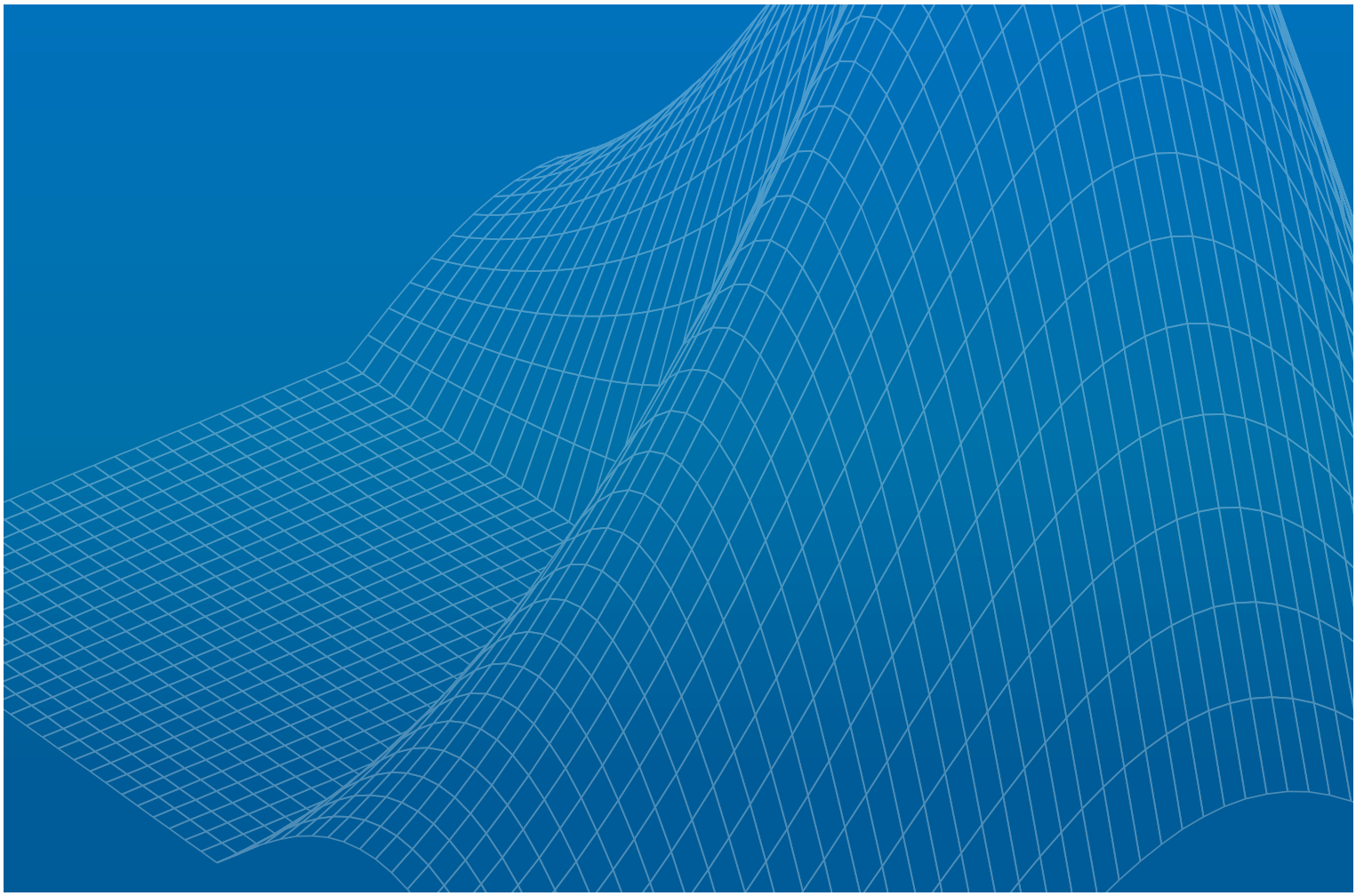

Deep Learning in the Cloud with MATLAB R2016b

By Stuart Moulder, Tish Sheridan, Amanjit Dulai, Giuseppe Rossini

WHITE PAPER

MathWorks®

Introduction

You can use MATLAB® to perform deep learning in the cloud using Amazon Elastic Compute Cloud (Amazon EC2) with new P2 instances and data stored in the cloud. Deep learning in the cloud can save you lots of time if you have big data and models that take hours or days to train.

Deep learning is much faster when you can use high performance GPUs for training. If you don't have a suitable GPU available, you can use the new Amazon EC2 P2 instances to experiment. Try it out using machines with a single GPU, and later scale up to 8 or 16 GPUs per machine to accelerate training, using parallel computing to train multiple models at once on the same data. You can compare and explore the performance of multiple deep neural network configurations to look for the best tradeoff of accuracy and memory use.

Read the following sections to learn:

- How to train, test, and explore neural networks for deep learning problems in MATLAB
- How to scale up deep learning using high performance GPU machines in the Amazon Web Services cloud

Deep Learning in MATLAB

Deep learning is a branch of machine learning that teaches computers to do what comes naturally to humans and animals: learn from experience. Machine learning algorithms use computational methods to "learn" information directly from data without relying on a predetermined equation as a model. Deep learning is especially suited for image recognition, which is important for solving problems such as face recognition, motion detection, and many advanced driver assistance technologies such as autonomous driving, lane detection, and autonomous parking.

