MATLAB EXPO 2018
Demystifying Deep Learning

Dr. Amod Anandkumar
Senior Team Lead – Signal Processing & Communications
What is Deep Learning?
Deep learning is a type of machine learning in which a model learns to perform classification tasks directly from images, text, or sound.

Deep learning is usually implemented using a neural network.

The term “deep” refers to the number of layers in the network—the more layers, the deeper the network.
Deep Learning is **Versatile**

Object Detection Using Faster R-CNN Deep Learning

Semantic Segmentation of Multispectral Images Using Deep Learning

Sequence-to-Sequence Regression Using Deep Learning

Single Image Super-Resolution Using Deep Learning

Classify Image Using GoogLeNet

Time Series Forecasting Using Deep Learning

Classify Text Data Using Deep Learning

Deep Learning Speech Recognition

MATLAB Examples Available Here
Many Network Architectures for Deep Learning

- **Series Network**
  - Single-in single-out
  - AlexNet
  - YOLO

- **Directed Acyclic Graph Network**
  - Multi-in, multi-out
  - No feedback loops
  - ResNet
  - R-CNN

- **Recurrent Network**
  - Memory
  - LSTM

and more (GAN, DQN,...)
Convolutional Neural Networks

Feature Learning

Classification

Input

Convolution + ReLU

Pooling

Convolution + ReLU

Pooling

Flatten

Fully Connected

Softmax

cake ✓
cat x
dog x
Deep Learning Inference in 4 Lines of Code

```
>> net = alexnet;
>> I = imread('peacock.jpg');
>> I1 = imresize(I,[227 227]);
>> classify(net,I1)
ans =
    categorical
    peacock
```
What is Training?

During training, neural network architectures learn features directly from the data without the need for manual feature extraction.
What Happens During Training?

*AlexNet Example*

Layer weights are learned during training

Training Data ➔ Convolution ➔ RelU ➔ rectified linear units ➔ Pooling ➔ Convolution ➔ RelU ➔ rectified linear units ➔ Pooling ➔ Convolution ➔ RelU ➔ rectified linear units ➔ Pooling ➔ Convolution ➔ RelU ➔ rectified linear units ➔ Pooling ➔ Convolution ➔ RelU ➔ rectified linear units ➔ Pooling ➔ Fully Connected layers to support classification

Labels ➔ Flower ➔ Cup ➔ Car ➔ Tree
Visualize Network Weights During Training

*AlexNet Example*

- **Training Data**
- **First Convolution Layer**
- **Convolution**
- **Trained Network**
- **Labels**: Flower, Cup, Car, Tree
Visualization Technique – Deep Dream

```matlab
deepDreamImage(...
    net, 'fc5', channel,
    'NumIterations', 50, ...
    'PyramidLevels', 4,...
    'PyramidScale', 1.25);
```

Synthesizes images that strongly activate a channel in a particular layer

Example Available Here
Visualize Features Learned During Training

*AlexNet Example*

Sample Training Data

Category: Arctic Fox Epoch 17

Features Learned by Network
Visualize Features Learned During Training

AlexNet Example

Category: Flamingo Epoch 10

Sample Training Data

Features Learned by Network
Deep Learning Challenges

Data
- Handling large amounts of data
- Labeling thousands of images & videos

Training and Testing Deep Neural Networks
- Accessing reference models from research
- Optimizing hyperparameters
- Training takes hours-days

Rapid and Optimized Deployment
- Desktop, web, cloud, and embedded hardware

Not a deep learning expert
Example – Semantic Segmentation

- Classify pixels into 11 classes
  - Sky, Building, Pole, Road, Pavement, Tree, SignSymbol, Fence, Car, Pedestrian, Bicyclist

- CamVid dataset

Label Images Using Image Labeler App
Accelerate Labeling With Automation Algorithms

Learn More

Define Ground Truth for Image Collections
Interactively label rectangular ROIs for object detection, pixels for semantic segmentation, and scenes for image classification.

