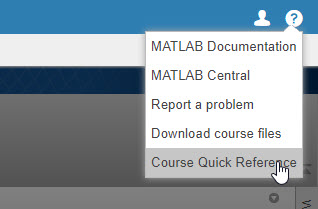
**1. Classifying Images with Convolutional Networks**

**Summary - Transfer Learning Functions**

These functions will be used throughout this course. Click on any function to visit the documentation.  
  
View this information in the course quick reference guide any time you need a reference.



**Create a network**

| **Function** | **Description** |
| --- | --- |
| [alexnet](http://www.mathworks.com/help/nnet/ref/alexnet.html) | Load pretrained network “AlexNet” |
| [supported networks](https://www.mathworks.com/solutions/deep-learning/models.html) | View list of available pretrained networks |
| [fullyConnectedLayer](https://www.mathworks.com/help/nnet/ref/nnet.cnn.layer.fullyconnectedlayer.html) | Create new fully connected network layer |
| [classificationLayer](http://www.mathworks.com/help/nnet/ref/classificationlayer.html) | Create new output layer for a classification network |

**Get training images**

| **Function** | **Description** |
| --- | --- |
| [imageDatastore](http://www.mathworks.com/help/matlab/ref/matlab.io.datastore.imagedatastore.html) | Create datastore reference to image files |
| [augmentedImageDatastore](https://www.mathworks.com/help/deeplearning/ref/augmentedimagedatastore.html) | Preprocess images in a datastore |
| [splitEachLabel](http://www.mathworks.com/help/matlab/ref/datastore.spliteachlabel.html) | Divide datastore into multiple datastores |
| [imresize](https://www.mathworks.com/help/matlab/ref/imresize.html) | Resize image to expected size for a pretrained network |

**Set training algorithm options**

| **Function** | **Description** |
| --- | --- |
| [trainingOptions](http://www.mathworks.com/help/nnet/ref/trainingoptions.html) | Create variable containing training algorithm options |

**Perform training**

| **Function** | **Description** |
| --- | --- |
| [trainNetwork](http://www.mathworks.com/help/nnet/ref/trainnetwork.html) | Perform training |

**Use trained network to perform classifications**

| **Function** | **Description** |
| --- | --- |
| [classify](https://www.mathworks.com/help/nnet/ref/classify.html) | Use trained network to classify input images |

**Evaluate trained network**

| **Function** | **Description** |
| --- | --- |
| [nnz](http://www.mathworks.com/help/matlab/ref/nnz.html) | Count non-zero elements in an array |
| [confusionchart](https://www.mathworks.com/help/nnet/ref/confusionchart.html) | Display confusion matrix |

**2. Interpreting Network Behavior**

**Summary - Interpreting Network Behavior**

**Extracting Activations**

>>

features

= activations(

net

,

img

,

layer

,

'OutputAs','rows'

)

|  |  |
| --- | --- |
| features | Activations from the specified network layer. The size of the features matrix depends on the 'OutputAs' setting. |
| **Outputs** | |

|  |  |
| --- | --- |
| net | A trained network that you want to investigate. |
| img | An image that was imported with imread. |
| layer | The layer to extract features from. This can be a name like 'conv1'. You can find the layer names using net.Layers. |
| 'OutputAs','rows' | The default value of the 'OutputAs' argument is 'channels'. Use the default setting when you want to display activations. If you want to use the activations to train a machine learning model, use the setting 'rows'. |
| **Inputs** | |

**Displaying Activations**

Displaying activations helps you investigate the behavior of your network at different points of the learned feature extraction.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Import a sample image and a network. | |  | | --- | |  | |  | |  | | im = imread('testim.jpg');  net = alexnet; |
| Extract the activations from the fifth convolution layer. | |  | | --- | |  | |  | |  | | features = activations(...  net,im,'conv5'); |
| Display a montage of the activations for all channels. Often there are too many channels to effectively view the features in a montage, and its more practical to view each channel separately. | |  | | --- | |  | |  | |  | | montage(rescale(features))  C:\Users\user1\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\F113076.tmp |
| Display the activations for the 90th channel. This channel activates negatively on a pair of glasses. | |  | | --- | |  | |  | |  | | actvn = features(:,:,90);  imshow(rescale(actvn))  C:\Users\user1\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\53ACCBF4.tmp |

**3. Creating Networks**

**Summary - Creating Networks**

Deep network architectures are created by appending layers together.

