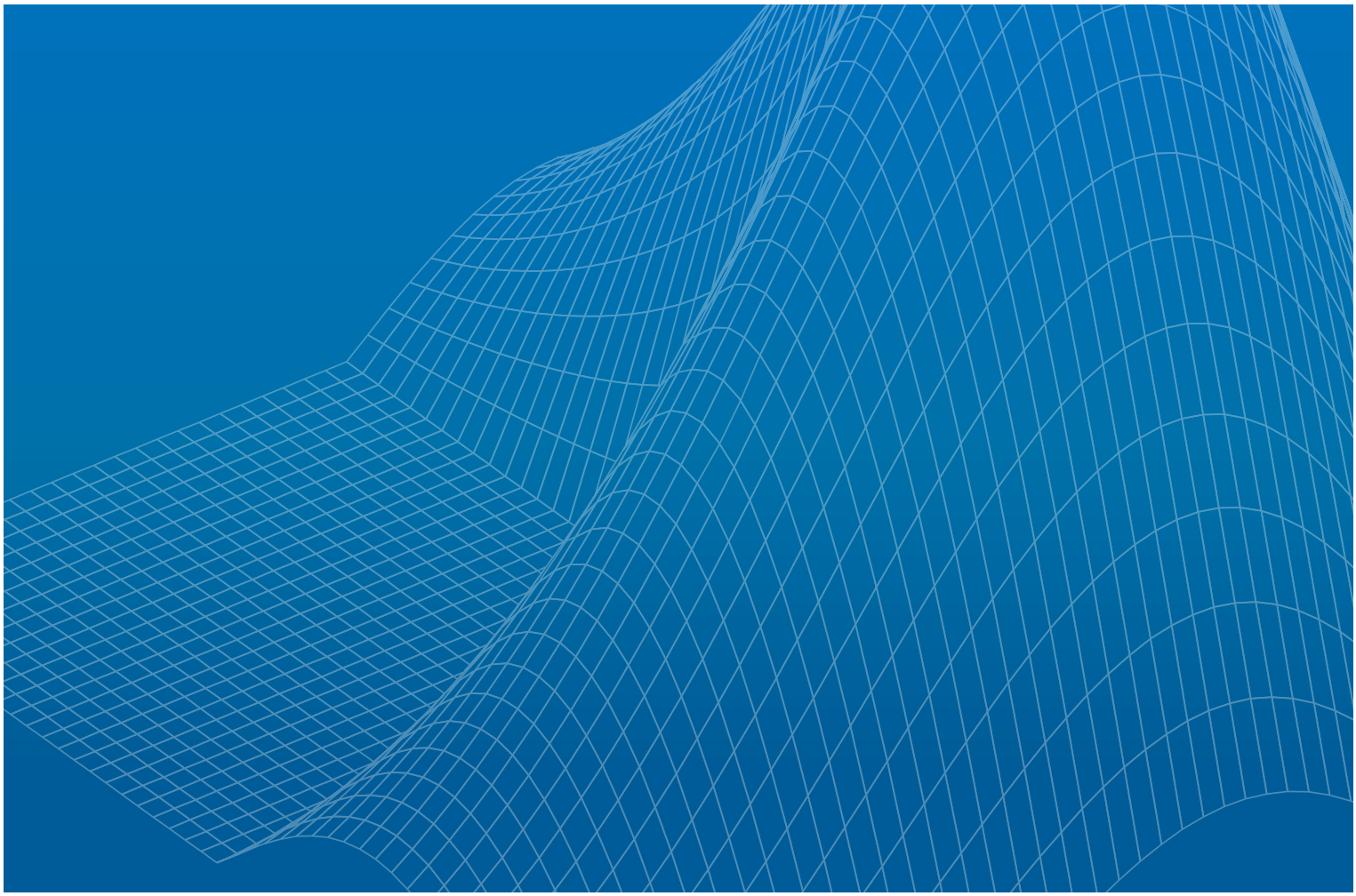

# Deep Learning with MATLAB and Multiple GPUs

By Stuart Moulder, Tish Sheridan, Pietro Cavallo, and Giuseppe Rossini

WHITE PAPER

MathWorks®

## Introduction

You can use MATLAB® to perform deep learning with multiple GPUs. Using multiple GPUs to train a single model provides greater memory and parallelism. These additional resources afford you larger networks and datasets; and for models which take hours or days to train, could save you time.