Deep learning uses neural networks to learn useful representations of features directly from data. Neural networks combine multiple nonlinear processing layers, using simple elements operating in parallel inspired by biological nervous systems. Deep learning models can achieve state-of-the-art accuracy in object classification, sometimes exceeding human-level performance. You train models using a large set of labeled data and neural network architectures that contain many layers, usually including some convolutional layers. Training these models is extremely computationally intensive and you can usually accelerate training by using a high specification GPU.

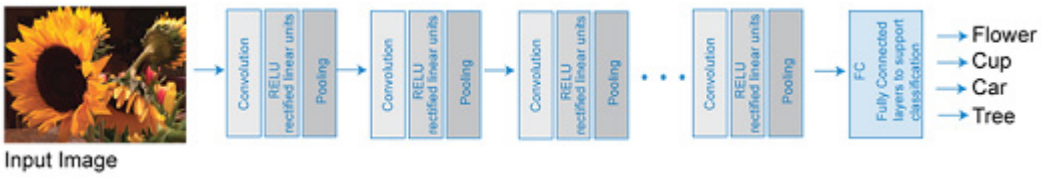

Figure 1: Example of an image classification model

For this paper, we use a relatively simple network that demonstrates the principles involved. You can use these same steps to work with larger networks and data-sets. We create a network to classify images and show how easy it is to use MATLAB for deep learning, even without extensive knowledge of advanced computer vision algorithms or neural networks.

The goal is to classify images into the correct class. The dataset is the CIFAR-10 dataset, which is an established computer-vision dataset used for object recognition. The labeled images were collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton (see ***Learning Multiple Layers of Features from Tiny Images***[1], Alex Krizhevsky, 2009). The CIFAR-10 dataset is a commonly used benchmark in machine learning, because it is complex enough to develop interesting models, but not so large that it takes days to train. The dataset contains 60,000 32x32 color images in 10 classes, with 6000 images per class. Here are the classes in the dataset, showing 10 random images from each class.

Figure 2: Classes and example images from the CIFAR-10 dataset

[1] *https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf*

Create a Deep Network

Neural Network Toolbox™ [2] provides simple MATLAB commands for creating the layers of a deep neural network and connecting them together. We expect that to solve the problem, the network will need a standard set of layers for a convolutional neural network: convolution, pooling, rectified linear unit (ReLU) nonlinearities, and local contrast normalization with a linear classifier on top of it all.

The following code shows how to specify and train a single network on your local machine.

1. Specify the network layers. The following code creates an array of eleven layers, where the first layer receives input images and the last layer classifies the image by returning the category.

```
layers = [
     imageInputLayer([32 32 3])
     convolution2dLayer(5,20)
     reluLayer()
     averagePooling2dLayer(3)
     crossChannelNormalizationLayer(3)
     convolution2dLayer(5,40)
     reluLayer()
     averagePooling2dLayer(3)
     fullyConnectedLayer(10)
     softmaxLayer()
     classificationLayer()
     ];
```

To learn more about any of the layers, see the *Neural Network Toolbox documentation*[3].

2. https://www.mathworks.com/products/neural-network/
3. https://www.mathworks.com/help/nnet/

2. Specify training options to define the training method and its parameters. The options here will train for 30 epochs with a learning rate of 0.001, then reduce the learning rate by a factor of 10, and continue training for another 10 epochs. Each epoch means one full training cycle on the whole training set. Verbose set to true displays the progress of training.

```
% Define the training options
options = trainingOptions('sgdm', ...
      'InitialLearnRate', 0.001, ...
      'LearnRateSchedule', 'piecewise', ...
      'LearnRateDropFactor', 0.1, ...
      'LearnRateDropPeriod', 30, ...
      'L2Regularization', 0.004, ...
      'MaxEpochs', 40, ...
      'MiniBatchSize', 100, ...
      'Verbose', true);
```

3. Supply the set of labeled training images to **imageDatastore**, specifying where you have saved the data. You can use an **imageDatastore** to efficiently access all of the image files. **imageDatastore** is designed to read batches of images for faster processing in machine learning and computer vision applications. **imageDatastore** can import data from image collections that are too large to fit in memory.

```
% Define the training data
imdsTrain = imageDatastore('cifar-10-32-by-32/cifar10Train',...
      'IncludeSubfolders',true,...
      'LabelSource','foldernames');
```

The CIFAR-10 images are split into a set to use for training and a second set to use for testing. The training set in this example is in a local folder called 'cifar-10-32-by-32/cifar10Train'.

To get the data, see *Appendix - Prepare the CIFAR-10 data.*

4. To train the network, use the trainNetwork function:

```
net = trainNetwork(imdsTrain,layers,options);
```

The result is a fully trained network that you can use to classify new images.

Test the Network

After you create a fully trained network, you can use it to classify a new set of images and measure how accurate it is. The following code tests the accuracy of classification using the test set. The accuracy score is the percentage of correctly classified images using the test set.

```
% Define the testing data
imdsTest = imageDatastore('cifar-10-32-by-32/cifar10Test',...
     'IncludeSubfolders',true,...
     'LabelSource','foldernames');

% Measure the accuracy
yTest = classify(net,imdsTest);
accuracy = (sum(yTest == imdsTest.Labels) / numel(imdsTest.Labels));
```

Try Different Models

Having trained one network, we want to see whether modifying this network can improve the accuracy of the model in classifying images correctly. There are many ways to configure your network, but for this example we investigate changing the number of filters, filter size, pooling size, and adding additional convolutional layers. The convolutional and fully connected layers add learnable parameters, which might increase accuracy but will increase memory usage. When we have a selection of trained models to compare, we can assess them to choose the best tradeoff of accuracy and memory footprint. For the complete MATLAB script training the networks in parallel, see *Appendix – MATLAB Code*.