Refer to the documentation for details on the available deep network layers. The architectures used in documentation examples can also be used as a starting place for creating your networks.

layers = [

imageInputLayer([28 28 1])

convolution2dLayer(3,16,'Padding',1)

batchNormalizationLayer

reluLayer

maxPooling2dLayer(2,'Stride',2)

fullyConnectedLayer(10)

softmaxLayer

classificationLayer]

|  |  |  |  |
| --- | --- | --- | --- |
|  | [List of Deep Learning Layers](https://www.mathworks.com/help/deeplearning/ug/list-of-deep-learning-layers.html) - Layer creation functions |  | [Create Simple Deep Learning Network for Classification](https://www.mathworks.com/help/deeplearning/ug/create-simple-deep-learning-network-for-classification.html?newdoc) - Create an architecture to classify handwritten digits |
|  | [Specify Layers of Convolutional Neural Network](https://www.mathworks.com/help/deeplearning/ug/layers-of-a-convolutional-neural-network.html) - The details of ConvNet layers |  |  |

**4. Training Networks**

**Summary - Training Networks**

**Setting training options**

The trainingOptions function can be used to control the training of a deep learning neural network.  
  
When possible, plot the training progress plot and use validation data. These settings allow you to get a more complete picture of network training.

>>

options

= trainingOptions(

'adam'

,

'Plots','training-progress'

,

'ValidationData',val

)

|  |  |
| --- | --- |
| options | Options to be used as an input to the trainNetwork function. |
| **Outputs** | |

|  |  |
| --- | --- |
| 'adam' | Name of a solver for network training. |
| 'Plots','training-progress' | Plot training progress. The plot shows mini-batch loss and accuracy. |
| 'ValidationData',val | Data to use for validation during training. The validation loss and accuracy will appear on the training progress plot. |
| **Inputs** | |

**Training the network**

Once you have prepared your training data, architecture, and options, you can pass these inputs to the trainNetwork function.

>>

mynet

= trainNetwork(

trainingData

,

layers

,

options

)

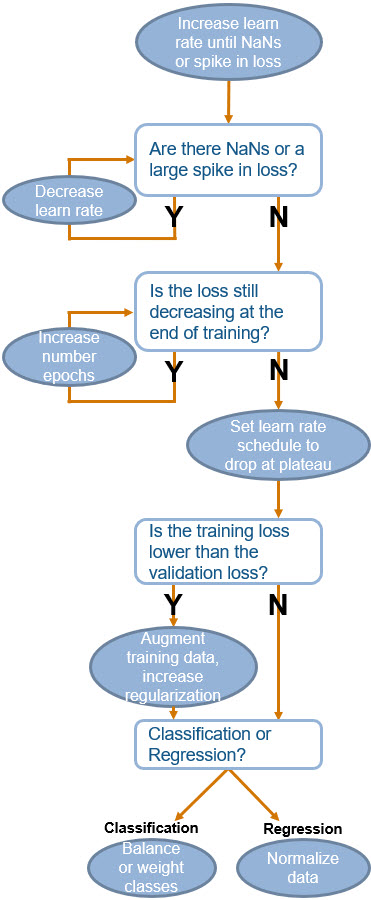
|  |  |
| --- | --- |
| mynet | Trained network. You can now use this network with the predict function. |
| **Outputs** | |

|  |  |
| --- | --- |
| trainingData | Training images with labels. |
| layers | Network layers. |
| options | Training options created with the trainingOptions function. |
| **Inputs** | |

**5. Improving Performance**

**Summary - Improving Performance**

**Training options**



**Troubleshooting process**

This chart to the left shows some general guidelines for setting training options when training a convolutional neural network.  
  
Check out the [Deep Learning Tips and Tricks](https://www.mathworks.com/help/deeplearning/ug/deep-learning-tips-and-tricks.html) page in the documentation for more general advice on improving the accuracy of your deep network.