Train an Object Detector from Ground Truth Data
Create training data for object detection using the Image Labeler.

Create Automation Algorithm for Image Labeling
Create a custom tracking algorithm to import into the Image Labeler.

Label Pixels for Semantic Segmentation
Label pixels for semantic segmentation using Image Labeler app.
Perform Bootstrapping to Label Large Datasets

Improve Network

More Ground Truth

Images Videos

Labeling Apps

Ground Truth

Train Network

Propose Labels

Automation Algorithm
Example – Semantic Segmentation

Semantic Segmentation Using Deep Learning

This example shows how to train a semantic segmentation network using deep learning.

A semantic segmentation network classifies every pixel in an image, resulting in an image that is segmented by class. Applications for semantic segmentation include road segmentation for autonomous driving and cancer cell segmentation for medical diagnosis. To learn more, see Semantic Segmentation Basics.

To illustrate the training procedure, this example trains SegNet [1], one type of convolutional neural network (CNN) designed for semantic image segmentation. Other types of networks for semantic segmentation include fully convolutional networks (FCN) and U-Net. The training procedure shown here can be applied to those networks too.

This example uses the CamVid dataset [2] from the University of Cambridge for training. This dataset is a collection of images containing street-level views obtained while driving. The dataset provides pixel-level labels for 32 semantic classes including car, pedestrian, and road.

Learn more about

Setup

This example creates the SegNet network with weights initialized from the VGG-16 network. To get VGG-16, install Neural Network Toolbox™ Model for VGG-16 Network. After installation is complete, run the following code to verify that the installation is correct.

vgg16();
Access Large Sets of Images

Handle Large Sets of Images

Organize Images in Folders
(~ 10,000 images, 5 folders)

imageData = imageDataStore('vehicles')

Easily manage large sets of images
- Single line of code to access images
- Operates on disk, database, big-data file system
Handle Big Image Collections without Big Changes

```matlab
fileLoc = 'FoodImages';
ds = imageDatastore(fileLoc,'IncludeSubfolders',true,...
    'LabelSource','foldernames')
```

```
fileLoc = 'hdfs://hadoop01glnxa64:54310/datasets/FoodImages';
ds = imageDatastore(fileLoc,'IncludeSubfolders',true,...
    'LabelSource','foldernames')
```

**Images in local directory**

```
Files: {
    ...
    ...
    ...
    ... and 1 and 1099 more
}
Labels: [chocolate; apple; apple ... and 10997 more]
ReadSize: 1
ReadFcn: @readDatastoreImage
```

**Images on HDFS**

```
Files: {
    'hdfs://hadoop01glnxa64:54310/datasets/FoodImages/apple_pie/1005649.jpg';
    'hdfs://hadoop01glnxa64:54310/datasets/FoodImages/apple_pie/1011328.jpg';
    'hdfs://hadoop01glnxa64:54310/datasets/FoodImages/apple_pie/101251.jpg'
    ... and 10997 more
}
Labels: [apple_pie; apple_pie; apple_pie ... and 10997 more categorical]
ReadSize: 1
ReadFcn: @readDatastoreImage
```
Import Pre-Trained Models and Network Architectures

Pretrained Models

- alexnet
- vgg16
- vgg19
- googlenet
- inceptionv3
- resnet50
- resnet101
- inceptionresnetv2
- squeezenet

Import Models from Frameworks

- Caffe Model Importer
  (including Caffe Model Zoo)
  - importCaffeLayers
  - importCaffeNetwork

- TensorFlow-Keras Model Importer
  - importKerasLayers
  - importKerasNetwork

Download from within MATLAB
Example – Semantic Segmentation

Semantic Segmentation Using Deep Learning

This example shows how to train a semantic segmentation network using deep learning.

A semantic segmentation network classifies every pixel in an image, resulting in an image that is segmented by class. Applications for semantic segmentation include road segmentation for autonomous driving and cancer cell segmentation for medical diagnosis. To learn more, see Semantic Segmentation Basics.

