

---

# Künstliches neuronales Netz für die Bilderkennung

## Table of Contents

Vorbereitung der Bilddaten .....	1
Die <i>Neural Pattern Recognition Application</i> .....	2
Erzeugen eines KNN auf der Kommandozeile .....	12
Überprüfung des KNN .....	15
Speichern und Export .....	16

Dieses Beispiel zeigt den Einsatz des Matlab Werkzeugs zur neuronalen Mustererkennung (`nprtool` - Neural Pattern Recognition Tool) für die Klassifikation von Bildern mit Ziffern. Für diese Aufgabe wird ein künstliches neuronales Netz (KNN) mit einer Eingangs-, einer verborgenen und einer Ausgabeschicht anlegt und trainiert.

Dieses Netzwerk soll anschließend in einer einfachen Automatisierungsaufgabe zum Sortieren von Bauteilen auf einem Förderband verwendet werden.

## Vorbereitung der Bilddaten

Die Bauteile der Sortieraufgabe werden in diesem Beispiel durch die Ziffern 0 bis 9 repräsentiert, die einem ähnlichen Beispiel aus Matlab 2015b entnommen sind. Das Erstellen geeigneter eigener Aufnahmen für das Trainieren eines KNN erfordert besondere Sorgfalt, damit nicht das KNN nicht "falsche" Informationen zur Unterscheidung lernt (z.B. Intensitätsunterschiede oder Helligkeitsverläufe im Hintergrund der Bauteile).

Die synthetischen Bilder durch die Anwendung zufälliger, affiner Transformationen auf digitale Bilder mit Ziffern verschiedener Fonts entstanden. Jedes Bild besteht aus 28x28 Bildpunkten und es gibt jeweils insgesamt 5000 Trainings- und Testbeispiele. Diese Daten müssen zuerst geladen werden und ein paar der Trainingsbilder werden angezeigt.

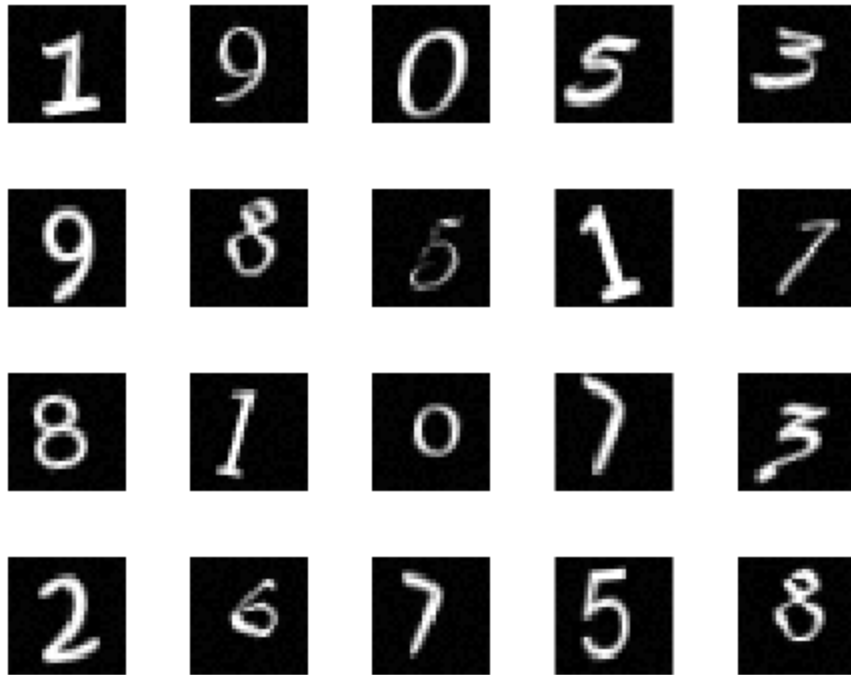
Da die Daten aus einem Beispiel für eine neuere Matlab-Version stammen, müssen diese noch an ein Format überführt werden, welches zum etwas älteren Werkzeug zur Mustererkennung kompatibel ist.

```
% load training and test data
[xTrainImages, tTrain] = digittrain_dataset;
[xTestImages, tTest] = digittest_dataset;

% display some of the training images
clf
for i = 1:20
    subplot(4,5,i);
    imshow(xTrainImages{i});
end

% convert training and testing images to 1D-Array
for i = 1:5000
    vTrainImages(:,i) = reshape(cell2mat(xTrainImages{i}),[784,1]);
end
```

```
vTestImages(:,i) = reshape(cell2mat(xTestImages(i)),[784,1]);  
end  
  
% remove some variables from workspace  
clear i
```



Die Kennzeichnung der Bilder ist in einer 10x5000 Matrix gespeichert. In jeder Spalte steht jeweils in genau einem Element eine 1, die die Zugehörigkeit zur Klasse signalisiert. Alle anderen Elemente der Spalte sind 0. Steht eine 1 im zehnten Element einer Spalte, dann gehört das Bild zur Ziffer 0.

## Die *Neural Pattern Recognition Application*


Die *Neural Pattern Recognition Application* erleichtert das Erstellen künstlicher neuronaler Netze auf Basis vorhandener Daten. Die Auswahl der Daten erfolgt dabei geführt durch die Applikation und kann auf den *Workspace* oder mat-Dateien zugreifen.

