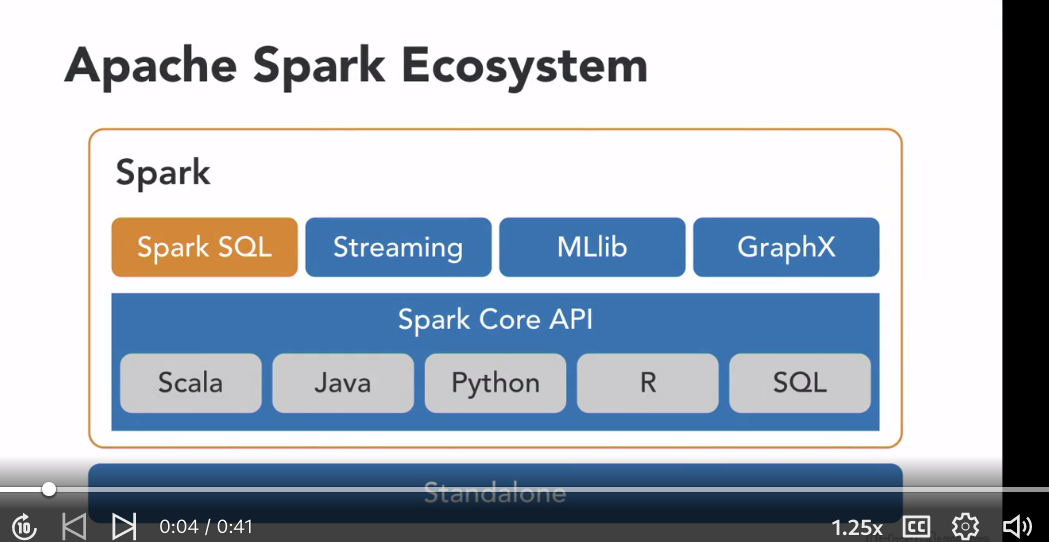
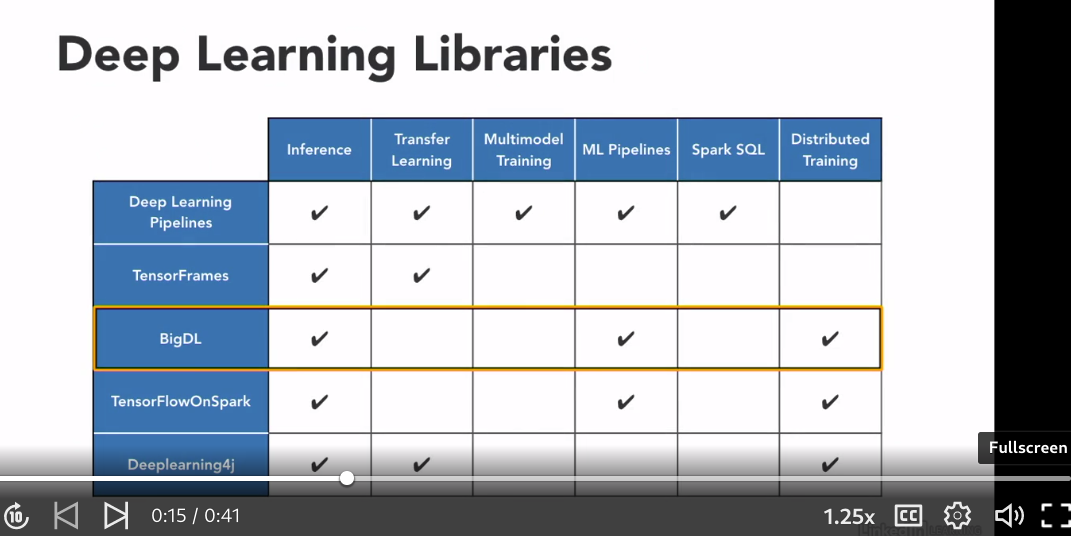
# **Apache Spark Deep Learning Essential Training**

### **Apache Spark**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Jonathan] Over the last couple of years, there have been two growing trends. Firstly, deep learning has proved to be one of the fastest growing areas within artificial intelligence, and second, around the same timeframe, Apache Spark has evolved into being the big data platform of choice. Now, if you want to do deep learning at scale, then Apache Spark offers a safe pair of hands. The deep learning package we'll use has an easy-to-use API, it has great support for images in Spark, and you'll be running deep learning algorithms with only a couple of lines of code. Hi, I'm Jonathan Fernandes and work in machine learning and AI for a consultancy. Join me on LinkedIn Learning as we explore deep learning on Spark using Python.





