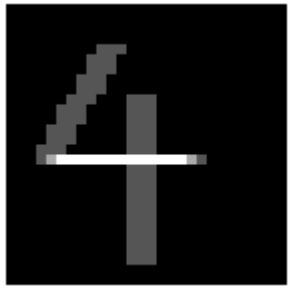
sygnał wejściowy

*

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 0 | 0 | 0 |

filtr B

=



splot

*Uwaga: w sieci konwolucyjnej filtry są elementami warstwy konwolucyjnej. Ich współczynniki traktowane są jak wagi neuronów i podlegają dopasowaniu w procesie uczenia, więc sieć sama decyduje które cechy sygnałów wejściowych są istotne w danym przypadku.*

# Deep Learning Toolbox – rozpoznawanie obrazów

**MNIST database** (ang. Modified National Institute of Standards and Technology)

Baza danych zawierająca 60000 obrazów odręcznie pisanych cyfr przeznaczona do testowania systemów automatycznego przetwarzania i rozpoznawania obrazów. Każda cyfra została znormalizowana do rozmiaru 28x28 pikseli i przekształcona na obraz w skali szarości. MATLAB udostępnia zbiór 10000 obrazów z oryginalnej bazy MNIST.

## Załadowanie bazy MNIST

```
path = fullfile(matlabroot,'toolbox','nnet', 'nndemos', ...
       'nndatasets', 'DigitDataset');
imds = imageDatastore(path, 'IncludeSubfolders',true, ...
       'LabelSource', 'foldernames');
```

*Uwaga: `imageDatastore` jest specjalną strukturą MATLABA przeznaczoną do przechowywana zbiorów danych graficznych. Sieci neuronowe tworzone przy użyciu Deep Learning Toolbox akceptują dane zapisane w tym formacie.*

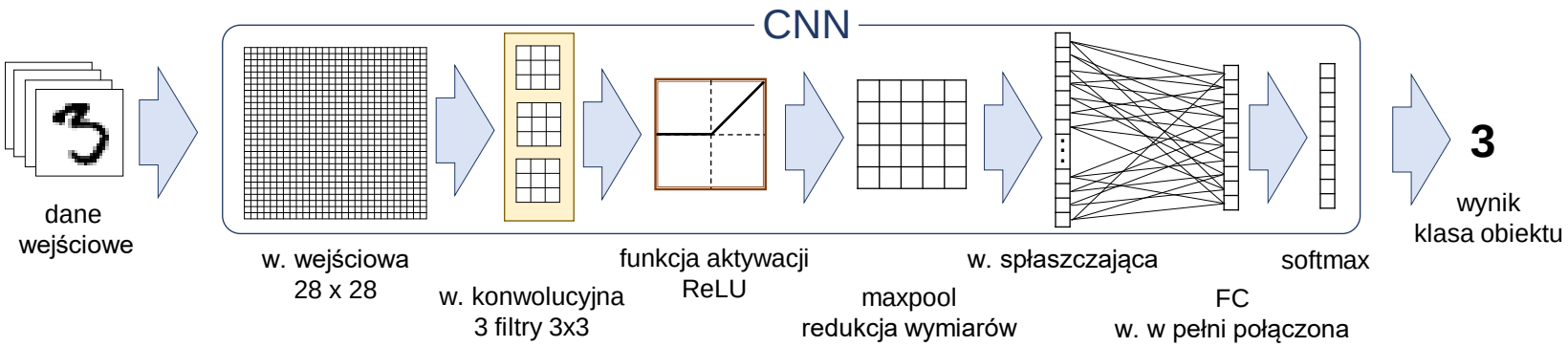
# Deep Learning Toolbox – definicja CNN

Sieć neuronową w Deep Learning Toolbox konstruuje się budując tablicę zawierającą kolejne warstwy wchodzące w skład sieci. Warstwy określonego typu są tworzone z wykorzystaniem specjalizowanych funkcji toolbox'a.

## Struktura sieci

```
layers = [ imageInputLayer([28 28 1])
           convolution2dLayer([3 3], 3, "Padding", "same")
           reluLayer
           maxPooling2dLayer(2, "Stride", 2)
           flattenLayer
           fullyConnectedLayer(10)
           softmaxLayer
           classificationLayer ];
```

*Uwaga: warstwa "softmax" określa prawdopodobieństwo przynależności do każdej z klas, wynikiem jest klasa z najwyższym prawdopodobieństwem*



# Deep Learning Toolbox – uczenie sieci

Jakość uczenia sieci należy przetestować na zestawie danych, które nie były używane podczas procesu uczenia, więc zbiór danych jest dzielony na dane uczące i testowe.