Von der Kommandozeile startet man das Programm mit: nprtool

Im folgenden sind die einzelnen Schritte durch das Programm mit Beispieldaten gezeigt. Experimentieren Sie mit den Einstellungen zu \* Validation (Seite Validation and Test Data) \* Testing (Seite Validation and Test Data) \* Number of Hidden Neurons (Seite Network Architecture) und vergleichen Sie die Auswirkungen auf das Konfusionsdiagramm und die Grenzwertoptimierungskurve (Receiver Operating Characteristics - ROC).

# Künstliches neuronales Netz für die Bilderkennung

Neural Pattern Recognition (nprtool)

 **Welcome to the Neural Pattern Recognition app.**  
Solve a pattern-recognition problem with a two-layer feed-forward network.

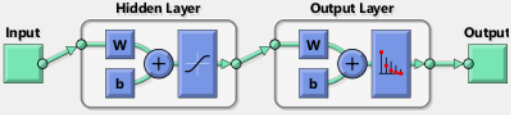
**Introduction**

In pattern recognition problems, you want a neural network to classify inputs into a set of target categories.

For example, recognize the vineyard that a particular bottle of wine came from, based on chemical analysis (*wine\_dataset*); or classify a tumor as benign or malignant, based on uniformity of cell size, clump thickness, mitosis (*cancer\_dataset*).

The Neural Pattern Recognition app will help you select data, create and train a network, and evaluate its performance using cross-entropy and confusion matrices.


**Neural Network**





A two-layer feed-forward network, with sigmoid hidden and softmax output neurons (*patternnet*), can classify vectors arbitrarily well, given enough neurons in its hidden layer.


The network will be trained with scaled conjugate gradient backpropagation (*trainscg*).


➡ To continue, click [Next].

 Neural Network Start

 Welcome

 Back

 Next

 Cancel

## Künstliches neuronales Netz für die Bilderkennung

Neural Pattern Recognition (nprtool)

### Select Data

What inputs and targets define your pattern recognition problem?

Get Data from Workspace

Input data to present to the network.

Inputs:  ...

Target data defining desired network output.

Targets:  ...

Samples are: ☒ Matrix columns ☐ Matrix rows

Want to try out this tool with an example data set?

#### Summary

Inputs 'vTrainImages' is a 784x5000 matrix, representing static data: 5000 samples of 784 elements.

Targets 'tTrain' is a 10x5000 matrix, representing static data: 5000 samples of 10 elements.

Accessing workspace.

## Künstliches neuronales Netz für die Bilderkennung

Neural Pattern Recognition (nprtool)

### Validation and Test Data

Set aside some samples for validation and testing.

Select Percentages

Randomly divide up the 5000 samples:

Training:	80%	4000 samples
Validation:	15% ▾	750 samples
Testing:	5% ▾	250 samples

Restore Defaults

Explanation

Three Kinds of Samples:

**Training:**  
These are presented to the network during training, and the network is adjusted according to its error.

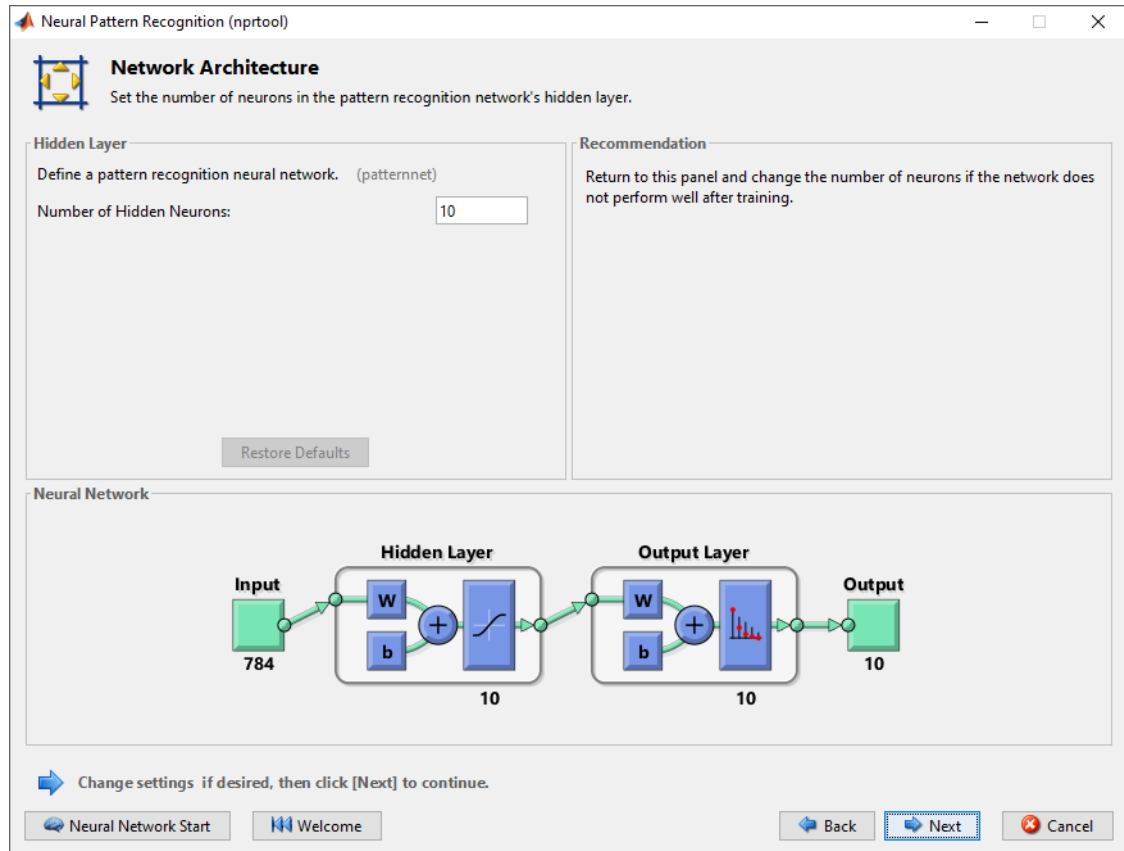
**Validation:**  
These are used to measure network generalization, and to halt training when generalization stops improving.