### **What you should know before watching this course**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] There are a couple of deep learning libraries available for Apache Spark. In this course we'll be using the one developed by Databricks and there's a good reason for this. The guys who created Spark and many of the Spark code developers also work for Databricks. This often means cutting edge technology from Databricks finds its way into Apache Spark's main code. We're going to be using Databricks cloud environment to run our notebooks. If you haven't used notebooks before, I'll show you how to get started. You're also welcome to install your version of Spark locally and run the Exercise Files from there. We'll also cover the main concepts we need for deep learning so don't worry if you don't have a lot of experience with it. It would be helpful to have some experience with PySpark so that we can quickly get started using the deep learning library. If you don't have any experience with PySpark, then check out my other course Apache PySpark by Example in the LinkedIn Library as that will help you get started with the basics. The aim of this course is to help you get started using one of the deep learning libraries available for Apache Spark.

### **Setting up a Databricks account**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Let's head over to databricks.com to set up our account. Databricks, very generously, allows you to create a Community Edition account free of charge. This is great to help students get started. So let's select Try Databricks and let's go to the Community Edition and select Get Started. And you need to enter your details here. Select what's your intended use case. So I'm going to select Personal - Taking an Online Course. And how would you describe your role? So select whatever's relevant to you. I'm going to uncheck the Keep Me Informed and I select Sign Up. After verifying your email address, you'll be redirected to enter a password. So I've done that. I'm now going to head over to the Databricks website to sign in and then I need to look for the Community Edition here so I then select Sign In Here. And then enter my email address and my password. And I've been able to log into my Databricks account.

## **Question 1 of 3**

Paula, a college junior on a budget, is looking into the cost of using Databricks. How much will it cost Paula to create a Databricks Community Edition account?

* The Community Edition of Databricks is free for the first month, then $7.99 per month, which makes it ideal for exploring on a trial basis.
* The Community Edition of Databricks is $5.99 per month, which makes it ideal for students who are on a tight budget.
* The Community Edition of Databricks is free, which makes it ideal for students just getting started with Apache Spark.  
  Correct
* The Community Edition of Databricks is $50 per year, which makes it ideal for students who are on a tight budget.

## **Question 2 of 3**

Why is it beneficial to use Databricks as your deep learning library?

* Technology from Databricks is often incorporated in Apache Spark's main code.  
  Correct
* Databricks is the newest deep learning library, and uses a new form of code.
* Databricks is the least expensive deep learning library on the market.
* Databricks supports all types of code, which allows for common use among teams at work.

## **Question 3 of 3**

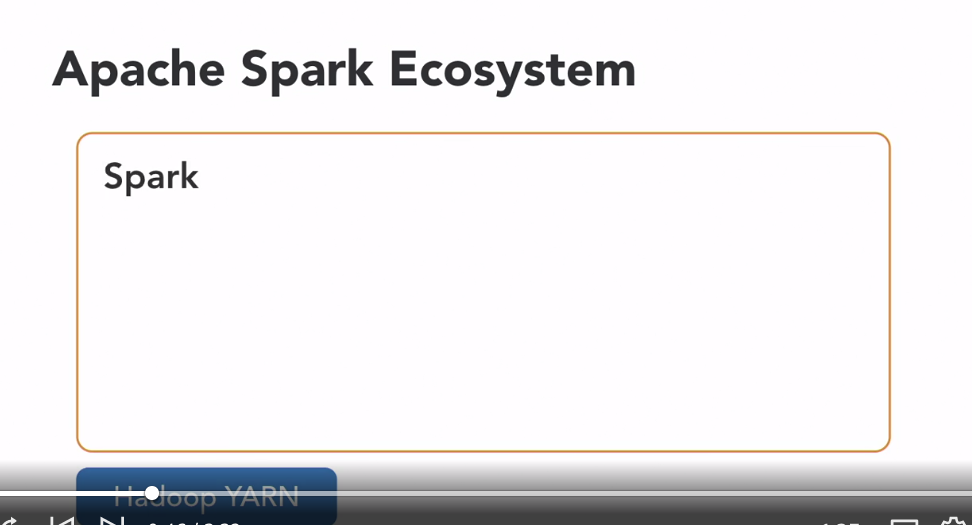
Deep learning in Apache Spark allows you to run deep learning algorithms with \_\_\_\_\_.