Deep learning is 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. P2 instances are high-specification multi-GPU machines. You can use deep learning on machines with a single GPU, and later scale up to 8 GPUs per machine to accelerate training, utilizing parallel computing to train a large, neural network with all of the processing power available.

Use the following sections to learn:

- How to train, test, and evaluate neural networks for deep learning problems in MATLAB
- How to scale up deep learning using high-performance multi-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 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 can 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 computationally intensive; you can usually accelerate training by using high-specification GPUs.

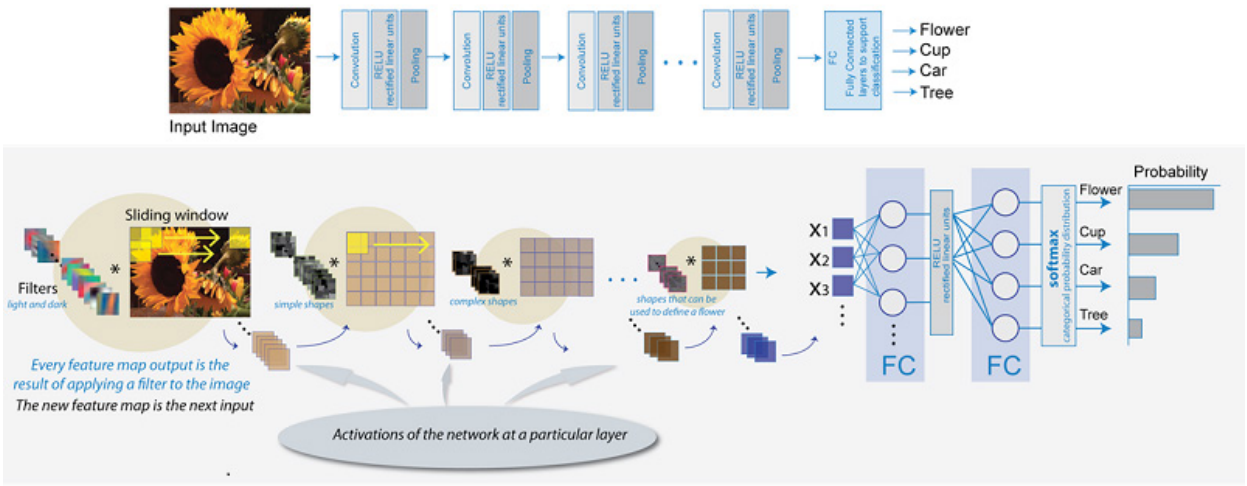


*Figure 1: Example of an image classification model.*

For this paper, we use a well-known existing network called AlexNet (refer to ***ImageNet Classification with Deep Convolutional Neural Networks***). AlexNet is a deep convolutional neural network (CNN), designed for image classification with 1000 possible categories. MATLAB has a built-in helper function to load a pre-trained AlexNet network:

```
% Load the AlexNet network
network = alexnet;
```

If the required package does not exist, you will be prompted to install it using the MATLAB Add-on Explorer. Inspect the layers of this network using the Layers property:

```
network.Layers
```

To learn more about any of the layers, refer to the ***Neural Network Toolbox™ documentation.***

The goal is to classify images into the correct class. For this paper, we created our own image dataset using images available under a Creative Commons license. The dataset contains 96,000 color images in 55 classes. Here we show five random images from the first five classes.