**Testing:**  
These have no effect on training and so provide an independent measure of network performance during and after training.

➡ Change percentages if desired, then click [Next] to continue.

Neural Network Start Welcome Back Next Cancel

# Künstliches neuronales Netz für die Bilderkennung



## Künstliches neuronales Netz für die Bilderkennung

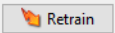
Neural Pattern Recognition (nprtool)

### Train Network

Train the network to classify the inputs according to the targets.







Train Network

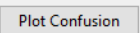
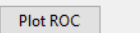
Train using scaled conjugate gradient backpropagation. (trainscg)






Training automatically stops when generalization stops improving, as indicated by an increase in the cross-entropy error of the validation samples.


#### Results

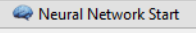

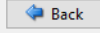


	 Samples	 CE	 %E
 Training:	4000	1.40299e-0	11.50000e-0
 Validation:	750	5.92265e-0	21.06666e-0
 Testing:	250	6.61452e-0	24.39999e-0

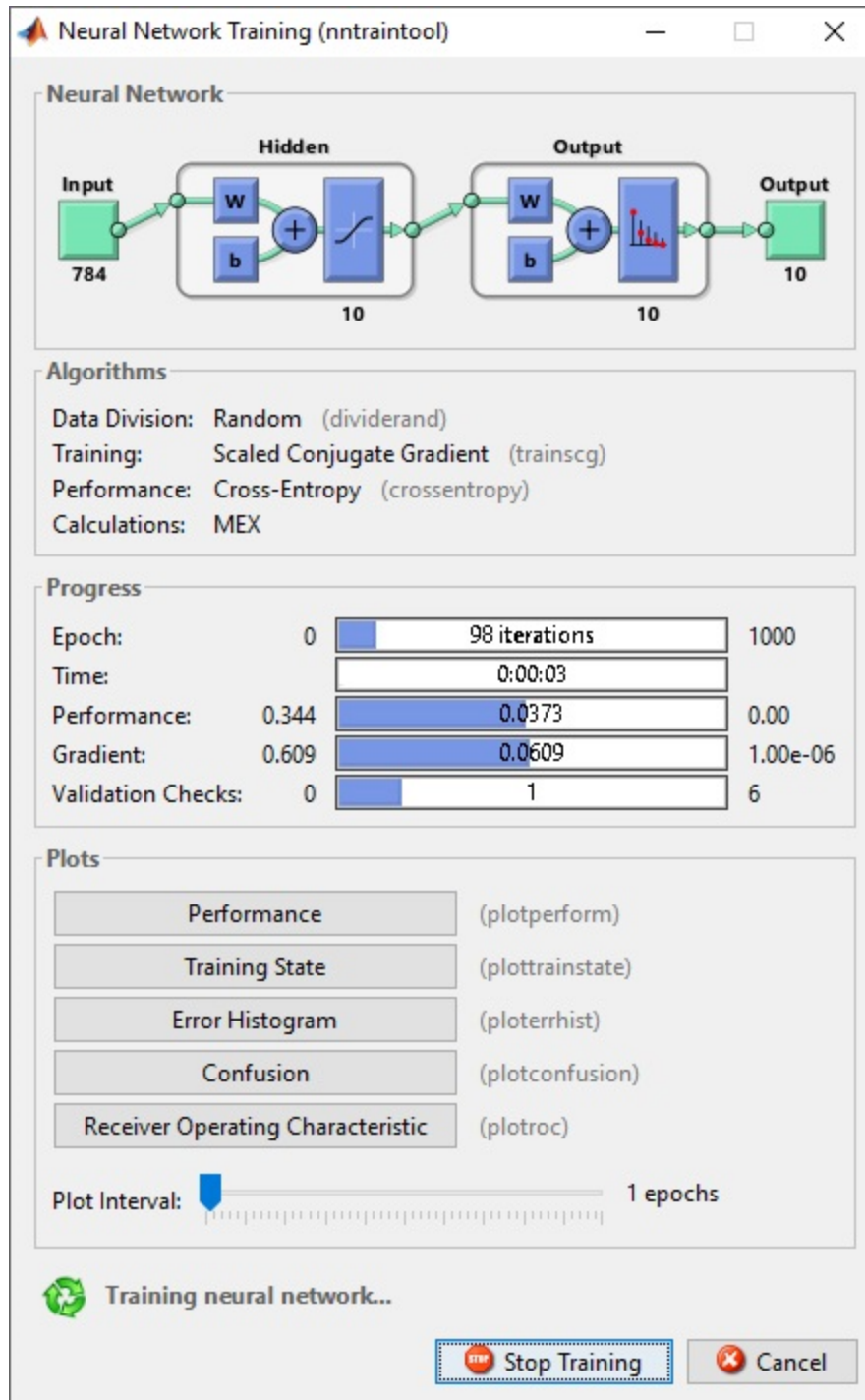
 

#### Notes

-  Training multiple times will generate different results due to different initial conditions and sampling.
-  Minimizing Cross-Entropy results in good classification. Lower values are better. Zero means no error.
-  Percent Error indicates the fraction of samples which are misclassified. A value of 0 means no misclassifications, 100 indicates maximum misclassifications.