Training all these different models on just a desktop PC is going to take a long time. Instead, we want to make use of a high specification multi-GPU machine. Amazon can provide us with suitable machines on demand using their new P2 instances. In the following sections, you can learn how to reserve a P2 instance, how to connect to the data, and then how to simultaneously train models in the cloud.

Scale Up to Deep Learning in the Cloud

To try out deep learning in the cloud, you need:

- MATLAB R2016b, Neural Network Toolbox, Parallel Computing Toolbox
- A MathWorks Account
- Access to *MATLAB Distributed Computing Server for Amazon EC2*[4].
- An Amazon Web Services account

Connecting to Amazon EC2 Using MathWorks Cloud Center

Amazon Elastic Compute Cloud (Amazon EC2) is a web service which you can use to set up compute capacity and storage in the cloud. Amazon EC2 is ideally suited for the intensive computational demands and large datasets found in deep learning. By using Amazon EC2, you can economically scale up your computing resources and gain access to domain-specific hardware. The new *Amazon EC2 P2*[5] instances are specifically designed for compute-intensive applications, providing up to 16 NVIDIA Tesla K80 GPUs per machine. You can use a single GPU to take advantage of the parallel nature of neural networks, dramatically reducing the time required to train a single model. You can use multiple GPUs to try more models, allowing you to train multiple models simultaneously. You can scale up beyond the desktop, and scale in a flexible way without requiring any long-term commitment.

MathWorks Cloud Center is a web application for creating and accessing compute clusters in the Amazon Web Services cloud for parallel computing with MATLAB. You can access a cloud cluster from your client MATLAB® session like any other cluster in your own onsite network. To learn more, see *MATLAB Distributed Computing Server for Amazon EC2*[6].

For instructions to help you set up your credentials and then create a new cluster, see *Create and Manage Clusters*[7] in the Cloud Center documentation. The main steps are:

1. Log in to *Cloud Center*[8] using your MathWorks account email address and password.
2. Click **User Preferences** and follow the on-screen instructions to set up your Amazon Web Services (AWS) credentials. For help, see the Cloud Center documentation: *Set Up Your Amazon Web Services (AWS) Credentials*[9]
3. To create a cluster of Amazon EC2 instances, click **Create a Cluster**.
4. Choose the settings as shown in the following screenshot. To create a cluster suitable for deep learning, you must configure the **Machine Type** to include high performance GPUs. Choose a machine type with multiple GPUs per machine to train multiple networks in parallel.

4. http://www.mathworks.com/products/parallel-computing/parallel-computing-on-the-cloud/distriben-ec2.html
5. https://aws.amazon.com/ec2/instance-types/p2/
6. http://www.mathworks.com/products/parallel-computing/parallel-computing-on-the-cloud/distriben-ec2.html
7. http://www.mathworks.com/help/cloudcenter/ug/cloud-computing-console.html
8. https://cloudcenter.mathworks.com/login
9. http://www.mathworks.com/help/cloudcenter/ug/cloud-computing-console.html#butnm_c-1