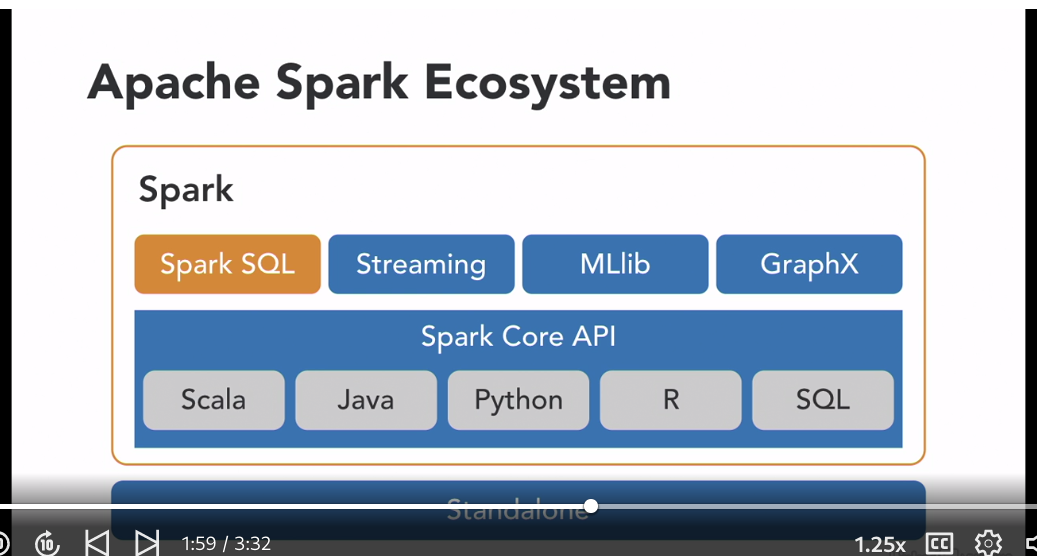
* only a couple lines of code  
  Correct
* only one line of code
* several short pieces of code
* several long lines of code

### **Apache Spark ecosystem**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] If you head over to the Apache Spark website, you can see some of the reasons that people are using it. It's fast, it's easy to develop in because it has a couple of language APIs, meaning that you can access it using Scala, Java, Python, R or SQL, and it has its own ecosystem. Just so you know, Apache Spark is written in Scala. Now the developers of Apache Spark wanted it to be fast to work on their large data sets. But when they were working on their big data projects, many of the scripting languages didn't fit the bill. Because Spark runs on a Java virtual machine, or JVM, it lets you call in to other Java based dictator systems, such as Cassandra, HTFS and HBase. Spark runs locally, as well as in clusters, on Premise or in the Cloud. It runs on top of Hadoop Yarn, Apache Mesos Standalone. Although Spark is designed to work on large distributed clusters, you can also use it on a single Machine Mode. The means of accessing these different modules is via the Spark Core API, which is the foundation of the Apache Spark Ecosystem. The Apache Spark Ecosystem consists of Spark SQL and DataFrames, Streaming, which is for real time data, MLlib, for machine learning and GraphX, which lets you make sense of graph structure data, at scale. We take a quick look at each of the different components to get a better understanding of them. Let's take a look at the Spark Core, which is at the bottom of the image. The Spark Core contains the basic functionality of Spark, including components for Task scheduling, Memory management, Fault recovery, Interacting with storage systems, and more. Remember that Spark is primarily written in Scala, so this is the default language. So Spark SQL allows you to interactively explore the data using SQL queries. Now, if you're familiar with DataFrames, either from Pandas or R, you know how easy they are to use. This is where I think the Spark developers were really smart. By creating this DataFrame abstraction, it meant that data analysts, and data scientists, who are familiar with these from Pandas and R, could get up and running with Apache Spark really quickly. Spark SQL allows you to intermix SQL queries with the programmatic data manipulations, supported by lower level APIs in Python, Java and Scala. This means that within a single application you can combine SQL with complex analytics. Now let's look at Streaming. Many applications need the ability to process, and analyze, not only batch data, but also streams of new data in real time. Running on top of Spark, Spark Streaming enables powerful interactive, and analytical, applications across both streaming and historical data. The real bonus is the fact that you can use virtually the same code, that you created for batch data, to process real time data. The MLlib machine learning model, built on top of Spark, is a scalable machine learning library that delivers high quality algorithms quickly. The last component of the Ecosystem we look at is Graph Computation. GraphX is a graph computation engine, built on top of Spark, that allows users to interactively build, and transform, graph structured data at scale. It comes complete with a library of common algorithms. Now, GraphX uses a directed multi graph with properties attached to each vertex and edge. They're really helpful for things like social networks, which you can model as a graph and you can determine relationships between different nodes or vertices. So, as you can see, Spark has a pretty impressive, and self sufficient, Ecosystem.