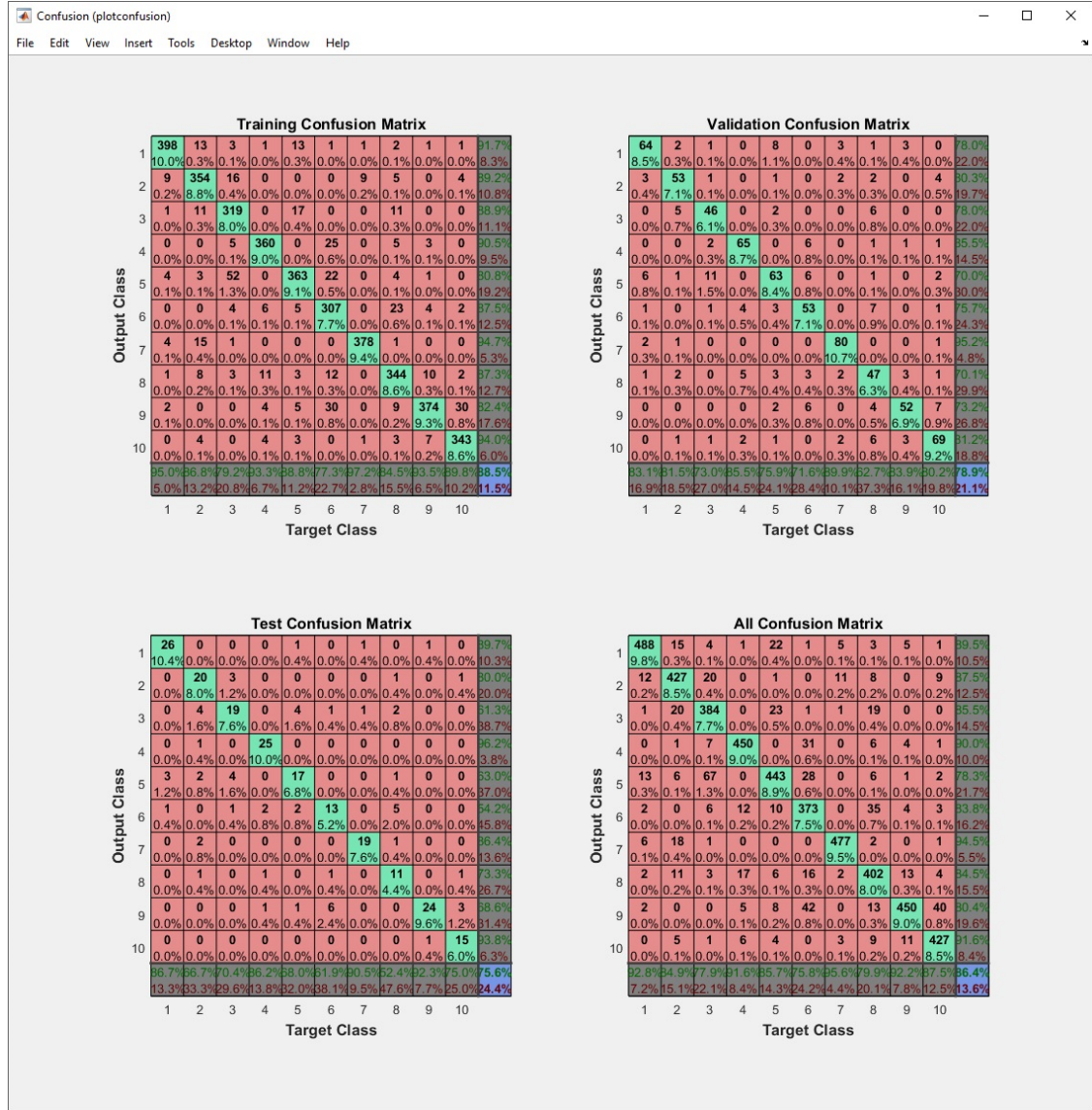
 Open a plot, retrain, or click [Next] to continue.