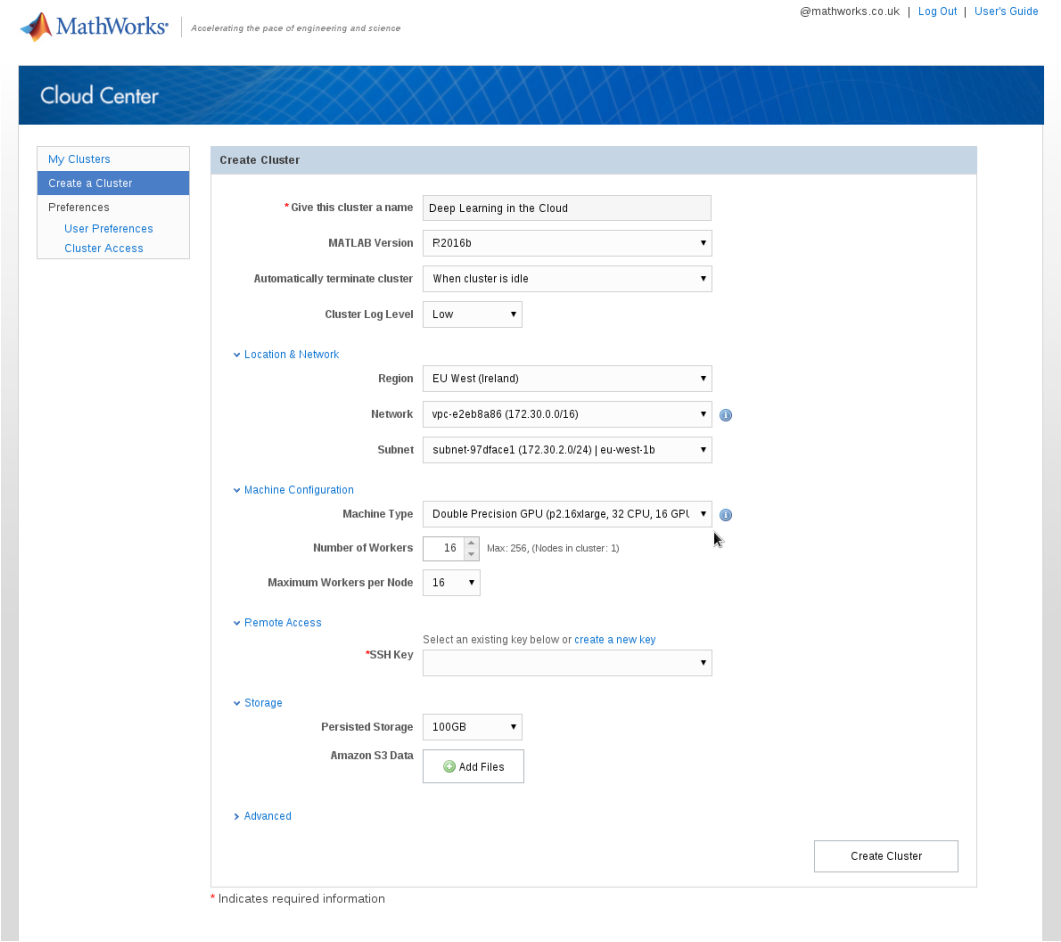

Figure 3 Cloud Center: Create a cluster with multiple GPUs

5. Click **create a new key** so that you can download and save the SSH key for root access. You will need this to log in using SSH and set up your cluster to access the data in a later step.
6. Click **Create Cluster**.
7. To access your cluster from MATLAB, use **Parallel > Discover Clusters** to search for your Amazon EC2 clusters. When you select the cluster, the wizard automatically sets it as your default cluster.
8. On the MATLAB Home tab, select **Parallel > Parallel Preferences**, and set **Preferred number of workers** to **16**.

Check if your cluster is online, either from Cloud Center, or from within MATLAB by creating a cluster instance and displaying details:

```
cluster = parcluster();
disp(cluster);
```

By default, if the cluster is left idle for too long it automatically shuts down to avoid incurring unwanted expense. If your cluster has shut down, bring it back online either from Cloud Center by clicking **Start Up**, or from MATLAB by entering:

```
start(cluster);
```

After your cluster is online, query the GPU device of each worker:

```
wait(cluster)
spmd
    disp(gpuDevice());
end
```

This returns details of the GPU device visible to each worker process in your cluster. The **spmd** block automatically starts the workers in a parallel pool, if you have default preferences. The first time you create a parallel pool on a cloud cluster can take several minutes.

You can start or shut down the parallel pool using the Parallel Pool menu in the bottom left of the MATLAB desktop.

To learn more about using parallel pools, see the *Parallel Pools documentation*[10].

Using Data in Amazon EBS Volumes

In this example, we choose to store our data in Amazon Elastic Block Store (Amazon EBS). Amazon EBS is a service which provides flexible, scalable, and durable storage in the cloud. You can use EBS volumes with your Amazon EC2 instances. To create a volume, we used the AWS Console to create a 100GB general purpose solid state drive (SSD gp2). This is Amazon Web Service's default volume type and is suitable for most applications.

AWS provides many options to configure your storage, including volume type and size. The optimal choice of storage depends on your Amazon EC2 instance type and the read/write requirements of your application. To learn more about the different Amazon EBS volumes available, see the *Amazon EBS Volume Types documentation*[11].

You must attach and mount the EBS volume for your cluster. For instructions, see *Appendix – Transfer Data to EBS Volume.*

10. https://www.mathworks.com/help/distcomp/parallel-pools.html
11. http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSVolumeTypes.html

To access your training data in MATLAB, you can simply create an **imageDatastore** pointing to the appropriate location. This example used /mnt/datastore.

```
parfor ii = 1:numNetworksToTrain

    % Define the Amazon EBS storage training data (must run on the
cluster)

    imdsTrain = imageDatastore(...

        '/mnt/datastore/cifar-10-32-by-32/cifar10Train', ...

        'IncludeSubfolders',true,...

        'LabelSource','foldernames');...
 % SEE APPENDIX FOR COMPLETE SCRIPT
```

You need to run this **imageDatastore** command inside a parallel for-loop because the data is only visible to the workers on the cluster. You can see this command inside the entire parfor loop in the next section. If you run the code in your local MATLAB client and not in parallel, it will error because the folder only exists on the cluster.

Note: when your cluster shuts down, the Amazon EBS volume is automatically detached. Your data is retained, but when you want to use it again in an online cluster, you must repeat the steps to attach and mount the volume.

Deep learning Using MATLAB and Amazon EC2

Having used MathWorks Cloud Center and Amazon EC2 to access sufficient compute capacity, and Amazon EBS Volumes to store and access our dataset, we now want to quickly train the optimal network for our problem. In this example we are searching for the network which gives the best balance between high classification accuracy and low memory footprint. We test 16 different networks, and for each model store the classification accuracy and memory footprint. We investigated changing the number of filters, filter size, pooling size, and adding additional convolutional layers.

Training and validating each model is an independent process, so you can use parallel computing to compute multiple models simultaneously. To do this, you can iterate over the different models using a parfor loop. A parfor loop executes loop iterations in parallel on MATLAB workers in the parallel pool and parfor distributes the work among as many workers as you have available in the pool.

Following is the part of the MATLAB script that trains and tests the 16 different networks in parallel. To see the entire script including the supporting functions, see *Appendix – MATLAB Code.*

```
%% Train all network configurations
parfor idx = 1:numNetworksToTrain
    % Get the network configuration to train on this worker
    layers = networks{idx};

    % Load the training and test data
    imdsTrain = imageDatastore('/mnt/datastore/cifar10Train',...
        'IncludeSubfolders',true,...
        'LabelSource','foldernames');
    imdsTest = imageDatastore('/mnt/datastore/cifar10Test',...
        'IncludeSubfolders',true, ...
        'LabelSource','foldernames');

    % Train the network
    net = trainNetwork(imdsTrain,layers,options);

    % Record the memory footprint for this network
    memoryFootprints(idx) = getMemoryFootprint(net);

    % Record the accuracy for this network
    yTest = classify(net,imdsTest);
    accuracies(idx) = sum(yTest == imdsTest.Labels) / numel(imdsTest.
Labels);

    % Save the network
    trainedNetworks{idx} = net;
end
% SEE APPENDIX FOR COMPLETE SCRIPT
```

Results

The following plot shows the results of training, validating, and characterizing the 16 different chosen networks. Using these results, we can identify which network to use depending on our accuracy and memory requirements. The best model is a tradeoff choice. Memory footprint might be an important consideration, for example if you need the final network to run on an embedded processor in a car.

