MathWorks®
Accelerating the pace of engineering and science

QUICK START GUIDE

Deep Learning with MATLAB

Deep Learning Toolbox™ provides built-in functionality for creating, training, and validating deep neural networks. This reference shows some common use cases. For additional examples, visit the documentation: *mathworks.com/help/deeplearning/examples.html*

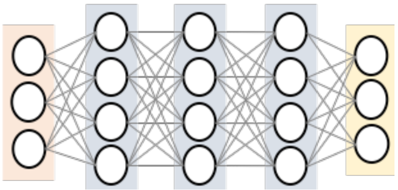
Choosing an Architecture

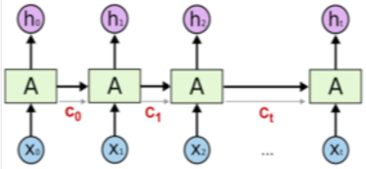
Convolution Neural Network (CNN)

- Image data: classification, detection
- Common layers:
  - Convolution layer
  - Max pooling
  - ReLU layer
  - Batch normalization
- Train from scratch or use transfer learning with pretrained models

Long Short Term Memory (LSTM) Network

- Sequential data: time series forecasting, signal classification, text prediction
- Common layers:
  - LSTM layer
  - BiLSTM layer
- Perform regression or classification tasks





Use the **Deep Network Designer app** to interactively create and evaluate networks

Pretrained Networks

Import Networks

The toolbox provides several functions for exporting models and layers. More can be found on GitHub and *File Exchange*.

Import layers	`importCaffeLayers` `importKerasLayers`
Import network	`importCaffeNetwork` `importKerasNetwork`
Export	`exportONNXNetwork`

Pretrained Models

From Add-on Explorer, use one of the following commands to import a network:

`alexnet`	`vgg19`	`inceptionv3`
`googlenet`	`resnet50`	`squeezenet`
`vgg16`	`resnet101`	

Training Options

Training Options

`Execution Environment`	Parallel, GPU, multi-GPU, auto (default)
`MaxEpochs`	An epoch is one full pass over entire training set
`MiniBatchSize`	Subset of training set to evaluate gradient and update weights
`InitialLearnRate`	A higher initial rate will speed up training but may diverge
`LearnRateSchedule`	Drop the learn rate over time by a factor
`ValidationData`	Validate during training
`ValidationPatience`	Stop training if accuracy is repeated a certain (saves time)

Validation

Inference

`predict` Returns probabilities belonging to each class

`classify` Returns labels and probabilities belonging to each class

```
[Ypred,scores] = classify(net,X);
```

State

Network state can be captured and updated with `predictAndUpdateState` and `classifyAndUpdateState`

Visualization

Several forms of validations and visualizations can be specified through `trainingOptions`

`Plots`	Visualize progress
`Verbose`	Set to true to display training progress each epoch
`VerboseFrequency`	How often to display
`OutputFcn`	Custom function
`CheckpointPath`	Directory to save model each epoch

Improving Accuracy

Improving model accuracy depends on the task and the data. Common approaches include:

Network architecture:

- Use pretrained models from community experts
- Update layers and adjust parameters

Data preparation:

- Add data
- Training/validation/test split
- Normalize data
- Remove outliers
- Balance classes (add weights)

Hyperparameter tuning:

- Tune the training parameters with Bayes optimization
- Set up problem with `optimizableVariable`
- Write function calling model and options
- Perform optimization with `bayesopt`

```
obj = bayesopt(ObjFcn,OptVars,…);
```

Learn more: *mathworks.com/solutions/deep-learning*

mathworks.com