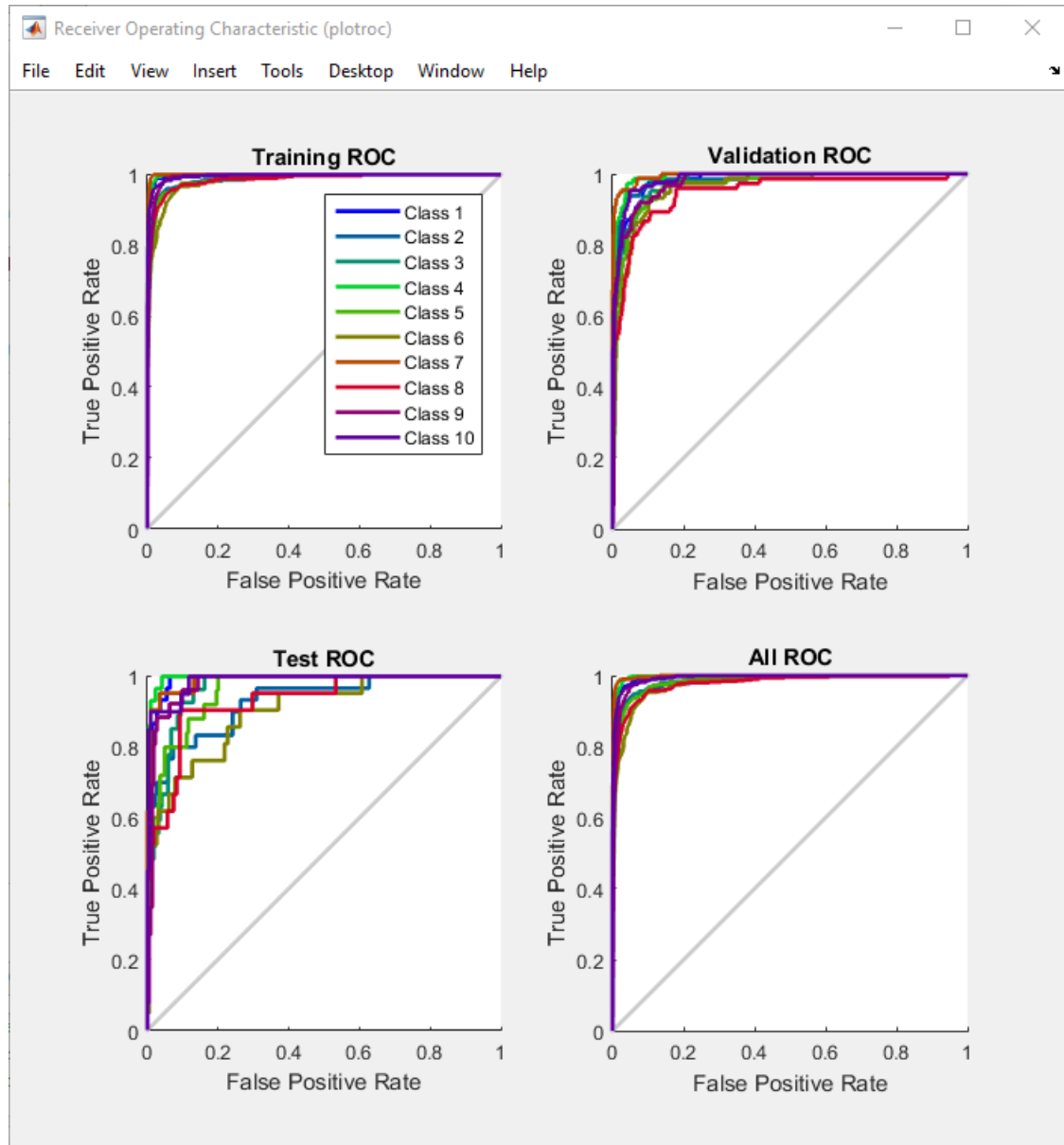
    



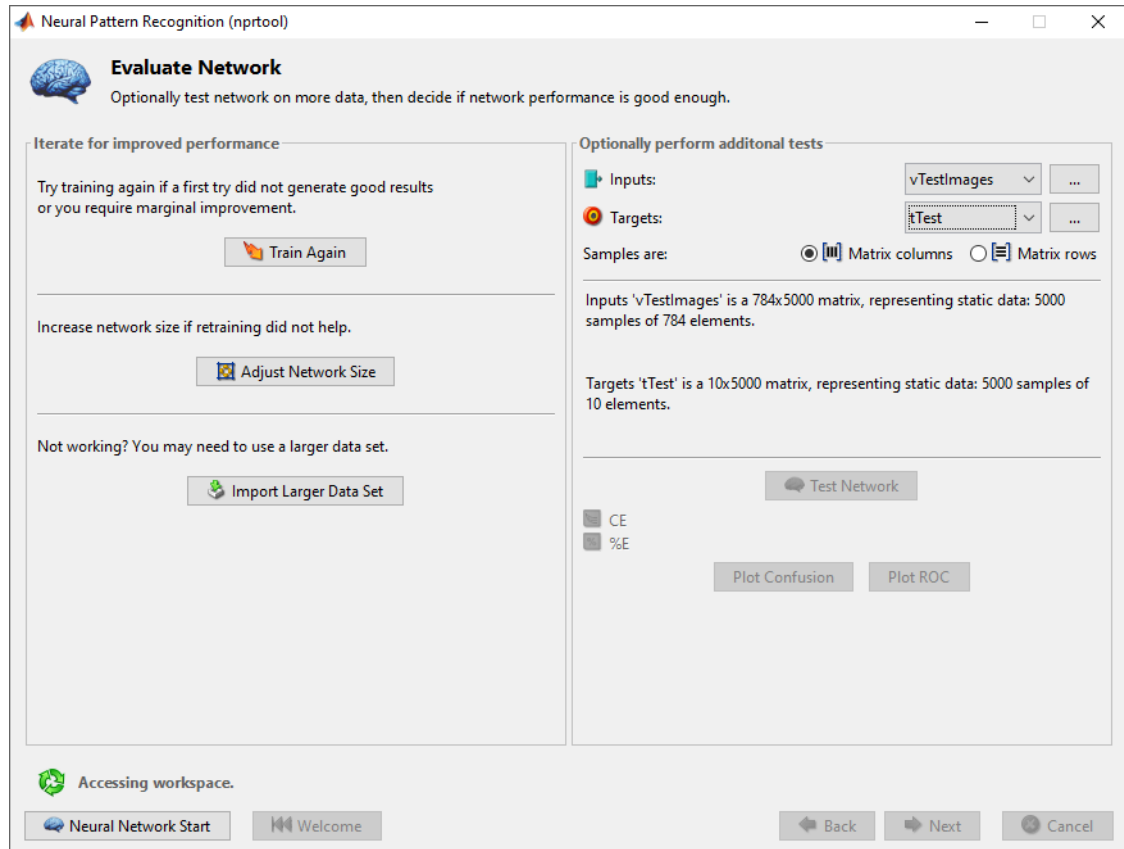