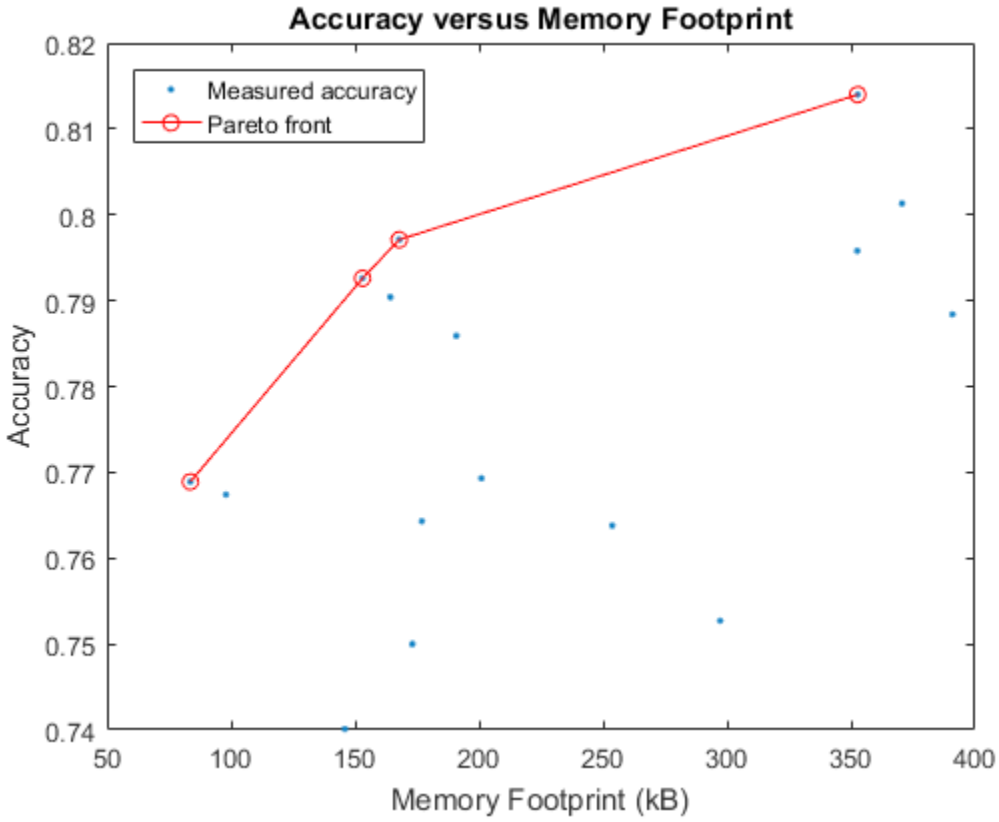

Figure 4: Classification accuracy versus memory footprint for the different model parameters tested

The graph shows accuracy against memory footprint. The red line shows the networks along the Pareto front – these are the optimal choices for accuracy against memory. Points that do not lie along the Pareto front all have worse accuracy or memory use. The best model is one of the points on the Pareto front, and which to choose depends on memory constraints and acceptable accuracy.

The memory footprint of a neural network is proportional to the number of learnable parameters, which come from the convolutional and fully connected layers. You might assume that more learnable parameters lead to better performance and so you should choose the largest model that will fit memory constraints. However, sometimes larger models can have worse performance than smaller models. More learnable parameters can sometimes lead to overfitting, especially if a large amount of the learnable parameters are in the final fully connected layer. You can use Dropout to mitigate this.

A larger model can also take longer to converge, and may require more epochs of training than a smaller model.

We investigated changing the number of filters, filter size, pooling size, and adding additional convolutional layers. To see all the settings, examine the code in the appendix. These types of parameter sweeps are very important for finding the best performance, but training each network in series can be prohibitively slow. Using a parallel pool gives multiple workers, but on a single GPU machine competition between workers for GPU resources limits the performance of each worker, reducing the parallel performance of the parfor loop. As described earlier, you can use MathWorks Cloud Center to start a cluster on Amazon EC2 with the necessary software and GPU resources to massively parallelize this sort of parameter sweep with MATLAB.

By creating a cluster using an Amazon EC2 p2.16xlarge instance with 16 GPUs, you can get up to 16 workers on a single machine, each with their own processor thread and GPU. We now repeat the same model parameter sweep, but using a different number of workers each time. Below we show the speedup as a function of the number of workers, defining speedup as the number of networks trained per unit time.