### **The origins of Spark and Databricks**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Spark started in 2009 as a research project in the UC Berkeley RAD Lab. The researchers in the lab had been previously working on Hadoop MapReduce and observed that MapReduce was inefficient for iterative and interactive computing jobs. So, right from the beginning Spark was designed to be fast for interactive queries and iterative algorithms. It brought in ideas like support for in-memory storage and efficient fault recovery. Research papers were published about Spark at academic conferences and soon after its creation it was already 10 to 20 times faster than MapReduce for certain jobs. In Matei's, 2009 paper they say that while Spark is still currently a working prototype the performance results they were getting were very encouraging. Even at that time Spark could outperform machine learning workloads by a factor of 10 and you can see this on page five of their paper. As part of their experiments into Sparks performance, they performed a logistic regression job across 20 nodes with each node having four cores. It then crashed a node to demonstrate that Spark could continue to function with the fewer node. They compared this with a Hadoop implementation. Now, remember that Hadoop was the big data solution of choice in 2009. Each Hadoop iteration would take over two minutes, in the case of Spark the first iteration took almost three minutes but because the data is cached further iterations only took six seconds compared to Hadoop's two minutes. So, what happened to Spark after 2009? Well, it was first open sourced in March 2010 and then after 2011 further libraries were added such as MLlib, Spark Streaming and GraphX, by 2013 Spark was enjoying widespread use and Berkeley's AMPLab contributed Spark to the Apache Software Foundation. A couple from the original AMPLab team then started a company called Databricks, it's now one of several companies that contributes to Apache Spark code. So, what's different with Databricks version of Apache Spark? Databricks offers Apache Spark with a platform that unifies the data science and data engineering. It provides an optimized version of Apache Spark offering interactive notebooks and provides full enterprise security that any large organization would need. You can also connect to other business intelligence tools using Databricks and because they're one of the main contributors you can be pretty confident that anything that is created by the Databricks developers will later end up in Apache Spark main code base. If you look on the Databricks website you can see that a couple of the guys who wrote the original paper, such as Zaharia and Stoica are still with Databricks. In this course, we'll be using Databricks Deep Learning Pipeline package for deep learning.

## **Question 1 of 2**

Why was Spark first created?

* to slow down the speed of interactive queries and algorithms for users
* to use as an antihacking tool to protect known algorithms
* to use as spyware to steal interactive algorithms, and then use them maliciously
* for the fast speed of interactive queries and algorithms  
  Correct

## **Question 2 of 2**

How are different modules accessed within Apache Spark?

* through the Spark SQL and Dataframes, which are the foundation of the Apache Spark Ecosystem
* through Streaming, which allows applications to process, analyze, and stream new data in real time
* through the pipelines, which allow applications to process, analyze, and pipe in new data in real time
* through the Spark Core API, which is the foundation of the Apache Spark Ecosystem  
  Correct

### **What is deep learning?**

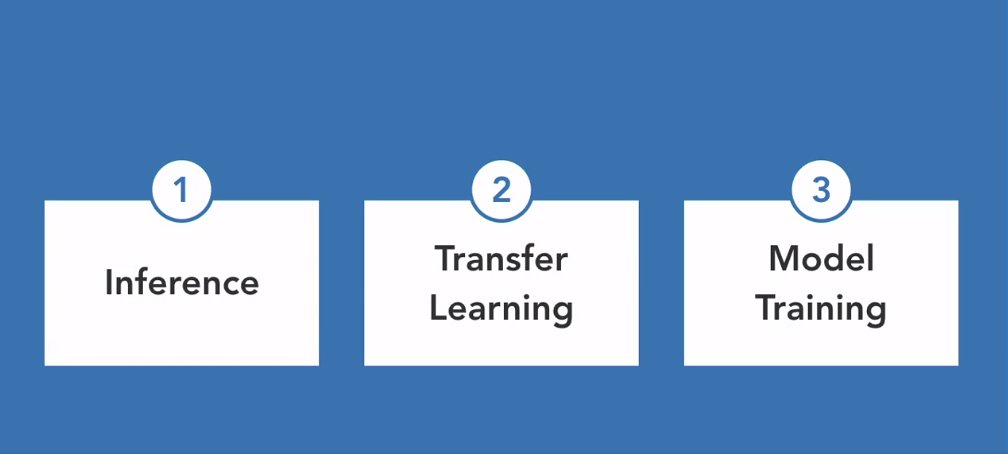
Selecting transcript lines in this section will navigate to timestamp in the video