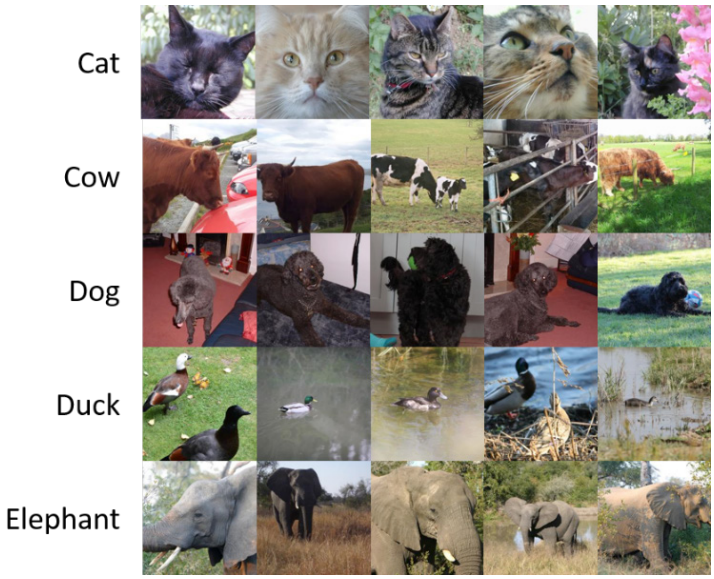


*Figure 2: Example classes and images from our image dataset.*

We resize and crop each image to 227x227 to match the size of the input layer of AlexNet.

## Transfer Learning

Training a large network, such as AlexNet, requires millions of images and several days of compute time. The original AlexNet was trained over several days on a subset of the ImageNet dataset, which consisted of over a million labelled images in 1000 categories (refer to ***ImageNet: A Large-Scale Hierarchical Image Database***). AlexNet has learned rich feature representations for a wide range of images. To quickly train the AlexNet network to classify our new dataset, we use a technique called transfer learning. Transfer learning utilizes the idea that the features a network learns when trained on one dataset are also useful for other similar datasets. You can fix the initial layers of a pre-trained network, and only fine-tune the last few layers to learn the specific features of the new dataset. Transfer learning usually results in faster training times than training a new CNN and enables use of a smaller dataset without overfitting.

The following code shows how to apply transfer learning to AlexNet to classify your own dataset.

1. Load the AlexNet network and replace the final few classification layers. To minimize changes to the feature layers in the rest of the network, increase the learning rate of the new fully-connected layer.

```
% Load the AlexNet network
networkOriginal = alexnet;
layersOriginal = networkOriginal.Layers;

% Copy all but the last 3 layers
layersTransfer = layersOriginal(1:end-3);

% Replace the fully connected layer with a higher learning rate
layersTransfer(end+1) = fullyConnectedLayer(55,...
    'WeightLearnRateFactor',10,...
    'BiasLearnRateFactor',20);

% Replace the softmax and classification layers
layersTransfer(end+1) = softmaxLayer();
layersTransfer(end+1) = classificationLayer();
```

2. Create the options for transfer learning. Compared to training a network from scratch, you can set a lower initial learning rate and train for fewer epochs.

```
% Define the transfer learning training options
optionsTransfer = trainingOptions('sgdm',...
    'MiniBatchSize',250,...
    'MaxEpochs',30,...
    'InitialLearnRate',0.00125,...
    'LearnRateDropFactor',0.1,...
    'LearnRateDropPeriod',20);
```

3. Supply the set of labelled 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('imageDataset/train',...
     'IncludeSubFolders',true,...
     'LabelSource','foldernames');
```

The dataset images are split into two sets: one for training, and a second for testing. The training set in this example is in a local folder called 'imageDataset/train'.

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

```
net = trainNetwork(imdsTrain,layersTransfer,optionsTransfer);
```

The result is a fully-trained network which can be used to classify your new dataset.

## 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 of images located in a local folder called 'imageDataset/test'. The accuracy score is the percentage of correctly classified images using the test set.