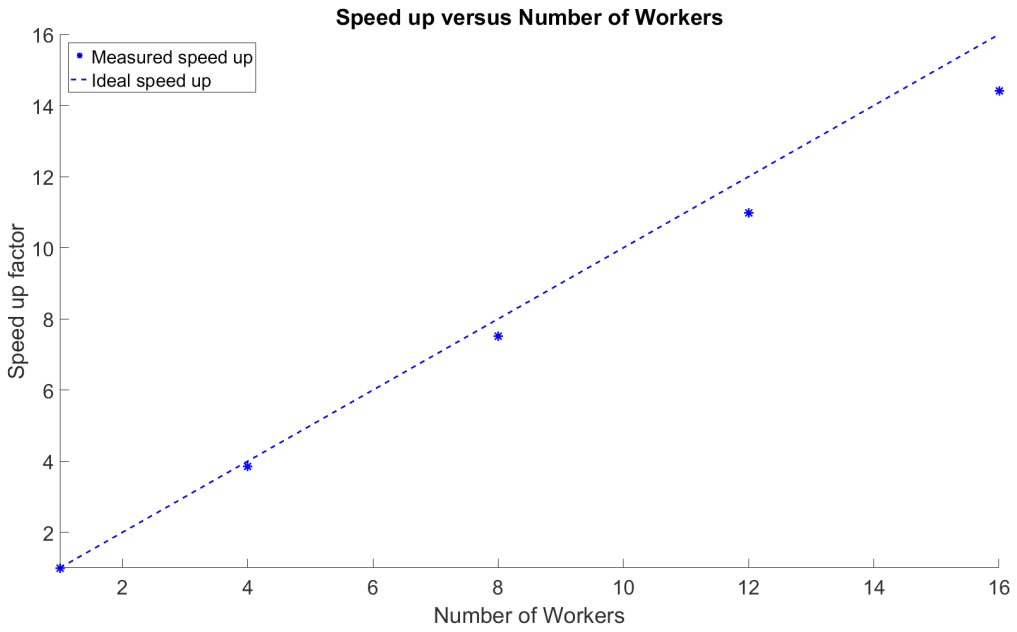

Figure 5 Speed up factor versus the number of workers for P2.16xlarge Amazon EC2 instance

These results show that the parallel performance of these instances scales well with the number of workers. You can get more workers and more GPUs by using Cloud Center to run MATLAB on powerful Amazon EC2 machines that speed up your deep learning calculations.

Conclusions

In this paper we show how you can use MATLAB R2016b to explore the performance of various deep neural network configurations, with accelerated training using the computing power of Amazon EC2 and cloud data storage. The code provides a worked example showing how to train a network to classify images, and use it to classify a new set of images plus measure the accuracy of the classification and its memory footprint. You can use a parfor loop to train multiple models simultaneously by distributing the work among as many MATLAB workers as you have available, and you can switch to processing on a cluster without leaving your client MATLAB.

Useful Links:

For more information, see the following resources:

- ***mathworks.com/discovery/deep-learning.html***
  Central resource for deep Learning with MATLAB
- ***mathworks.com/help/nnet/convolutional-neural-networks.html***
  Neural Network Toolbox documentation on essential tools for deep learning
- ***Learning Multiple Layers of Features from Tiny Images***[12], Alex Krizhevsky, 2009
  About the CIFAR-10 dataset
- ***mathworks.com/products/parallel-computing/parallel-computing-on-the-cloud/***
- ***https://aws.amazon.com/console/***

See the appendices for MATLAB code and steps for setting up data on a cluster.