To illustrate the training procedure, this example trains SegNet [1], one type of convolutional neural network (CNN) designed for semantic image segmentation. Other types of networks for semantic segmentation include fully convolutional networks (FCN) and U-Net. The training procedure shown here can be applied to those networks too.

This example uses the CamVid dataset [2] from the University of Cambridge for training. This dataset is a collection of images containing street-level views obtained while driving. The dataset provides pixel-level labels for 32 semantic classes including car, pedestrian, and road.

Learn more about

Setup

This example creates the SegNet network with weights initialized from the VGG-16 network. To get VGG-16, install Neural Network Toolbox™ Model for VGG-16 Network. After installation is complete, run the following code to verify that the installation is correct.

```matlab
vgg16();
```
Augment Training Images

Training Images

ImageDataAugmenter

rotation
reflection
scaling
shearing
translation

Colour pre-processing
Resize / Random crop / Centre crop

augmentedImageSource

trainNetwork
Tune Hyperparameters to Improve Training

Many hyperparameters
- depth, layers, solver options, learning rates, regularization, ...

Techniques
- Parameter sweep
- Bayesian optimization
Training Performance

Training Performance (per epoch), Titan Xp GPU, Linux Desktop

- ResNet50
- Vgg16
- AlexNet

TensorFlow
MATLAB
MXNet

#ResNet50: ImageNet, Batch-size64
#Vgg16: ImageNet, Batch-size64
#AlexNet: ImageNet, Batch-size32
NVIDIA Tesla V100 32GB

The Fastest and Most Productive GPU for AI and HPC

Volta Architecture

Most Productive GPU

Tensor Core

125 Programmable TFLOPS Deep Learning

Improved NVLink & HBM2

Efficient Bandwidth

Volta MPS

Inference Utilization

Improved SIMT Model

New Algorithms

Visit NVIDIA booth to learn more

| Core | 5120 CUDA cores, 640 Tensor cores |
| Compute | 7.8 TF DP • 15.7 TF SP • 125 TF DL |
| Memory | HBM2: 900 GB/s • 32 GB/16 GB |
| Interconnect | NVLink (up to 300 GB/s) + PCIe Gen3 (up to 32 GB/s) |
Deep Learning on CPU, GPU, Multi-GPU & Clusters

**HOW TO TARGET?**

```matlab
opts = trainingOptions('sgdm', ...
    'MaxEpochs', 100, ...
    'MiniBatchSize', 250, ...
    'InitialLearnRate', 0.00005, ...
    'ExecutionEnvironment', 'auto' );
```

```matlab
opts = trainingOptions('sgdm', ...
    'MaxEpochs', 100, ...
    'MiniBatchSize', 250, ...
    'InitialLearnRate', 0.00005, ...
    'ExecutionEnvironment', 'multi-gpu' );
```

```matlab
opts = trainingOptions('sgdm', ...
    'MaxEpochs', 100, ...
    'MiniBatchSize', 250, ...
    'InitialLearnRate', 0.00005, ...
    'ExecutionEnvironment', 'parallel' );
```
Multi-GPU Performance Scaling

Ease of scaling

- MATLAB “transparently” scales to multiple GPUs
- Runs on Windows!
Examples to Learn More

Train Network in the Cloud Using Built-in Parallel Support

Use parfor to Train Multiple Deep Learning Networks

Use parfeval to Train Multiple Deep Learning Networks

Upload Deep Learning Data to the Cloud

Send Deep Learning Batch Job To Cluster
Example – Semantic Segmentation

Semantic Segmentation Using Deep Learning

This example shows how to train a semantic segmentation network using deep learning.

A semantic segmentation network classifies every pixel in an image, resulting in an image that is segmented by class. Applications for semantic segmentation include road segmentation for autonomous driving and cancer cell segmentation for medical diagnosis. To learn more, see Semantic Segmentation Basics.

To illustrate the training procedure, this example trains SegNet [1], one type of convolutional neural network (CNN) designed for semantic image segmentation. Other types of networks for semantic segmentation include fully convolutional networks (FCN) and U-Net. The training procedure shown here can be applied to those networks too.