```
% Define the testing data
imdsTest = imageDatastore('imageDataset/test',...
     'IncludeSubfolders',true,...
     'LabelSource','foldernames');

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

## Training with Multiple GPUs

Cutting-edge neural networks rely on increasingly large training datasets and networks structures. In turn, this requires increased training times and memory resources. To support training such networks, MATLAB provides support for training a single network using multiple GPUs in parallel. Depending on your network and dataset, this can provide the following benefits.

## Increased GPU Memory

Convolutional neural networks are typically trained iteratively using batches of images. This is done because the whole dataset is far too big to fit into GPU memory. The optimal batch size depends on the exact network and dataset in question, so you need to experiment. Too large a batch size can lead to slow convergence, while too small a batch size can lead to no convergence at all. Often the batch size is dictated by the GPU memory available. For larger networks, the memory requirements per image increases and the maximum batch size is reduced.

When training with multiple GPUs, each image batch is distributed between the GPUs. This effectively increases the total GPU memory available, allowing larger batch sizes. Depending on your application, a larger batch size could provide better convergence or classification accuracy.

## Reduced Training Time

Using multiple GPUs can provide a significant improvement in performance. When deciding if you expect multi-GPU training to deliver a performance gain, consider the following factors:

- How long is the iteration on each GPU? If each GPU iteration is short, the added overhead of communication between GPUs can dominate. Try increasing the computation per iteration by using a larger batch size.
- Are you using more than 8 GPUs? Communication between more than 8 GPUs on a single machine introduces a significant communication delay.
- Are all the GPUs on a single machine? Communication between GPUs on different machines introduces a significant communication delay.

By default, the `trainNetwork` function uses a GPU (if available), otherwise the CPU is used. If you have more than one GPU on your local machine, enable multiple GPU training by setting the 'ExecutionEnvironment' option to 'multi-gpu' in your training options. As discussed above, you may also wish to increase the batch size and learning rate for better convergence and/or performance.

```
% Define the multi-gpu training options
optionsTransfer = trainingOptions('sgdm',...
      'MiniBatchSize',2000,...
      'MaxEpochs',30,...
      'InitialLearnRate',0.01,...
      'LearnRateDropFactor',0.1,...
      'LearnRateDropPeriod',20,...
      'ExecutionEnvironment','multi-gpu');
```

If you do not have multiple GPUs on your local machine, you can use Amazon EC2 to lease a multi-GPU cloud cluster.

## Scale Up to Deep Learning in the Cloud

Having performed transfer learning on one desktop computer, you now want to make use of a high-specification multi-GPU machine. Amazon can provide suitable machines on demand, using their new P2 instances. The new *Amazon EC2 P2* instances are machines specifically designed for compute-intensive applications, providing up to 16 NVIDIA Tesla K80 GPUs per machine. In the following sections, you can learn how to reserve a P2 instance, connect to the data, and train a model in parallel using multiple GPUs in the cloud.

To use deep learning in the cloud, you need:

- MATLAB, Neural Network Toolbox, Parallel Computing Toolbox™
- A MathWorks account
- Access to *MATLAB Distributed Computing Server™ for Amazon EC2*
- 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 in the cloud. Amazon EC2 is ideally suited for 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. 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 train larger models in less time. 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, refer to *MATLAB Distributed Computing Server for Amazon EC2*. To set up your credentials and create a new cluster, refer to *Create and Manage Clusters* in the Cloud Center documentation.

In the following example, we create a cluster of a single machine with 8 GPUs and 8 workers. Setting the number of workers equal to the number of GPUs ensures there is no competition between workers for GPU resources. The main steps are:

1. Log in to *Cloud Center* using your MathWorks 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, refer to the Cloud Center documentation: *Set Up Your Amazon Web Services (AWS) Credentials.*
3. To create a cluster of Amazon EC2 instances, click **Create a Cluster**.
4. Complete the following steps, and then ensure your settings look similar to the screenshot below.
    a. Name the cluster
    b. Choose an appropriate **Region**
    c. Select **Machine Type: Double Precision GPU (p2.8xlarge, 16 core, 8 GPU)**
    d. For **Number of Workers**, select **8**
    e. Leave the other settings as defaults and click **Create Cluster**