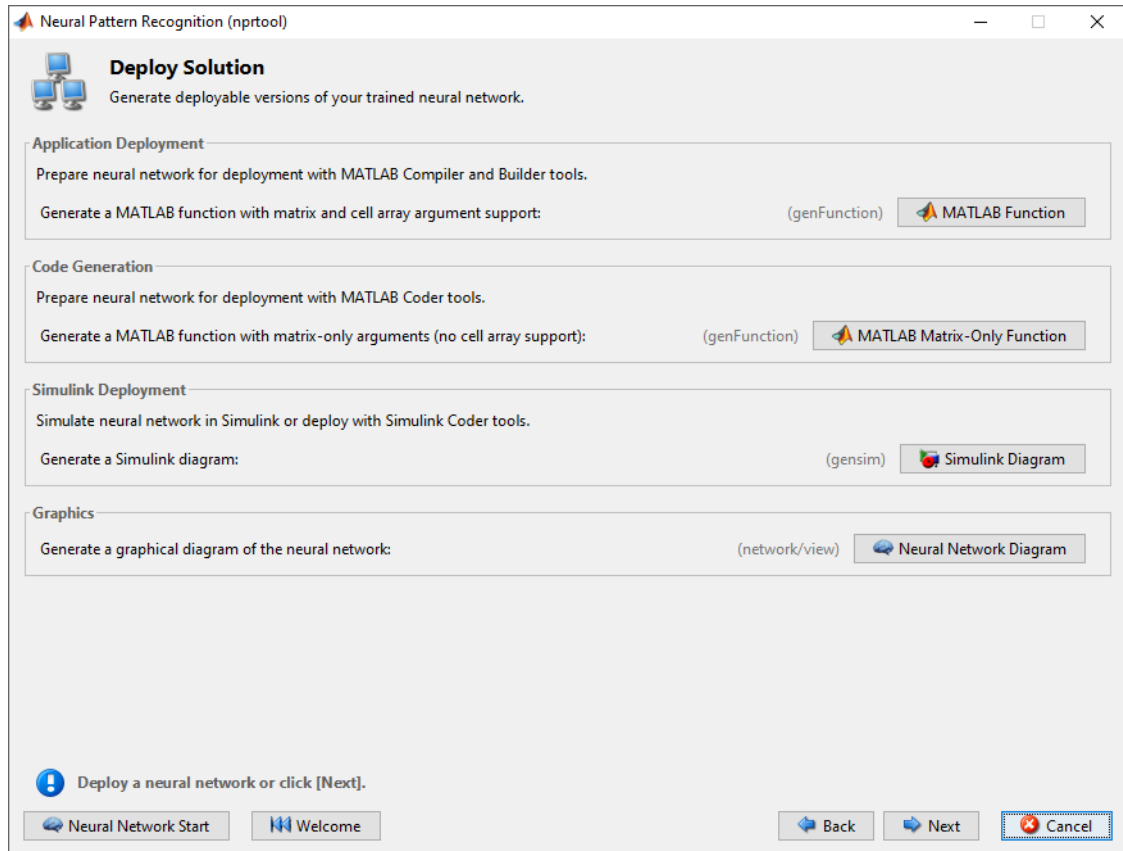
# Künstliches neuronales Netz für die Bilderkennung