This example uses the CamVid dataset [2] from the University of Cambridge for training. This dataset is a collection of images containing street-level views obtained while driving. The dataset provides pixel-level labels for 32 semantic classes including car, pedestrian, and road.

Learn more about

Setup

This example creates the SegNet network with weights initialized from the VGG-16 network. To get VGG-16, install Neural Network Toolbox™ Model for VGG-16 Network. After installation is complete, run the following code to verify that the installation is correct.

```matlab
vgg16();
```
Accelerate Using GPU Coder
Prediction Performance: Fast with GPU Coder

Images/Sec

Prediction (TitanXP GPU)

- AlexNet
- ResNet-50
- VGG-16

~2x faster than TensorFlow

TensorFlow
MATLAB
MXNet
GPU Coder

~2x
Deploying Deep Learning Application

Embedded Hardware

Application logic

Desktop, Web, Cloud

Code Generation

Application Deployment
Deploying Deep Neural Networks to Embedded GPUs and CPUs
15:30–16:15

Designing and deploying deep learning and computer vision applications to embedded CPU and GPU platforms is challenging because of resource constraints inherent in embedded devices. A MATLAB® based workflow facilitates the design of these applications, and automatically generated C or CUDA® code can be deployed on boards like the Jetson TX2 and DRIVE™ PX to achieve very fast inference.

The presentation illustrates how MATLAB supports all major phases of this workflow. Starting with algorithm design, the algorithm may employ deep neural networks augmented with traditional computer vision techniques and can be tested and verified within MATLAB. Next, these networks are trained using GPU and parallel computing support for MATLAB either on the desktop, cluster, or the cloud. Finally, GPU Coder™ generates portable and optimized C/C++ and/or CUDA® code from the MATLAB algorithm, which is then cross-compiled and deployed to CPUs and/or a Tegra® board. Benchmarks show that performance of the auto-generated CUDA code is ~2.5x faster than MXNet, ~5x faster than Caffe2, ~7x faster than TensorFlow®, and on par with TensorRT™ implementation.

Rishu Gupta, Ph.D., Senior Application Engineer, MathWorks India
Addressing Deep Learning Challenges

✓ Perform deep learning without being an expert

✓ Automate ground truth labeling

✓ Create and visualize models with just a few lines of code

✓ Seamless scale training to GPUs, clusters and cloud

✓ Integrate & deploy deep learning in a single workflow
Framework Improvements

- Architectures / layers
  - Regression LSTMs
  - Bidirectional LSTMs
  - Multi-spectral images
  - Custom layer validation

- Data pre-processing
  - Custom Mini-Batch Datastores

- Performance
  - CPU performance optimizations
  - Optimizations for zero learning-rate

- Network training
  - ADAM & RMSProp optimizers
  - Gradient clipping
  - Multi-GPU DAG network training
  - DAG network activations
Deep Learning Network Analyzer

**igraph**

Analysis date: 05-Jan-2018 17:30:42

**ISSUES**

- **Layers**
  - softmax_alone
  - mpool
  - unpool

- **Message**
  - Disconnected layers. All layers in the layer graph must be connected.
  - Unused output. Each layer output must be connected to the input of another layer.
  - Missing input. Each layer input must be connected to the output of another layer.