**Directed acyclic graph networks**

To perform transfer learning from a DAG network, you should

1. Import a pretrained DAG network, like ResNet-50 or GoogLeNet.
2. Extract the layer graph from the network architecture using layerGraph.
3. Find and name the layers that you need to replace.
4. Create the new layers to be added and name them.
5. Connect the new layers using replaceLayers.

**6. Project**

**7. Performing Regression**

**Summary - Performing Regression**

**Training a regression network**

Regression refers to assigning continuous response values to data, instead of discrete classes. Like with classification, you can train a regression network with the trainNetwork function.  
  
If you are performing transfer learning from a pretrained network, you should remove the softmax layer.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Import a pretrained network and save the layers. | |  | | --- | |  | |  | |  | | net = alexnet;  ly = net.Layers; |
| Remove the last three layers from the architecture. | |  | | --- | |  | |  | |  | | ly(end-2:end) = []; |
| Create new fully connected layer and regression output layer. Append the new layers to your architecture. | |  | | --- | |  | |  | |  | | newlayers = [fullyConnectedLayer(1);regressionLayer()];  ly = [ly;newlayers]; |

**Evaluating a regression network**

Instead of calculating accuracy or misclassications, you can calculate the root-mean-square error (RMSE).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| To find the error for one image, you can just find the difference between the known value and the predicted value. | |  | | --- | |  | |  | |  | | pred = predict(mynet,testImage);  err = trueValue - pred; |
| To find the error for multiple images, calculate RMSE. | |  | | --- | |  | |  | |  | | rmse = sqrt(sum(err.^2)); |

**8. Detecting Objects in Images**

**Summary - Detecting Objects in Images**

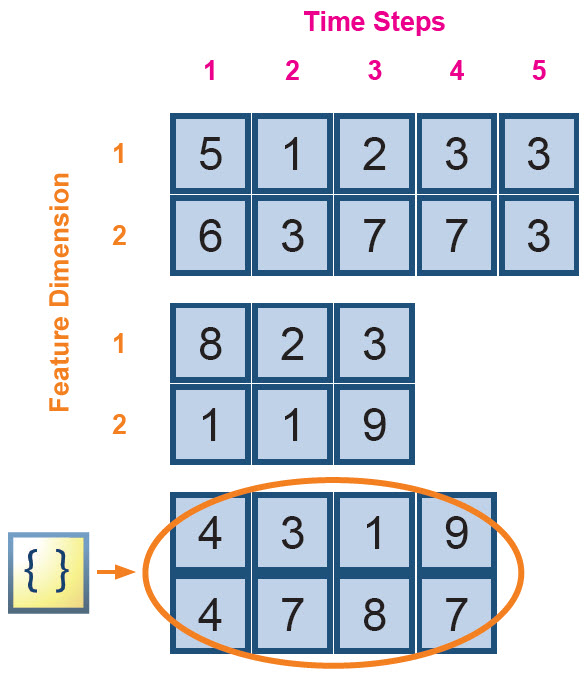
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Train an R-CNN, Fast R-CNN, or Faster R-CNN using functionality in the Computer Vision System Toolbox. | |  | | --- | |  | |  | |  | | bikeDetector = trainRCNNObjectDetector(data,net,options);  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Training a R-CNN Object Detector for the following object classes:  \* Bike  --> Extracting region proposals from 359 training images...done.  Training on **single** GPU.  |=======================================================================================================|  | Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base Learning |  | | | (hh:mm:ss) | Loss | Accuracy | RMSE | Rate |  |=======================================================================================================|  | 1 | 1 | 00:00:00 | 0.8941 | 54.69 | 0.94 | 1.0000e-04 |  | 1 | 50 | 00:00:07 | 0.3544 | 93.75 | 0.99 | 1.0000e-04 |  | 1 | 100 | 00:00:15 | 0.2978 | 96.09 | 0.83 | 1.0000e-04 |  | 1 | 150 | 00:00:23 | 0.2498 | 94.53 | 0.97 | 1.0000e-04 |  | 1 | 200 | 00:00:32 | 0.8540 | 73.44 | 1.05 | 1.0000e-04 |  |=======================================================================================================|  Detector training complete.  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| The detect function can return the object bounding boxes, the corresponding confidence score, and the corresponding label. | |  | | --- | |  | |  | |  | | [bikebox,score,label] = detect(bikeDetector,myimage)  bikebox = 1×4  68 162 194 415  score =  0.5680  label =  Bike |
| Add the bounding box to the image using the insertObjectAnnotation function. | |  | | --- | |  | |  | |  | | imshow(insertObjectAnnotation(myimage,'rectangle',bikebox,label))  C:\Users\user1\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\11D77E80.tmp |