## Künstliches neuronales Netz für die Bilderkennung





## Erzeugen eines KNN auf der Kommandozeile

Neben der *Neural Pattern Recognition Application* können künstliche neuronale Netze auch von der Kommandozeile oder in Matlab-Skripten erzeugt werden. Hier stehen wesentlich mehr und flexiblere Möglichkeiten zur Verfügung. Hier soll nur gezeigt werden, wie die gleiche Aufgabe in textueller Form gelöst werden kann.

```
% define neural network
net = patternnet(10)

% train neural network with training data
net = train(net,vTrainImages,tTrain);

% display resulting network
view(net)

% test network and plot confusion matrix
y = net(vTrainImages);
plotconfusion(tTrain,y,'Training Data ');

net =

    Neural Network
```

```
        name: 'Pattern Recognition Neural Network'
        userdata: (your custom info)

dimensions:

    numInputs: 1
    numLayers: 2
    numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
    numFeedbackDelays: 0
    numWeightElements: 10
    sampleTime: 1

connections:

    biasConnect: [1; 1]
    inputConnect: [1; 0]
    layerConnect: [0 0; 1 0]
    outputConnect: [0 1]

subobjects:

    input: Equivalent to inputs{1}
    output: Equivalent to outputs{2}

    inputs: {1x1 cell array of 1 input}
    layers: {2x1 cell array of 2 layers}
    outputs: {1x2 cell array of 1 output}
    biases: {2x1 cell array of 2 biases}
    inputWeights: {2x1 cell array of 1 weight}
    layerWeights: {2x2 cell array of 1 weight}

functions:

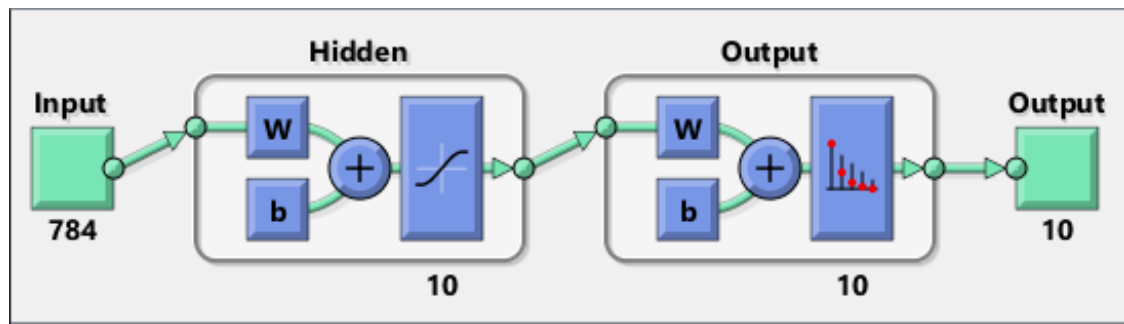
    adaptFcn: 'adaptwb'
    adaptParam: (none)
    derivFcn: 'defaultderiv'
    divideFcn: 'dividerand'
    divideParam: .trainRatio, .valRatio, .testRatio
    divideMode: 'sample'
    initFcn: 'initlay'
    performFcn: 'crossentropy'
    performParam: .regularization, .normalization
    plotFcns: {'plotperform', plottrainstate, ploterrhist,
               plotconfusion, plotroc}
    plotParams: {1x5 cell array of 5 params}
    trainFcn: 'trainscg'
    trainParam: .showWindow, .showCommandLine, .show, .epochs,
               .time, .goal, .min_grad, .max_fail, .sigma,
               .lambda

weight and bias values:
```

*IW: {2x1 cell} containing 1 input weight matrix*  
*LW: {2x2 cell} containing 1 layer weight matrix*  
*b: {2x1 cell} containing 2 bias vectors*

*methods:*

*adapt: Learn while in continuous use*  
*configure: Configure inputs & outputs*  
*gensim: Generate Simulink model*  
*init: Initialize weights & biases*  
*perform: Calculate performance*  
*sim: Evaluate network outputs given inputs*  
*train: Train network with examples*  
*view: View diagram*  
*unconfigure: Unconfigure inputs & outputs*



**Training Data Confusion Matrix**

1	504 10.1%	4 0.1%	5 0.1%	1 0.0%	0 0.0%	2 0.0%	1 0.0%	0 0.0%	1 0.0%	0 0.0%	97.3% 2.7%
2	7 0.1%	475 9.5%	2 0.0%	0 0.0%	1 0.0%	0 0.0%	3 0.1%	4 0.1%	0 0.0%	2 0.0%	96.2% 3.8%
3	5 0.1%	5 0.1%	455 9.1%	1 0.0%	17 0.3%	0 0.0%	0 0.0%	8 0.2%	2 0.0%	0 0.0%	92.3% 7.7%
4	1 0.0%	4 0.1%	6 0.1%	477 9.5%	0 0.0%	4 0.1%	0 0.0%	3 0.1%	0 0.0%	0 0.0%	96.4% 3.6%
5	2 0.0%	1 0.0%	17 0.3%	2 0.0%	492 9.8%	2 0.0%	1 0.0%	11 0.2%	6 0.1%	1 0.0%	92.0% 8.0%
6	2 0.0%	1 0.0%	0 0.0%	2 0.0%	3 0.1%	465 9.3%	0 0.0%	2 0.0%	0 0.0%	3 0.1%	97.3% 2.7%
7	4 0.1%	6 0.1%	3 0.1%	0 0.0%	0 0.0%	0 0.0%	487 9.7%	0 0.0%	3 0.1%	0 0.0%	96.8% 3.2%
8	1 0.0%	5 0.1%	5 0.1%	6 0.1%	0 0.0%	4 0.1%	5 0.1%	469 9.4%	3 0.1%	0 0.0%	94.2% 5.8%
9	0 0.0%	2 0.0%	0 0.0%	2 0.0%	2 0.0%	1 0.0%	2 0.0%	4 0.1%	464 9.3%	5 0.1%	96.3% 3.7%
10	0 0.0%	0 0.0%	0 0.0%	0 0.0%	2 0.0%	14 0.3%	0 0.0%	2 0.0%	9 0.2%	477 9.5%	94.6% 5.4%
	95.8% 4.2%	94.4% 5.6%	92.3% 7.7%	97.1% 2.9%	95.2% 4.8%	94.5% 5.5%	97.6% 2.4%	93.2% 6.8%	95.1% 4.9%	97.7% 2.3%	95.3% 4.7%
	1	2	3	4	5	6	7	8	9	10	
	Target Class										

## Überprüfung des KNN

Um die Funktionsfähigkeit des Netzes zu überprüfen, steht ein eigener Datensatz von Bildern zur Verfügung.