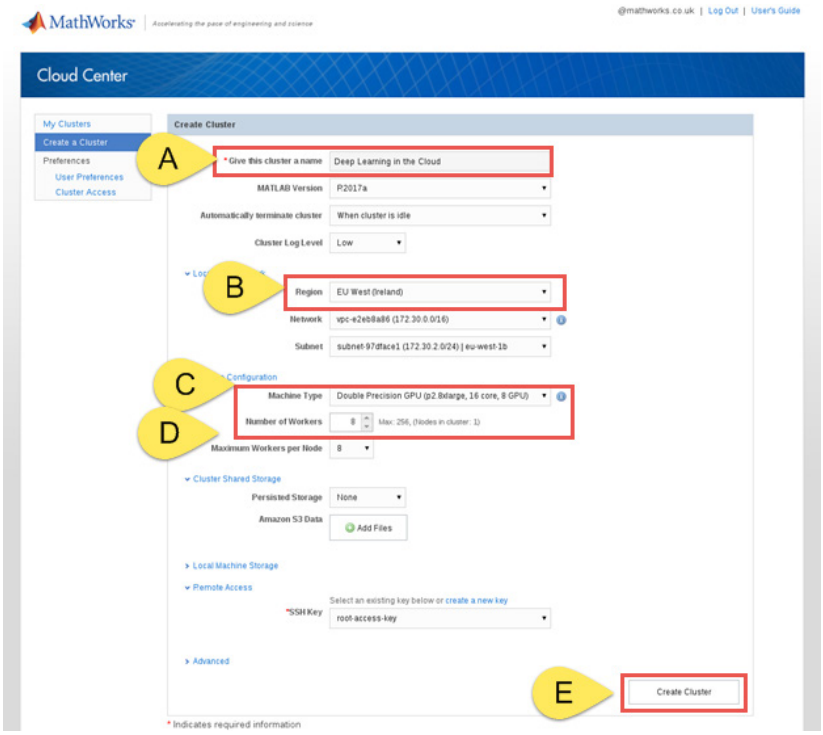


*Figure 3: Cloud Center: Create a cluster with multiple GPUs.*

5. To access your cluster from MATLAB, on the home tab select **Parallel > Discover Clusters** to search for your Amazon EC2 clusters. When you select the cluster, the wizard automatically sets it as your default cluster.

Confirm your cluster is online: either from Cloud Center, or from within MATLAB by creating a cluster instance and displaying the 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 by clicking **Start Up** in Cloud Center, or typing `start(cluster);` in MATLAB:

```
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, refer to the ***Parallel Pools documentation***.

## Training with a GPU Cluster

To enable training on a compute cluster, set the 'ExecutionEnvironment' option to 'parallel' in your training options.

```
% Define the parallel training options
optionsTransfer = trainingOptions('sgdm',...
      'MiniBatchSize',2000,...
      'MaxEpochs',30,...
      'InitialLearnRate',0.01,...
      'LearnRateDropFactor',0.1,...
      'LearnRateDropPeriod',20,...
      'ExecutionEnvironment','parallel');
```

If no pool is open, **`trainNetwork`** will automatically open one using the default cluster profile. If the pool has access to GPUs, then they will be used.

When training on a cluster, the location passed to **`imageDatastore`** must exist on both the client MATLAB and on the worker MATLAB. For this reason, we store our data in an Amazon S3 bucket. An S3 location is referenced by a URL, hence the path is visible on all machines. For more information on using data in S3, refer to Use Data in Amazon S3 Storage.

## Use Data in Amazon S3 Storage

Amazon Simple Storage Service (S3) provides secure and scalable storage in the cloud. Once data is uploaded to Amazon S3, it can be accessed from anywhere, making it ideally suited for distributing machine learning datasets to cloud clusters. We uploaded our training and test images stored locally in the 'imageDataset' folder to an S3 bucket of the same name. More information about uploading and accessing data using S3 can be found in the ***Amazon S3 documentation.***

By default, objects stored in Amazon S3 are private and can only be accessed by their owner. To access or modify these resources, the client must first have the correct authentication tokens. To generate these tokens, the resource owner can use the AWS Management Console to create an **Access Key ID** and corresponding **Secret Access Key**. Clients who they share tokens with will then have programmatic access to their data stored in Amazon S3. Further details about generating access keys can be found in the ***Amazon Access Keys documentation.***

The following example demonstrates how you would configure your AWS authentication tokens as environment variables. Enter in your local MATLAB:

```
% Set AWS credentials as environment variables on local client MATLAB
setenv('AWS_ACCESS_KEY_ID', 'AKIAIOSFODNN7EXAMPLE');
setenv('AWS_SECRET_ACCESS_KEY', ...
'wJalrXUtnFEMI/K7MDENG/nPxRiCYEXAMPLEKEY');
```

To access training data, you now simply create an image datastore pointing to the URL of the S3 bucket.

```
% Define the Amazon S3 training data
imdsTrain = imageDatastore('s3://imageDataset/train',...
    'IncludeSubfolders', true, ...
    'LabelSource', 'foldernames');
```

To access S3 resources from a cluster, you must set the same environment variables on your workers. The following code manually opens a parallel pool on your cloud cluster and copies the necessary environment variables to the workers.

```
% Set AWS credentials on all workers
aws_access_key_id = getenv('AWS_ACCESS_KEY_ID');
aws_secret_access_key_id = getenv('AWS_SECRET_ACCESS_KEY');
spmd
    setenv('AWS_ACCESS_KEY_ID',aws_access_key_id);
    setenv('AWS_SECRET_ACCESS_KEY',aws_secret_access_key_id);
end
```

The workers now have access to the Amazon S3 resources for the lifetime of the parallel pool.

## Results

Having used Mathworks Cloud Center and Amazon EC2 to lease a multi-GPU compute cluster, we would now like to investigate the performance benefits which this affords. The following plot shows the classification accuracy of our network as a function of training time. The classification accuracy is defined as the accuracy achieved by the network on the mini batch of images for the current training iteration. The training was repeated four times, each using a different number of GPUs to train the network in parallel. As discussed in *Increased GPU Memory*, training across more GPUs permits

a larger image batch size. We therefore scale the image batch size with the number of GPUs to fully utilize the available memory. To normalize the learning rate per epoch, we also scale the learning rate with the image batch size, because a larger batch size provides fewer iterations per epoch.

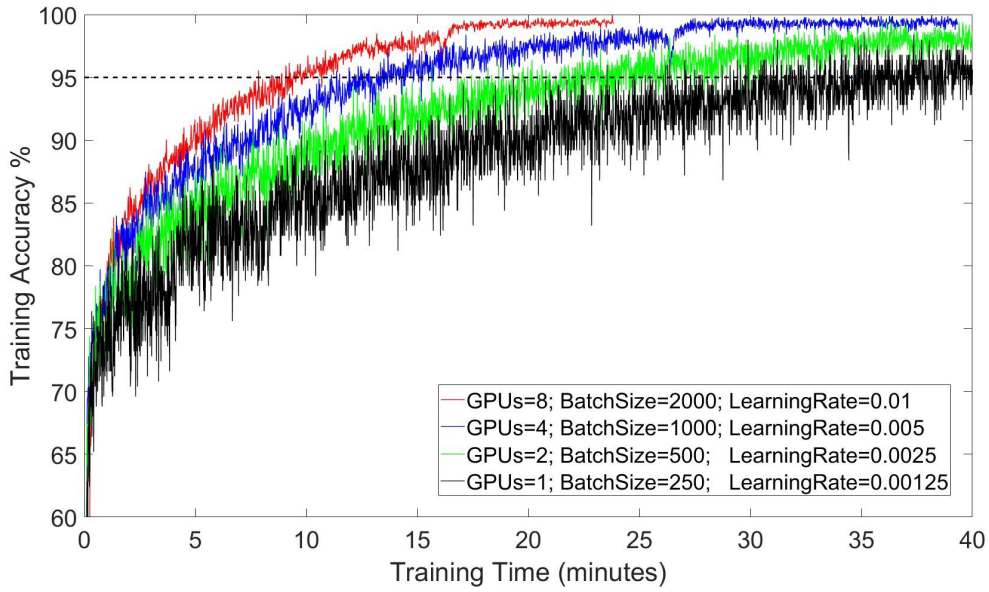


*Figure 4: Training convergence with varying numbers of GPUs.*

These results show that the additional memory and parallelism of more GPUs can provide significantly faster training. Smoothing out the data, we find that the time taken to reach a classification accuracy of 95% decreases by approximately 30-40% with each doubling of the number of GPUs used to train. This reduces the training time to around 10 minutes when using 8 GPUs, compared to more than 40 minutes with a single GPU.