**9. Classifying Sequence Data with Recurrent Networks**

**Summary - Classifying Sequence Data with Recurrent Networks**

**Structuring sequence data**

Training an LSTM requires the data to be stored in a particular format.  
  
The input data is a cell array with one column.  
  
The columns in each sample are the time steps. Every sample can have a different number of time steps, but be cautious because sequences with different lengths will be padded. Significant padding can negatively impact training.  
  
The rows correspond to the feature dimension of the sample.  
  
This could be signal data from different sensors, or different letters in a vocabulary. All samples must have the same number of rows.



**Creating an architecture**

You can create a long short-term memory architecture by appending layers together.

The primary layers for sequence classification are shown to the right.

layers = [

sequenceInputLayer(1)

lstmLayer(256,'OutputMode','last')

fullyConnectedLayer(10)

softmaxLayer()

classificationLayer()]

**Training an LSTM**

You can train a long short-term memory network with the trainNetwork function. The inputs are your data, architecture, and options.

>>

net

= trainNetwork(

C

,

Y

,

layers

,

options

)

|  |  |
| --- | --- |
| net | Long short-term memory network. |
| **Outputs** | |

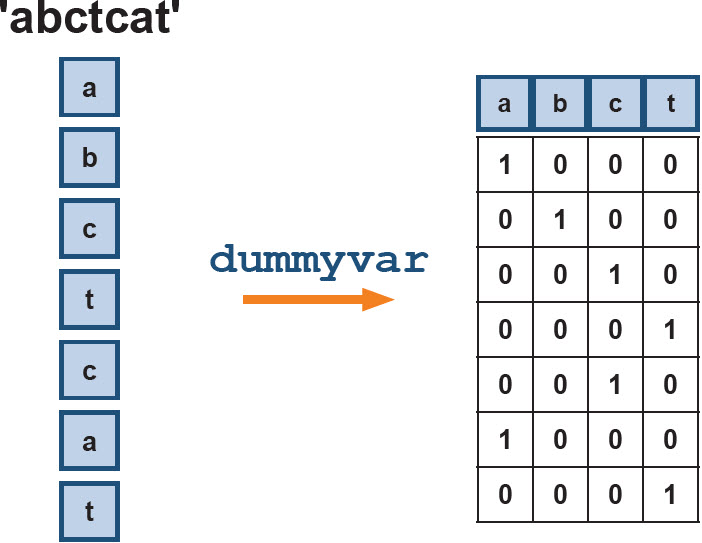
|  |  |
| --- | --- |
| C | Cell array of matrices. Each matrix is one sample, where the columns are the time steps and the rows are the features. |
| Y | Corresponding label for each element of C. |
| layers | Array of network layers. One of these layers should be an lstmLayer or bilstmLayer. |
| options | Training options created with the trainingOptions function. |
| **Inputs** | |

**10. Classifying Categorical Sequences**

**Summary - Classifying Categorical Sequences**

**Dummy variables**

If you want to classify categorical data with LSTMs, you can create dummy predictors for the categories using the dummyvar function.



**Numeric representation of text**

This course uses text data. There are multiple ways that you can convert text to a numeric format. The course example uses the dummifyText function, where vocab is a list of the characters in your text.

dummifyText.mlx

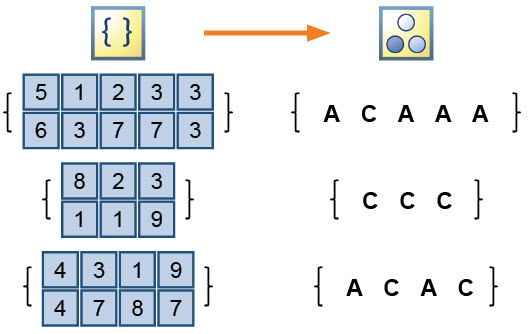
1. function m = dummifyText(t,vocab)
2. t = **uint8**(t);
3. t = categorical(t,vocab)';
4. m = dummyvar(t)';
5. end

**11. Generating Sequences of Output**

**Summary - Generating Sequences of Output**

**Sequence-to-sequence classification**

Sequence-to-sequence classification involves predicting one label for every time step in an input sequence.  
  