## Przygotowanie danych

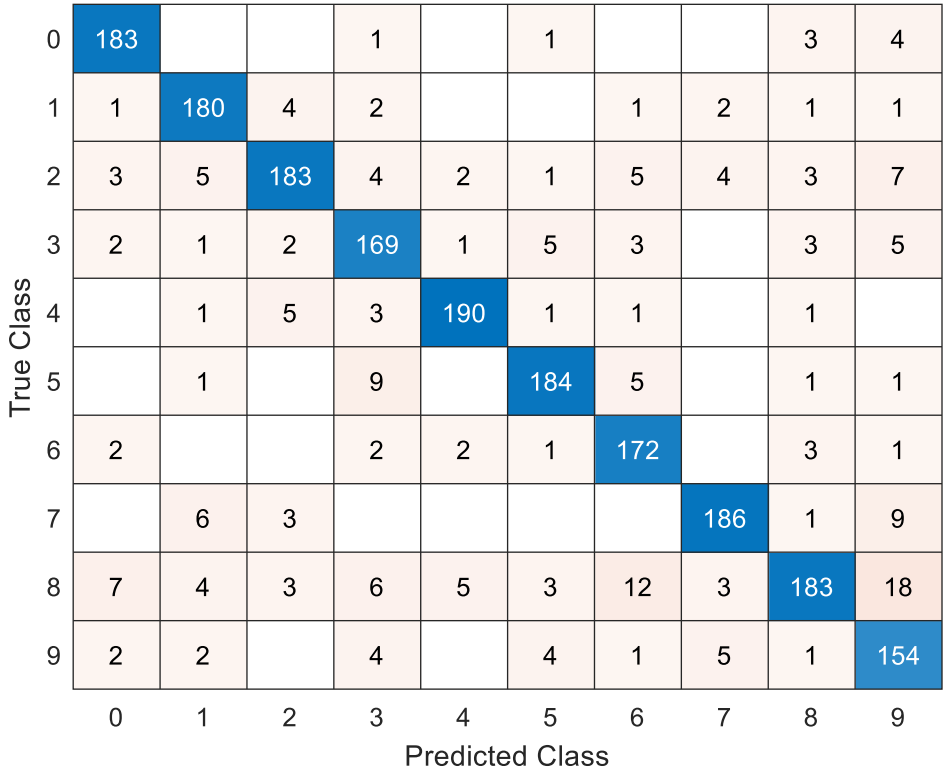
```
[imdsTrain,imdsTest] = splitEachLabel(imds, 0.8, 'randomize');
```

## Uczenie

```
options = trainingOptions("adam", Plots="training-progress");
[net, info] = trainNetwork(imdsTrain, layers, options);
```

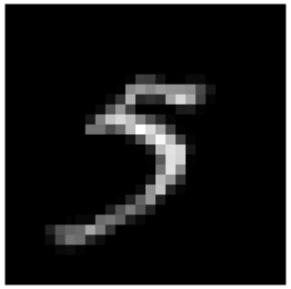
# Deep Learning Toolbox – testowanie sieci

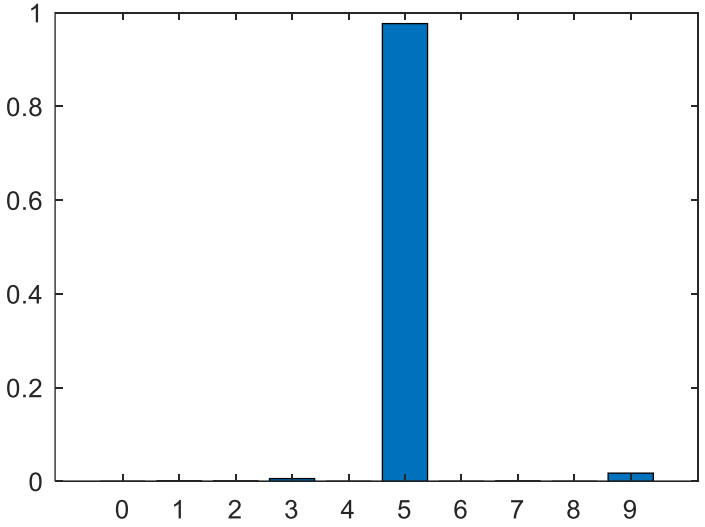
```
YPred = classify(net, imdsTest); % klasyfikacja obrazów testowych
correct = (YPred==imdsTest.Labels); % prawidłowo rozpoznane
accTest = sum(correct)/numel(imdsTest.Labels)  % dokładność
figure; confusionchart(YPred, imdsTest.Labels) % macierz pomyłek
```



# Deep Learning Toolbox – analiza wyników

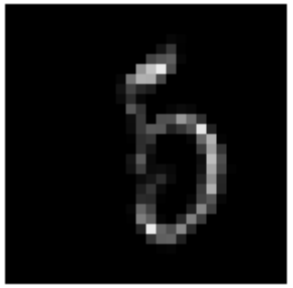
## Obraz rozpoznany prawidłowo

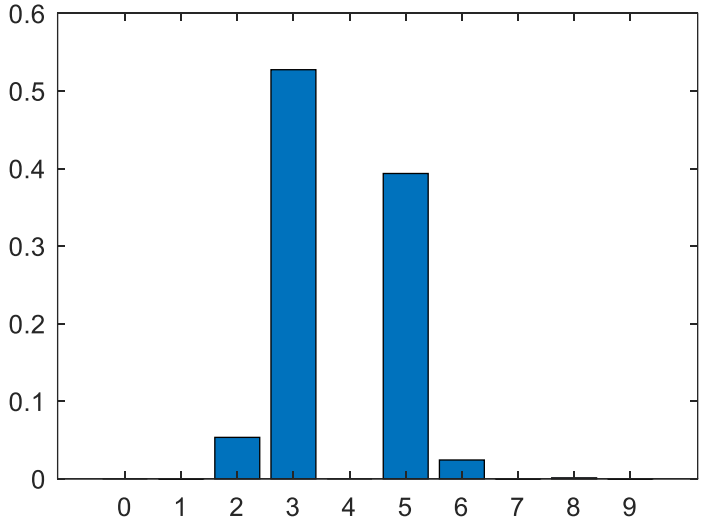





Prawdopodobieństwo identyfikacji
(warstwa softmax)

## Obraz rozpoznany nieprawidłowo






Prawdopodobieństwo identyfikacji
(warstwa softmax)