**ANALYSIS RESULT**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Activations</th>
<th>Learnables</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>Image Input</td>
<td>28x28=1</td>
<td>-</td>
</tr>
<tr>
<td>conv</td>
<td>Convolution</td>
<td>28x28=16</td>
<td>5x5x1=16</td>
</tr>
<tr>
<td>relu</td>
<td>ReLU</td>
<td>28x28=16</td>
<td>-</td>
</tr>
<tr>
<td>softmax</td>
<td>Softmax</td>
<td>Error</td>
<td>-</td>
</tr>
<tr>
<td>conv_b1</td>
<td>Convolution</td>
<td>28x28=32</td>
<td>3x3x16=32</td>
</tr>
<tr>
<td>relu_b1</td>
<td>ReLU</td>
<td>28x28=32</td>
<td>-</td>
</tr>
<tr>
<td>conv_b2</td>
<td>Convolution</td>
<td>28x28=32</td>
<td>3x3x16=32</td>
</tr>
<tr>
<td>relu_b2</td>
<td>ReLU</td>
<td>28x28=32</td>
<td>-</td>
</tr>
<tr>
<td>conv_b3</td>
<td>Convolution</td>
<td>28x28=32</td>
<td>3x3x16=32</td>
</tr>
<tr>
<td>relu_b3</td>
<td>ReLU</td>
<td>28x28=32</td>
<td>-</td>
</tr>
<tr>
<td>conv_b4</td>
<td>Convolution</td>
<td>28x28=32</td>
<td>-</td>
</tr>
</tbody>
</table>

**Add-Ons**

MATLAB EXPO 2018
Creating Custom Architectures

MATLAB provides a simple programmatic interface to create layers and form a network

- **addLayers**
- **removeLayers**
- **connectLayers**
- **disconnectLayers**

… but are all these layers compatible?

```matlab
previous = reluLayer('Name', 'input to inception');

oneByOne = [
    convolution2dLayer(1, 64, 'Stride', 3, 'Name', '1x1')
    reluLayer('Name', 'relu 1x1')
];

threeByThree = [
    convolution2dLayer(1, 96, 'Name', '3x3 reduce')
    reluLayer('Name', 'relu 3x3_reduce')
    convolution2dLayer(3, 128, 'Stride', 3, 'Name', '3x3')
    reluLayer('Name', 'relu 3x3')
];

fiveByFive = [
    convolution2dLayer(1, 16, 'Name', '5x5 reduce')
    reluLayer('Name', 'relu 5x5_reduce')
    convolution2dLayer(5, 32, 'Stride', 3, 'Padding', 0, 'Name', '5x5')
    reluLayer('Name', 'relu 5x5')
];

threeMaxPool = [
    maxPooling2dLayer(3, 'Stride', 3, 'Name', '3x3 pool')
    reluLayer('Name', 'relu 3x3_pool')
    convolution2dLayer(1, 32, 'Name', '1x1 after pool')
    reluLayer('Name', 'relu 1x1 after pool')
];

concat = depthConcatenationLayer(4, 'Name', 'concat');
```
myLayerGraph

Analysis date: 05-Jan-2018 17:20:01

ISSUES

<table>
<thead>
<tr>
<th>Layers</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>concat</td>
<td>Input size mismatch. Size of input to this layer is different from the expected input size.</td>
</tr>
</tbody>
</table>

ANALYSIS RESULT

<table>
<thead>
<tr>
<th></th>
<th>Name</th>
<th>Type</th>
<th>Activations</th>
<th>Learnables</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2x28 grey 28x40x1 images with 'zerocenter' normalization</td>
<td>Image Input</td>
<td>28x40x1=1</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>very first conv 20x5x5x1 convolutions with stride [1 1] and padding [0 0 0]</td>
<td>Convolution</td>
<td>24x36x20</td>
<td>5x5x1x20</td>
</tr>
<tr>
<td>3</td>
<td>input to inception</td>
<td>ReLU</td>
<td>24x36x20</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>1x1x1x20 convolutions with stride [3 3] and padding [0 0 0]</td>
<td>Convolution</td>
<td>8x12x64</td>
<td>1x1x20x64</td>
</tr>
<tr>
<td>5</td>
<td>relu 1x1</td>
<td>ReLU</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>3x3 reduce 96x1x1x20 convolutions with stride [1 1] and padding [0 0 0]</td>
<td>Convolution</td>
<td>-</td>
<td>3x3x96x128</td>
</tr>
<tr>
<td>7</td>
<td>relu3x3_reduce</td>
<td>ReLU</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>3x3</td>
<td>ReLU</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>relu3x3</td>
<td>ReLU</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>5x5 reduce 18x1x1x10 convolutions with stride [1 1] and padding [0 0 0]</td>
<td>Convolution</td>
<td>24x36x16</td>
<td>1x1x20x16</td>
</tr>
<tr>
<td>11</td>
<td>relu5x5_reduce</td>
<td>ReLU</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>12</td>
<td>5x5</td>
<td>ReLU</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Layer: relu 1x1
Output: 8 x 12 x 64
The image shows a neural network graph with layers labeled as 2x20, very 3x3, ... 3x3 pool, etc. The highlighted layer is labeled as relu 3x3. The corresponding table in the analysis result section shows a layer with the dimensions 8 x 12 x 128.
Input size mismatch. Size of input to this layer is different from the expected input size.