When you create your LSTM or BiLSTM layer, set the 'OutputMode' to 'sequence'.



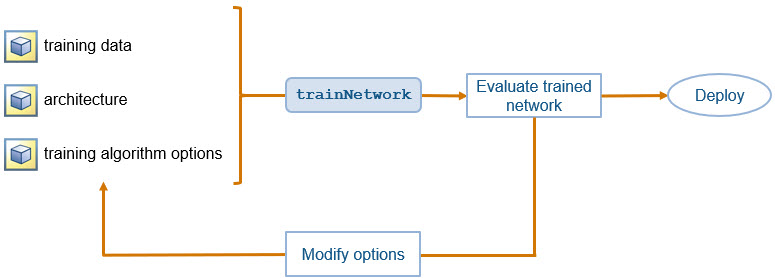
**Sequence forecasting**

Long short-term memory networks can be used to forecast future time steps of a sequence. Forecasting is often performed with time-series data.  
  
The output for every element in the sequence is a prediction for the next value in the sequence.  
  


**12. Project**

**13. Conclusion**

**Summary - Deep Learning Workflow**



To train a deep network, you need to create three components:

1. Image or sequence data with known labels to be used as training data.
2. An array of layers representing the network architecture.
3. Options that control the behavior of the training algorithm.

These three components are provided as the inputs to the trainNetwork function which returns the trained network as output.  
  
You should test the performance of the newly trained network. If it is not adequate, typically you should try adjusting some of the training options and retraining.

**Summary - Deep Learning Tasks**

The table below shows a summary of the different tasks you performed with deep learning in this course.

* **Architecture**: A deep network is represented in MATLAB as an array of layers. Architectures are distinguished by the layers in the network. For example, CNNs contain convolutional layers and LSTMs contain LSTM layers.

* **Training Data Format**: For some tasks, there are multiple ways your can organize your training data. This table shows the format you used in this course.

* **Network output**: Data type when this network is used to predict. This output type generally depends on the output layer used in the network architecture.

* **Course Example**: The data set used to accomplish this task in the course.

* **Documentation Example**: Link to relevant documentation example. You can see a list of all examples [here](https://www.mathworks.com/help/nnet/examples.html).

| **Task** | **Architecture** | **Training Data Format** | **Network Output** | **Course Example** | **Documentation Example** |
| --- | --- | --- | --- | --- | --- |
| Image classification | CNN | Image datastore | Categorical | Classify pets | [Classify digits](https://www.mathworks.com/help/deeplearning/ug/create-simple-deep-learning-network-for-classification.html?newdoc) |
| Image regression | CNN | Table | Numeric | Color corrector | [Angles of rotated digits](https://www.mathworks.com/help/deeplearning/ug/train-a-convolutional-neural-network-for-regression.html?newdoc) |
| Object detection | R-CNN | Table | Bounding box | Detect pet faces | [Stop sign detector](https://www.mathworks.com/help/vision/examples/object-detection-using-deep-learning.html) |
| Sequence to label | LSTM | Cell array | Categorical | Classify author | [Classify vowels](https://www.mathworks.com/help/deeplearning/ug/classify-sequence-data-using-lstm-networks.html?newdoc) |
| Sequence to sequence | LSTM | Cell array | Categorical vector | Classify musical instrument | [Classify activity over time](https://www.mathworks.com/help/deeplearning/ug/sequence-to-sequence-classification-using-deep-learning.html?newdoc) |

**Summary - Training Options**

Every training option has a default value. Some options do not need to be modified for most problems. Other options, like the number of training epochs or the learning rate, are commonly modified to improve network performance.  
  
Below you can see most options for training deep networks and recommendations for modifying these options. The complete list of training options can be found in [the documentation](https://www.mathworks.com/help/nnet/ref/trainingoptions.html).