To further quantify these results, we plot the average image throughput, to normalize for the different batch sizes. This shows an increase in the number of images processed per unit time of approximately 70% with each doubling of the number of GPUs.

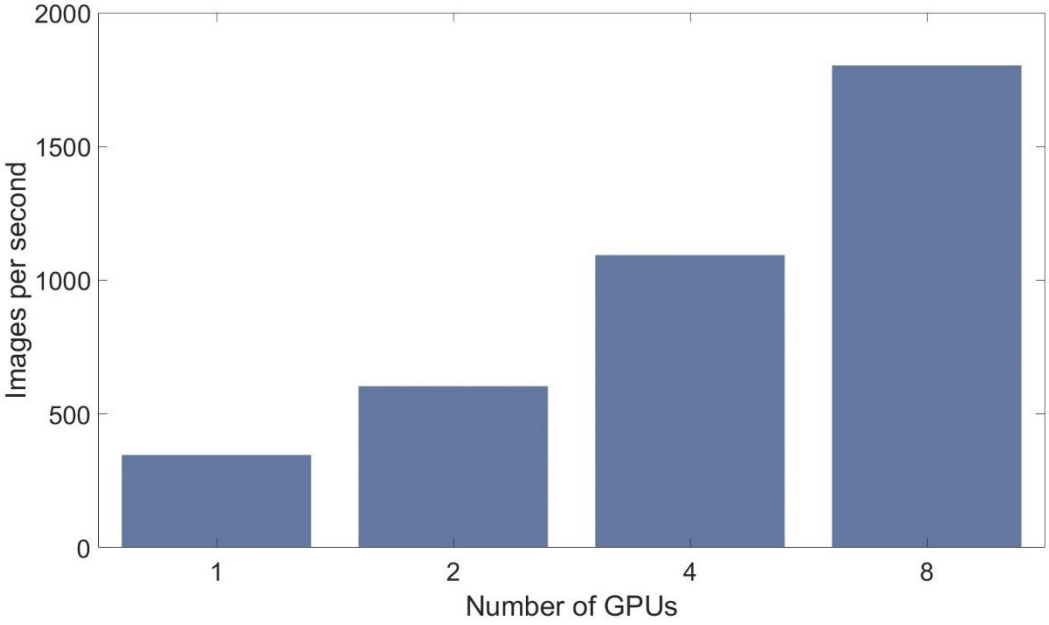


*Figure 5: Image throughput with varying numbers of GPUs.*

Using a separate dataset of test images, we measure the classification accuracy of our networks on unseen images. The four networks trained with different numbers of GPUs all achieve an accuracy of 87%, to within half a percent.

To see the script for training AlexNet with your own data using a multi-GPU cluster, see ***Appendix – MATLAB code***.

**Note:** Creating a cluster and parallel pool in the cloud can take up to 10 minutes. For the small problem analyzed in this paper, this was a significant proportion of the total time. For other problems where the total training time can take hours or days, this cluster startup time becomes negligible.

## Conclusions

In this paper, we show how you can use MATLAB to train a large, deep neural network. The code provides a worked example; demonstrating how to train a network to classify images, use it to classify a new set of images, and measure the accuracy of the classification. Using Mathworks Cloud Center with Amazon EC2 to lease a multi-GPU compute cluster, we demonstrate a significant reduction in training time by distributing the work across multiple GPUs in parallel. For large datasets, such a performance increase can save hours or days.

## Useful links:

For more information, see the following resources:

- *https://www.mathworks.com/discovery/deep-learning.html*

Central resource for Deep Learning with MATLAB

- *https://www.mathworks.com/help/nnet/convolutional-neural-networks.html*

Neural Network Toolbox documentation on essential tools for deep learning

- *https://www.mathworks.com/products/parallel-computing/parallel-computing-on-the-cloud/*
- *https://aws.amazon.com/console/*

## Appendix – MATLAB Code

The following MATLAB code uses transfer learning to train AlexNet on a new image dataset stored in Amazon S3 using a cluster. To run this script, use your own Amazon S3 bucket and set the appropriate environment variables for your Amazon Access key.