Führen Sie den Test durch und vergleichen Sie das Ergebnis mit der Konfusionsmatrix aus dem Trainingslauf.

```
% test network and plot confusion matrix
y = net(vTestImages);
plotconfusion(tTest,y,'Testing Data');
```

**Testing Data Confusion Matrix**

1	440 8.8%	0 0.0%	12 0.2%	2 0.0%	3 0.1%	7 0.1%	6 0.1%	1 0.0%	3 0.1%	0 0.0%	92.8% 7.2%
2	34 0.7%	485 9.7%	49 1.0%	0 0.0%	12 0.2%	7 0.1%	90 1.8%	26 0.5%	3 0.1%	7 0.1%	68.0% 32.0%
3	10 0.2%	1 0.0%	365 7.3%	2 0.0%	9 0.2%	4 0.1%	0 0.0%	26 0.5%	1 0.0%	0 0.0%	87.3% 12.7%
4	2 0.0%	4 0.1%	4 0.1%	465 9.3%	1 0.0%	10 0.2%	0 0.0%	0 0.0%	0 0.0%	1 0.0%	95.5% 4.5%
5	1 0.0%	2 0.0%	48 1.0%	2 0.0%	471 9.4%	13 0.3%	0 0.0%	23 0.5%	5 0.1%	1 0.0%	83.2% 16.8%
6	0 0.0%	0 0.0%	0 0.0%	13 0.3%	0 0.0%	436 8.7%	0 0.0%	1 0.0%	0 0.0%	4 0.1%	96.0% 4.0%
7	1 0.0%	1 0.0%	1 0.0%	0 0.0%	0 0.0%	0 0.0%	396 7.9%	0 0.0%	2 0.0%	0 0.0%	98.8% 1.2%
8	0 0.0%	6 0.1%	20 0.4%	5 0.1%	0 0.0%	1 0.0%	1 0.0%	418 8.4%	2 0.0%	3 0.1%	91.7% 8.3%
9	6 0.1%	1 0.0%	1 0.0%	11 0.2%	4 0.1%	0 0.0%	7 0.1%	1 0.0%	472 9.4%	29 0.6%	88.7% 11.3%
10	6 0.1%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	22 0.4%	0 0.0%	4 0.1%	12 0.2%	455 9.1%	91.2% 8.8%
	88.0% 12.0%	97.0% 3.0%	73.0% 27.0%	93.0% 7.0%	94.2% 5.8%	87.2% 12.8%	79.2% 20.8%	83.6% 16.4%	94.4% 5.6%	91.0% 9.0%	88.1% 11.9%
	1	2	3	4	5	6	7	8	9	10	
	Target Class										

## Speichern und Export

Das erzeugte künstliche neuronale Netz wird für den zweiten Teil der Automatisierungsaufgabe benötigt. Speichern Sie deshalb das Netzwerk auf einem USB-Stick. Die Befehle zum Speichern und Einlesen des Netzes sowie das Erzeugen des Simulink-Modells sehen wie folgt aus.

```
% save neural net from workspace to file
save('net.mat', 'net');
```

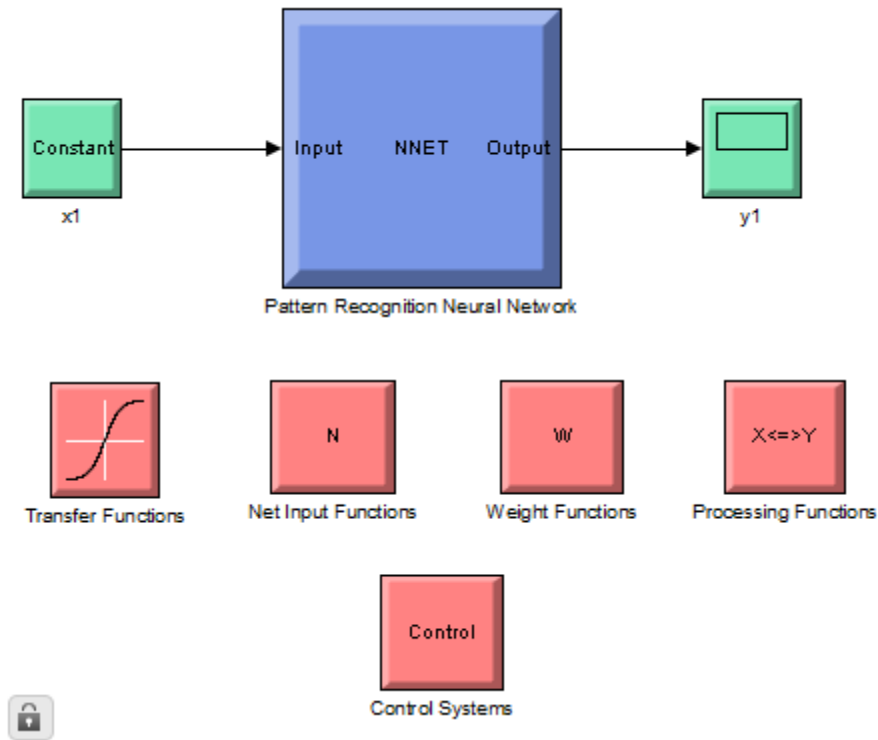
```
% loading the neural net into workspace
load('net.mat');
```

```
% generate Simulink block for neural network simulation
gensim(net);
```

*Warning: The model name 'untitled' is shadowing another name in the MATLAB*



*workspace or path. Type "which -all untitled" at the command line to find the other uses of this name. You should change the name of the model to avoid problems.*



*Published with MATLAB® R2015a*