**Algorithms**

|  |  |
| --- | --- |
| 'sgdm' - Stochastic gradient descent with momentum | Minimize the loss function by taking small steps in the direction of the negative gradient of the loss, using momentum to reduce oscillation |
| 'rmsprop' - Root mean square propagation | Use learning rates that are different for different parameters |
| 'adam' - Adaptive moment estimation | Similar to RMSProp with momentum |

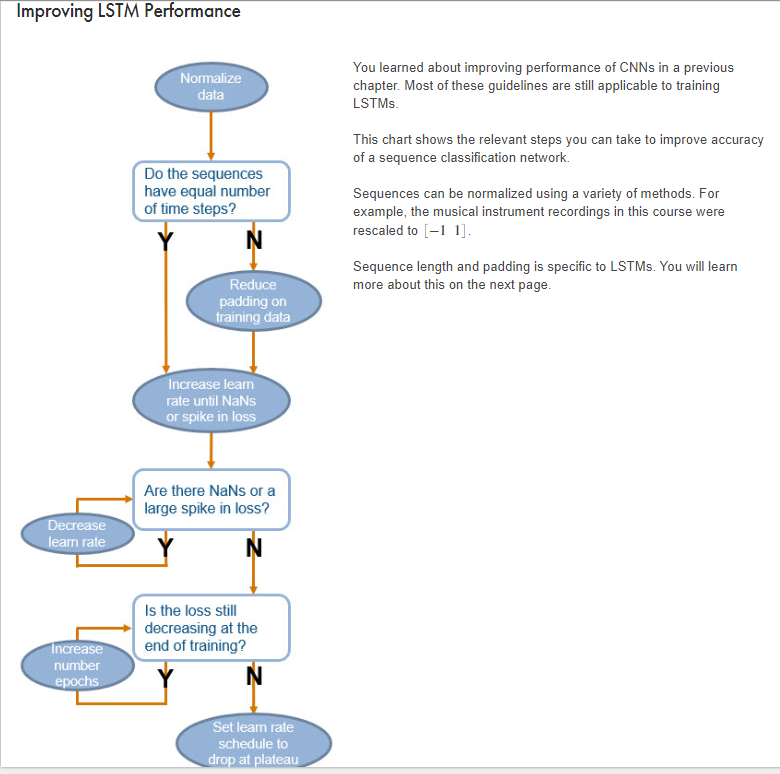
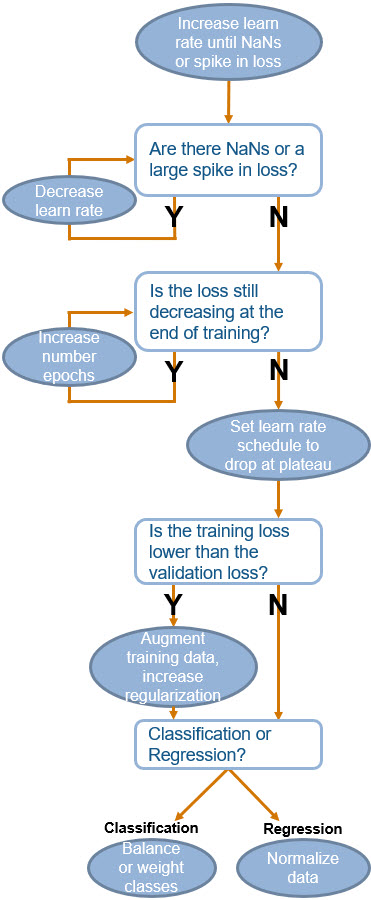
**General Training Options**