```
% Number of workers to train with. Set this number equal to the number of
% GPUs on you cluster. If you specify more workers than GPUs, the remaining
% workers will be idle.
numberOfWorkers = 8;

% Scale batch size with expected number of GPUs
miniBatchSize = 250 * numberOfWorkers;

% Scale learning rate with batch size
learningRate = 0.00125 * numberOfWorkers;

%% Load the AlexNet network
networkOriginal = alexnet;
layersOriginal = networkOriginal.Layers;

% Copy all but the last 3 layers
layersTransfer = layersOriginal(1:end-3);

% Replace the fully connected layer with a higher learning rate
```

```
% The output size should be equal to the number of labels in your
dataset.
layersTransfer(end+1) = fullyConnectedLayer(55,...
    'WeightLearnRateFactor',10,...
    'BiasLearnRateFactor',20);

% Replace the softmax and classification layers
layersTransfer(end+1) = softmaxLayer();
layersTransfer(end+1) = classificationLayer();

%% Start a parallel pool if one is not already open
pool = gcp('nocreate');
if isempty(pool)
    parpool(numberOfWorkers);
elseif (pool.NumWorkers ~= numberOfWorkers)
    delete(pool);
    parpool(numberOfWorkers);
end

%% Copy local AWS credentials to all workers
aws_access_key_id = getenv('AWS_ACCESS_KEY_ID');
aws_secret_access_key_id = getenv('AWS_SECRET_ACCESS_KEY');
spmd
    setenv('AWS_ACCESS_KEY_ID',aws_access_key_id);
    setenv('AWS_SECRET_ACCESS_KEY',aws_secret_access_key_id);
end

%% Load the training and test data
imds = imageDatastore('s3://imageDataset',...
    'IncludeSubfolders',true, ...
    'LabelSource','foldernames');
```

```
%% Shuffle and split data into training and testing
[imdsTrain,imdsTest] = splitEachLabel(shuffle(imds),0.9);

%% Define the transfer learning training options
optionsTransfer = trainingOptions('sgdm',...
    'MiniBatchSize',miniBatchSize,...
    'MaxEpochs',30,...
    'InitialLearnRate',learningRate,...
    'LearnRateDropFactor',0.1,...
    'LearnRateDropPeriod',20,...
    'Verbose',true,...
    'ExecutionEnvironment','parallel');

%% Train the network on the cluster
net = trainNetwork(imdsTrain,layersTransfer,optionsTransfer);

%% Record the accuracy for this network
% Uses the trained network to classify the test images on the local machine
% and compares this to their ground truth labels.
YTest = classify(net,imdsTest);
accuracy = sum(YTest == imdsTest.Labels)/numel(imdsTest.Labels);
```