12. *https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf*

Appendix – MATLAB Code

The following MATLAB code trains the selection of neural networks in parallel, and generates the plot shown in the Results section. The supporting functions define which parameters to sweep in the different networks and compute the memory footprints of the trained networks.

```
% Create the 16 different network configurations we want to test
networks = createTestNetworks();

% Allocate variables for networks, memory footprints, and accuracy
numNetworksToTrain = numel(networks);
trainedNetworks = cell(numNetworksToTrain,1);
memoryFootprints = zeros(numNetworksToTrain,1);
accuracies = zeros(numNetworksToTrain,1);

% Define the training options
% For a quick test, try setting MaxEpochs to 1.
options = trainingOptions('sgdm',...
    'InitialLearnRate',0.001,...
    'LearnRateSchedule','piecewise',...
    'LearnRateDropFactor',0.1,...
    'LearnRateDropPeriod',120,...
    'L2Regularization',0.004,...
    'MaxEpochs',140,...
    'MiniBatchSize',100,...
    'Verbose',false);

% Start parallel pool of 16 workers
p = gcp('nocreate');
if isempty(p)
    parpool(16);
elseif p.NumWorkers ~= 16
    delete(p);
```

```
    parpool(16);
end

%% Train all network configurations
disp('Starting parallel training ...');
parfor idx = 1:numNetworksToTrain
    % Get the network configuration to train on this worker
    layers = networks{idx};

    % Load the training and test data
    imdsTrain = imageDatastore('/mnt/datastore/cifar10Train',...
        'IncludeSubfolders',true,...
        'LabelSource','foldernames');
    imdsTest = imageDatastore('/mnt/datastore/cifar10Test',...
        'IncludeSubfolders',true,...
        'LabelSource','foldernames');

    % Train the network
    net = trainNetwork(imdsTrain,layers,options);

    % Record the memory footprint for this network
    memoryFootprints(idx) = getMemoryFootprint(net);

    % Record the accuracy for this network
    yTest = classify(net,imdsTest);
    accuracies(idx) = sum(yTest == imdsTest.Labels) / numel(imdsTest.
Labels);

    % Save the network
    trainedNetworks{idx} = net;
end
disp('Finished parallel training ...');
```

```
%% Plot the memory footprints against the accuracy
scatter(memoryFootprints,accuracies);
title('Accuracy against Memory Footprint');
xlabel('Memory Footprint in Bytes');
ylabel('Accuracy');

%--------------------------------------------------------------------------
function networks = createTestNetworks()
% Network parameters to iterate over
numFilters1Values =       [20 32];
filterSizeValues =        [5 3];
poolSizeValues =          [3 2];
useExtraLayerValues =     [0 1];

numNetworksToTrain = 16;
numFilters1Sweep = repmat(repelem(numFilters1Values,8),1,1);
filterSizeSweep = repmat(repelem(filterSizeValues,4),1,2);
poolSizeSweep = repmat(repelem(poolSizeValues,2),1,4);
useExtraLayerSweep = repmat(repelem(useExtraLayerValues,1),1,8);

networks = cell(numNetworksToTrain,1);

for idx = 1:numNetworksToTrain

    numFilters1 = numFilters1Sweep(idx);
    numFilters2 = numFilters1 * 2;
    filterSize = filterSizeSweep(idx);
    poolSize = poolSizeSweep(idx);
    useExtraLayer = useExtraLayerSweep(idx);

    if useExtraLayer
```

```
        extraLayer = [
            convolution2dLayer(filterSize,numFilters1,'Padding',2,...
            'BiasLearnRateFactor',2)
            maxPooling2dLayer(poolSize,'Stride',2)
            reluLayer()
            crossChannelNormalizationLayer(3,'Alpha',5e-05,'Beta',0.75,'K',1)
            ];
    else
        extraLayer = [];
    end

    networks{idx} = [
        imageInputLayer([32 32 3])
        extraLayer
        convolution2dLayer(filterSize,numFilters1,'Padding',2,...
        'BiasLearnRateFactor',2)
        reluLayer()
        averagePooling2dLayer(poolSize,'Stride',2)
        crossChannelNormalizationLayer(3,'Alpha',5e-05,'Beta',0.75,'K',1)
        convolution2dLayer(filterSize,numFilters2,'Padding',2)
        reluLayer()
        averagePooling2dLayer(poolSize,'Stride',2)
        fullyConnectedLayer(10,'WeightL2Factor',250,...
        'BiasLearnRateFactor',2)
        softmaxLayer()
        classificationLayer()
        ];
end
end

%--------------------------------------------------------------------------
function memoryInBytes = getMemoryFootprint(net)
% Compute the total memory footprint of a trained network
```

```
layers = net.Layers;
numParameters = zeros(size(layers));

for idx = 1:numel(net.Layers)
    switch class(net.Layers(idx))
        case 'nnet.cnn.layer.ImageInputLayer'
            numParameters(idx) = numImageInputParameters(net.
Layers(idx));
        case 'nnet.cnn.layer.Convolution2DLayer'
            numParameters(idx) = num2dConvolutionParameters(net.
Layers(idx));
        case 'nnet.cnn.layer.FullyConnectedLayer'
            numParameters(idx) = numFullyConnectedParameters(net.
Layers(idx));
        otherwise
            numParameters(idx) = 0;
    end
end

% Each value will be 4 bytes in single precision
memoryInBytes = 4*sum(numParameters);
end

function numParameters = numImageInputParameters(layer)
if(layerUsesZeroCentering(layer))
    numParameters = prod(layer.InputSize);
else
    numParameters = 0;
end
end

function numParameters = num2dConvolutionParameters(layer)
```

```
numParameters = (prod(layer.FilterSize)*layer.NumChannels + 1) * layer.
NumFilters;
end

function numParameters = numFullyConnectedParameters(layer)
numParameters = (layer.InputSize + 1)*layer.OutputSize;
end

function tf = layerUsesZeroCentering(layer)
tf = strcmp(layer.Normalization,'zerocenter');
end
```

Appendix – Transfer Data to EBS Volume

If your application needs many small files, which is common in deep learning image recognition applications, the fastest way to access your data on the cluster is to attach and mount Amazon Elastic Block Store (EBS) volumes.

To get your data onto your cluster, use Cloud Center, the AWS console, and SSH to create and mount an EBS volume.

1. In Cloud Center, create a cluster and create a new SSH key for root access.
   See *mathworks.com/help/cloudcenter/ug/cloud-computing-console.html*
2. Use the Amazon Web Services (AWS) Console to create an Amazon EBS volume.
   See *https://aws.amazon.com/console/*
   a. In AWS, select EC2.
   b. In the top navigation bar, select the same region as your cluster (eg US-East, EU-West, etc.). You can only attach a volume to Amazon EC2 clusters from the same region.
   c. Under ELASTIC BLOCK STORE, choose Volumes, and click **Create Volume**.
   d. Choose a volume type. For example, the 100GB general purpose solid state drive (SSD gp2) is Amazon EBS's default volume type and is suitable for most applications. Amazon provides many options and the optimal choice depends on your application's read/write requirements.
   e. Select the **Availability Zone** that matches your Amazon EC2 instances. For example, eu-west-1b. You can check the zone of your cluster on the Instances pane in the EC2 console.

For help see *https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-creating-volume.html*

3. Use the Amazon Web Services (AWS) Console to attach your Amazon EBS volume.
   a. Select your created volume and chose **Actions > Attach Volume**.
   b. Enter the name or ID of your Amazon EC2 instance, or simply select it from the drop down menu.

c. Leave the default device name.

d. Click Attach. For help attaching volumes, refer to Amazon Web Service's instructions in *Attaching an Amazon EBS Volume*[13].