**Analysis Result**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Activations</th>
<th>Learnables</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>3x3 128 3x3x96</td>
<td>Convolution</td>
<td>8x12x128, 1x1x128, 1x1x128</td>
</tr>
<tr>
<td>9</td>
<td>ReLU</td>
<td>8x12x128</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>Convolution</td>
<td>24x36x16</td>
<td>1x1x16, 1x1x16, 1x1x16</td>
</tr>
<tr>
<td>11</td>
<td>ReLU</td>
<td>24x36x16</td>
<td>-</td>
</tr>
<tr>
<td>12</td>
<td>Convolution</td>
<td>7x11x32</td>
<td>5x5x16x32, 1x1x32</td>
</tr>
<tr>
<td>13</td>
<td>ReLU</td>
<td>8x12x128</td>
<td>-</td>
</tr>
<tr>
<td>14</td>
<td>Max Pooling</td>
<td>8x12x32</td>
<td>-</td>
</tr>
<tr>
<td>15</td>
<td>ReLU</td>
<td>8x12x128</td>
<td>-</td>
</tr>
<tr>
<td>16</td>
<td>Convolution</td>
<td>8x12x32</td>
<td>1x1x20x32, 1x1x32</td>
</tr>
<tr>
<td>17</td>
<td>ReLU</td>
<td>8x12x32</td>
<td>-</td>
</tr>
<tr>
<td>18</td>
<td>Depth concatenation</td>
<td>Error</td>
<td>-</td>
</tr>
<tr>
<td>19</td>
<td>Max Pooling</td>
<td>Error</td>
<td>-</td>
</tr>
</tbody>
</table>
Create the layers

```matlab
previous = reluLayer('Name', 'Input to inception');

oneByOne = [
    convolution2dLayer(1, 64, 'Stride', 3, 'Name', '1x1')
    reluLayer('Name', 'relu 1x1')
];

threeByThree = [
    convolution2dLayer(1, 96, 'Name', '3x3 reduce')
    reluLayer('Name', 'relu 3x3_reduce')
    convolution2dLayer(3, 128, 'Stride', 3, 'Name', '3x3')
    reluLayer('Name', 'relu 3x3')
];

fivebyFive = [
    convolution2dLayer(1, 16, 'Name', '5x5 reduce')
    reluLayer('Name', 'relu 5x5_reduce')
    convolution2dLayer(5, 32, 'Stride', 3, 'Padding', 1, 'Name', '5x5')
    reluLayer('Name', 'relu 5x5')
];

threeMaxPool = [
    maxPooling2dLayer(3, 'Stride', 3, 'Name', '3x3_pool')
    reluLayer('Name', 'relu 3x3_pool')
    convolution2dLayer(1, 32, 'Name', '1x1 after pool')
    reluLayer('Name', 'relu 1x1 after pool')
];

concat = depthConcatenationLayer(4, 'Name', 'concat');
```

Change the padding from zero to one
myLayerGraph
Analysis date: 05-Jan-2018 17:23:00