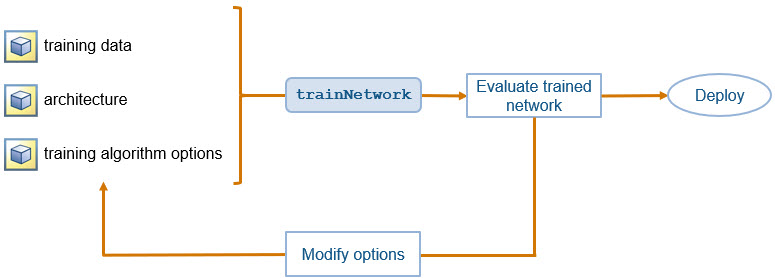
|  |  |
| --- | --- |
| Display | 'Plots', 'Verbose', 'VerboseFrequency' |
| Learning Rate | 'InitialLearnRate', 'LearnRateSchedule', 'LearnRateDropPeriod', 'LearnRateDropFactor' |
| Validation | 'ValidationData', 'ValidationFrequency', 'ValidationPatience' |
| Mini-Batch | 'MaxEpochs', 'MiniBatchSize', 'Shuffle' |
| Sequence | 'SequenceLength', 'SequencePaddingValue' |

**Algorithm Options**

|  |  |
| --- | --- |
| SGDM, Adam, or RMSProp | 'GradientThreshold', 'GradientThresholdMethod', 'L2Regularization' |
| SGDM | 'Momentum' |
| Adam | 'GradientDecayFactor' |
| Adam or RMSProp | 'SquaredGradientDecayFactor', 'Epsilon' |

**Troubleshooting Process**



To train a deep network, you need to create three components:

1. Image or sequence data with known labels to be used as training data.
2. An array of layers representing the network architecture.
3. Options that control the behavior of the training algorithm.

These three components are provided as the inputs to the trainNetwork function which returns the trained network as output.  
  
You should test the performance of the newly trained network. If it is not adequate, typically you should try adjusting some of the training options and retraining

The table below shows a summary of the different tasks you performed with deep learning in this course.

* **Architecture**: A deep network is represented in MATLAB as an array of layers. Architectures are distinguished by the layers in the network. For example, CNNs contain convolutional layers and LSTMs contain LSTM layers.

* **Training Data Format**: For some tasks, there are multiple ways your can organize your training data. This table shows the format you used in this course.

* **Network output**: Data type when this network is used to predict. This output type generally depends on the output layer used in the network architecture.

* **Course Example**: The data set used to accomplish this task in the course.

* **Documentation Example**: Link to relevant documentation example. You can see a list of all examples [here](https://www.mathworks.com/help/nnet/examples.html).

| **Task** | **Architecture** | **Training Data Format** | **Network Output** | **Course Example** | **Documentation Example** |
| --- | --- | --- | --- | --- | --- |
| Image classification | CNN | Image datastore | Categorical | Classify pets | [Classify digits](https://www.mathworks.com/help/deeplearning/ug/create-simple-deep-learning-network-for-classification.html?newdoc) |
| Image regression | CNN | Table | Numeric | Color corrector | [Angles of rotated digits](https://www.mathworks.com/help/deeplearning/ug/train-a-convolutional-neural-network-for-regression.html?newdoc) |
| Object detection | R-CNN | Table | Bounding box | Detect pet faces | [Stop sign detector](https://www.mathworks.com/help/vision/examples/object-detection-using-deep-learning.html) |
| Sequence to label | LSTM | Cell array | Categorical | Classify author | [Classify vowels](https://www.mathworks.com/help/deeplearning/ug/classify-sequence-data-using-lstm-networks.html?newdoc) |
| Sequence to sequence | LSTM | Cell array | Categorical vector | Classify musical instrument | [Classify activity over time](https://www.mathworks.com/help/deeplearning/ug/sequence-to-sequence-classification-using-deep-learning.html?newdoc) |

# Summary - Training Options

Every training option has a default value. Some options do not need to be modified for most problems. Other options, like the number of training epochs or the learning rate, are commonly modified to improve network performance.  
  
Below you can see most options for training deep networks and recommendations for modifying these options. The complete list of training options can be found in [the documentation](https://www.mathworks.com/help/nnet/ref/trainingoptions.html).