4. To make the volume available for the cluster to use, you must mount the volume.

On Windows:

a. Use Putty Key Generator to import and save your SSH root access key.

b. Use Putty to connect to your cluster using SSH. Your user name is ubuntu and the host name is the MATLAB Job Scheduler Host name, that you can look up in Cloud Center on the Cluster Summary. This is an example: **ubuntu@ec2-52-50-101-209.eu-west-1.compute.amazonaws.com**

For more help on connecting to the cluster machine from Windows, see *http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/putty.html.*

On Linux:

a. Set permissions on your SSH root access key.

```
[user~] % chmod 600 root_key.pem
```

b. Connect to your cluster using SSH, the root access key, and the MATLAB Job Scheduler Host name of your cluster, as shown in this example:

```
[user~] % ssh -i root_key.pem ubuntu@ec2-xx-yy-xxx-yyy.eu-west-x.compute.amazonaws.com
```

5. Create a mount point directory for your volume. This determines the location where you will read and write to your volume after it is mounted. Do not specify /mnt/matlab or /mnt/persisted, because Cloud Center creates these volumes. For example, we specified /mnt/datastore:

```
[ubuntu ~] % sudo mkdir -p -m a=rwx /mnt/datastore
```

6. Use the lsblk command to find the disk drive corresponding to your volume – look for the size you specified in AWS, in this example, xvdf.

```
lsubuntu@ip-172-30-2-71:~$ lsblk
NAME    MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
xvda    202:0    0   20G  0 disk
```

13. http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-attaching-volume.html

```
└─xvda1 202:1     0     20G   0 part /
xvdf     202:80    0    100G   0 disk
xvdm   202:192   0     15G   0 disk /mnt/matlab
xvdp    202:240   0    100G 0 disk /mnt/persisted
```

7. Set the file system of the new volume. Caution: this removes any filesystem or data already on the volume. For example:

```
[ubuntu ~] % sudo mkfs -t ext4 /dev/xvdf
```

8. Mount the volume to this location:

```
[ubuntu ~] % sudo mount /dev/xvdf /mnt/datastore
```

9. Upload your data to this volume. If your data is only available locally, transfer your data using standard utilities such as scp or sftp. For example:

```
scp -r -i  /home/.ssh/your-key.pem cifar10Train ubuntu@ec2-52-50-5-101-
  209.eu-west-1.compute.amazonaws.com:/mnt/datastore
```

This data set contains many small files and data transfer can take an hour. To make this faster, zip the data files before transferring.

For more help uploading data, see ***Transfer Data with Standard Utilities*** [14].

After you upload your data to your EBS volume, it can be useful to create a snapshot. This creates a backup copy of your data in Amazon S3, allowing you to save costs by deleting the EBS volume when not in use. When you want to use the data again, you can quickly and easily duplicate your data in new volumes. You can use the snapshot as a backup, a baseline for creating multiple new EBS volumes, to expand the size of a volume, or to move volumes across Availability Zones.

For more details about using volumes, see ***Amazon EBS Volumes*** [15]

14. http://www.mathworks.com/help/cloudcenter/ug/tips.html#bto1g9x-1
15. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSVolumes.html?icmpid=docs_ec2_console

Appendix - Prepare the CIFAR-10 data

This appendix shows you how to download and prepare the data used in this white paper.

Deep learning applications often use image files. The example in this paper uses image files to illustrate typical deep learning workflows, including using **imageDatastore** to efficiently access image files. **imageDatastore** is designed to read batches of images for faster processing in machine learning and computer vision applications. **imageDatastore** can import data from image collections that are too large to fit in memory.

The data that you can download for the example in this paper is not in the form of image files. Follow these steps to get the data and process it into image files. Then you can run the example in this paper, using typical image file workflows for deep learning.

1. Go to *https://www.cs.toronto.edu/~kriz/cifar.html* and download the MATLAB version of the CIFAR-10 data, **cifar-10-matlab.tar.gz**.
2. Extract the contents of **cifar-10-matlab.tar.gz**.
3. Run the script below in the folder where you extracted the contents of **cifar-10-matlab.tar.gz**. Specify the variable **outputPath** to set the location where you want to create the folders.

This script creates two folders **cifar10Train** and **cifar10Test** which you can use with the MATLAB script in this paper.

```
% Specify the path for saving the output data
outputPath = pwd;

% Specify output directories for training and test sets
trainDirectory = fullfile(outputPath,'cifar10Train');
testDirectory = fullfile(outputPath,'cifar10Test');

% Create directories for the output
mkdir(trainDirectory);
mkdir(testDirectory);

labelNames = {'airplane','automobile','bird','cat','deer','dog',...
    'frog','horse','ship','truck'};
makeDirectories(trainDirectory,labelNames);
makeDirectories(testDirectory,labelNames);
for i = 1:5
```

```
    fileToLoad = fullfile('cifar-10-batches-mat',...
        ['data _ batch _ ' num2str(i) '.mat']);
    saveImagesToFolders(fileToLoad,trainDirectory,labelNames,(i-1)*10000);
end
fileToLoad = fullfile('cifar-10-batches-mat','test _ batch.mat');
saveImagesToFolders(fileToLoad,testDirectory,labelNames,0);

%-------------------------------------------------------------------------
function makeDirectories(outputPath,directoryNames)
for i = 1:numel(directoryNames)
    mkdir(fullfile(outputPath,directoryNames{i}));
end
end

%-------------------------------------------------------------------------
function saveImagesToFolders(inputFilePath,outputPath,labelNames,nameOffs
et)
load(inputFilePath);
data = data'; %#ok<NODEF>
data = reshape(data, 32,32,3,[]);
data = permute(data, [2 1 3 4]);
for i = 1:size(data,4)
    outputFilename = fullfile(outputPath,labelNames{labels(i)+1},...
        ['image' num2str(i + nameOffset) '.png']);
    imwrite(data(:,:,:,i),outputFilename);
end
end
```