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Activations</th>
<th>Learnables</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2x28 grey 28x40x1 images with 'zerocenter' normalization</td>
<td>Image Input</td>
<td>28x40x1</td>
<td>-</td>
</tr>
<tr>
<td>2 very first conv 20 5x5x1 convolutions with stride [1 1] and padding [0 0 0]</td>
<td>Convolution</td>
<td>24x36x20</td>
<td>5x5x1x20</td>
</tr>
<tr>
<td>3 input to inception ReLU</td>
<td>ReLU</td>
<td>24x36x20</td>
<td>-</td>
</tr>
<tr>
<td>4 1x1 64 1x1x20 convolutions with stride [3 3] and padding [0 0 0]</td>
<td>Convolution</td>
<td>8x12x64</td>
<td>1x1x20x64</td>
</tr>
<tr>
<td>5 relu 1x1 ReLU</td>
<td>ReLU</td>
<td>8x12x64</td>
<td>-</td>
</tr>
<tr>
<td>6 3x3 reduce 86 3x3x20 convolutions with stride [1 1] and padding [0 0 0]</td>
<td>Convolution</td>
<td>24x36x96</td>
<td>1x1x20x96</td>
</tr>
<tr>
<td>7 relu 3x3_reduce ReLU</td>
<td>ReLU</td>
<td>24x36x96</td>
<td>-</td>
</tr>
<tr>
<td>8 3x3 128 3x3x96 convolutions with stride [3 3] and padding [0 0 0]</td>
<td>Convolution</td>
<td>8x12x128</td>
<td>3x3x96x128</td>
</tr>
<tr>
<td>9 relu 3x3 ReLU</td>
<td>ReLU</td>
<td>8x12x128</td>
<td>-</td>
</tr>
<tr>
<td>10 5x5 reduce 16 1x1x20 convolutions with stride [1 1] and padding [0 0 0]</td>
<td>Convolution</td>
<td>24x36x16</td>
<td>1x1x20x16</td>
</tr>
<tr>
<td>11 relu 5x5_reduce ReLU</td>
<td>ReLU</td>
<td>24x36x16</td>
<td>-</td>
</tr>
<tr>
<td>12 5x5 32 5x5x16 convolutions with stride [3 3] and padding [1 1 1]</td>
<td>Convolution</td>
<td>8x12x32</td>
<td>5x5x16x32</td>
</tr>
<tr>
<td>13 relu 5x5 ReLU</td>
<td>ReLU</td>
<td>8x12x32</td>
<td>-</td>
</tr>
<tr>
<td>14 3x3 pool 3x3 max pooling with strides [3 3] and padding [0 0 0]</td>
<td>Max Pooling</td>
<td>8x12x20</td>
<td>-</td>
</tr>
<tr>
<td>15 relu 3x3_pool ReLU</td>
<td>ReLU</td>
<td>8x12x20</td>
<td>-</td>
</tr>
<tr>
<td>16 1x1 after pool 32 1x1x32 convolutions with strides [1 1] and padding [1 0 0 1]</td>
<td>Convolution</td>
<td>8x12x32</td>
<td>1x1x32</td>
</tr>
</tbody>
</table>
Call to Action – Deep Learning Onramp

Free Introductory Course

Deep Learning Onramp

This free self-paced course provides an interactive introduction to practical deep learning. It focuses on using MATLAB® to apply deep learning methods to perform image recognition. The course consists of hands-on exercises and short videos. In the exercises, you will enter commands in an online version of MATLAB and receive contextual feedback that will help you correct common mistakes. Topics include:

- Convolutional neural networks
- Preprocessing images
- Using pretrained networks
- Transfer learning
- Evaluating network performance

Available Here
MATLAB Deep Learning Framework

**Access Data**
- Manage large image sets
- Automate image labeling

**Design + Train**
- Easy access to models
- Acceleration with GPU’s
- Scale to GPUs & clusters

**Deploy**
- Optimized inference deployment
  - Target NVIDIA, Intel, ARM platforms
• Share your experience with MATLAB & Simulink on Social Media
  ▪ Use #MATLABEXPO

• Share your session feedback:
  Please fill in your feedback for this session in the feedback form