## Algorithms

|  |  |
| --- | --- |
| 'sgdm' - Stochastic gradient descent with momentum | Minimize the loss function by taking small steps in the direction of the negative gradient of the loss, using momentum to reduce oscillation |
| 'rmsprop' - Root mean square propagation | Use learning rates that are different for different parameters |
| 'adam' - Adaptive moment estimation | Similar to RMSProp with momentum |

## General Training Options

|  |  |
| --- | --- |
| Display | 'Plots', 'Verbose', 'VerboseFrequency' |
| Learning Rate | 'InitialLearnRate', 'LearnRateSchedule', 'LearnRateDropPeriod', 'LearnRateDropFactor' |
| Validation | 'ValidationData', 'ValidationFrequency', 'ValidationPatience' |
| Mini-Batch | 'MaxEpochs', 'MiniBatchSize', 'Shuffle' |
| Sequence | 'SequenceLength', 'SequencePaddingValue' |

## Algorithm Options

|  |  |
| --- | --- |
| SGDM, Adam, or RMSProp | 'GradientThreshold', 'GradientThresholdMethod', 'L2Regularization' |
| SGDM | 'Momentum' |
| Adam | 'GradientDecayFactor' |
| Adam or RMSProp | 'SquaredGradientDecayFactor', 'Epsilon' |

## Troubleshooting Process

# Deep Learning Resources

**Congratulations!** You now have the MATLAB skills to build, train, and evaluate deep networks.  
  
Below you can find links containing information on other related topics.

## Computing Resources for Training

>>>> ORIGINAL //team/training\_services/elearning/training-app/main/src/main/content/Additional Resources/deepSteps.html#3 ==== THEIRS //team/training\_services/elearning/training-app/main/src/main/content/Additional Resources/deepSteps.html#4 ==== YOURS //gkennedy.selfPaced/training-app/R2019b/src/main/content/Additional Resources/deepSteps.html <<<<

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | [**Scale Up Deep Learning in Parallel and in the Cloud**](https://www.mathworks.com/help/nnet/ug/scale-up-deep-learning-with-multiple-gpus.html) - Description of different computing resources for training deep networks |  | [**Train Network in the Cloud**](https://www.mathworks.com/help/deeplearning/examples/train-a-network-in-the-cloud-using-built-in-parallel-support.html) - Example of training a deep network using built-in parallel support |  | [**Train Network in the Cloud**](https://www.mathworks.com/help/parallel-computing/examples/train-a-network-in-the-cloud-using-built-in-parallel-support.html?newdoc) - Example of training a deep network using built-in parallel support |  | [**Train Network in the Cloud**](https://www.mathworks.com/help/nnet/examples/train-a-network-in-the-cloud-using-built-in-parallel-support.html) - Example of training a deep network using built-in parallel support |
|  |  |  |  |  |  |  |  |

## Deploying to a GPU

|  |  |  |  |
| --- | --- | --- | --- |
|  | [**GPU Coder**](https://www.mathworks.com/products/gpu-coder.html) - Overview of GPU Coder Toolbox |  | [**Code Generation for Deep Learning Networks**](https://www.mathworks.com/help/gpucoder/examples/code-generation-for-deep-learning-networks.html)- Example of code generation for image classification with deep learning |
|  |  |  |  |

## Image Segmentation

|  |  |  |  |
| --- | --- | --- | --- |
|  | [**Overview of Semantic Segmentation**](https://www.mathworks.com/videos/semantic-segmentation-overview-1510858047780.html) - Learn the five major steps that make up semantic segmentation |  | [**Deploy Semantic Segmentation Algorithm**](https://www.mathworks.com/videos/generate-cuda-code-for-a-semantic-segmentation-algorithm-1522147796162.html) - Generate CUDA code using GPU Coder |
|  | [**Semantic Segmentation Basics**](https://www.mathworks.com/help/vision/ug/semantic-segmentation-basics.html) - Documentation page on the semantic segmentation workflow |  | [**Segment Street Images**](https://www.mathworks.com/help/vision/examples/semantic-segmentation-using-deep-learning.html) - Example of semantic segmentation in MATLAB |

## Automated Driving

|  |  |  |  |
| --- | --- | --- | --- |
|  | [**Introduction to Automated Driving System Toolbox**](https://www.mathworks.com/videos/introduction-to-automated-driving-system-toolbox-1501177087798.html) - Learn answers to common questions from automated driving engineers |  | [**Getting Started with Automated Driving System Toolbox**](https://www.mathworks.com/help/driving/getting-started-with-automated-driving-system-toolbox.html) - Documentation hub for automated driving |