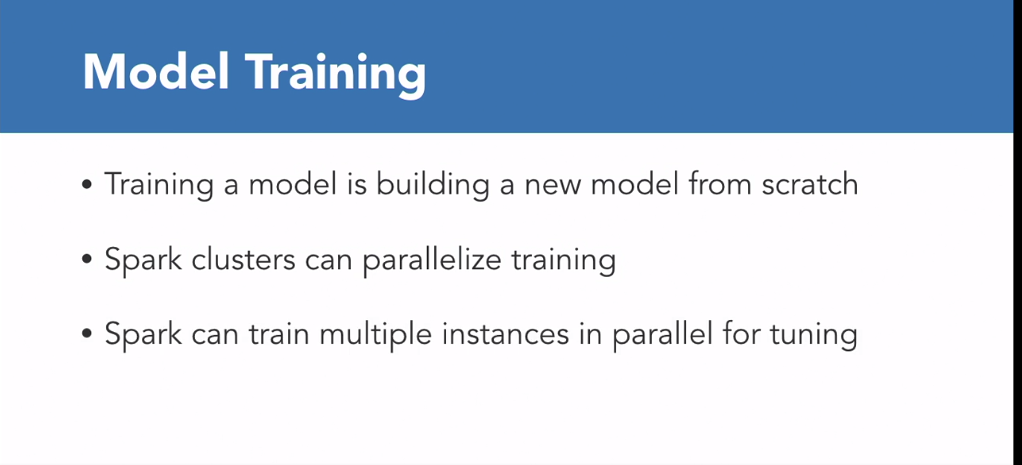
- [Instructor] As we mentioned earlier, the OpenCV code is originally written in C++. When we use OpenCV in Python, what's happening under the hood is that we have a Python wrapper around the original C++ code. We also use NumPy, which is a highly optimized Python library for numerical operations. All the OpenCV area structures are converted to and from NumPy arrays. Here are a couple of things NumPy contains. A powerful N-dimensional array object, sophisticated broadcasting functions, and tools for integrating C, and C++, and Fortran code, and this is very helpful, because it makes processing for images faster. It's also useful for linear algebra, Fourier Transforms, and random number capabilities. NumPy's main object is the multidimensional array. It's a table of elements, usually numbers, indexed by a tuple of positive integers. Every NumPy array is a grid of elements of the same type. NumPy provides a large set of numeric data types that you can use to construct arrays. Let's take a look at what's involved in images. A grayscale camera records the intensity of reflected light. A color camera records the light reflected at three points of the visible spectrum, red, green, and blue. Color is the way the human eye perceives a mixture of lights of different wavelengths. The RGB color model gets its name from the three primary colors, red, green, and blue. Now, what's a little confusing is that in OpenCV we use the BGR format, so that's blue, green, and red, rather than RGB. The reason for this is when OpenCV first started off, the BGR color format was popular among camera manufacturers and software providers. Let's take a look at an image that we will read in in OpenCV. Now, within the exercise files for this course are a couple of images. We're going to use the file called devon in the image folder. So I select devon. Now, this is a photo of a place in Devon in England that I took when on holiday. So, let's head over to the command line and activate our virtual environment. So, I type cmd, 'cause I'm going to head over to the exercise files which are on my desktop, and I'm going to go to Chapter 02\_01. And I'm going to activate my virtual environment, so activate ocv4, and wihtin this virtual environment I'm going to open the atom editor, so atom, and I'm going to open a file called image.py. So, let's import NumPy and CV2, so import NumPy as np, and import CV2. Now, the function we'll use to read in images is imread for images read, so, cv2.imread, and the path to the devon file is in the folder images, devon.jpg, and I'm just going to confirm that that's the name of the file, so I go to CD, images, and I do a dir, and I can see that it's devon.jpg, and I head back into the 02\_01 file and I clear my screen. Now, I'm going to give this the name image, so img equals cv2.imread, and that's the path to my file. And now if I do a print of the type of img, and save this file. So, File, and Save. So now let's run this on our command line. So, python. You can see that img is of type NumPy array. So let's delete that line, and one of the frustrating parts of this function is that even if you have the wrong path, it won't throw an error. So let me just modify the name of that file from devon.jpg to, let's just call it devo, and I'll remove the n. And I run that file again. You can see that there's absolutely no error thrown there. So what I normally do when using the imread function is to also print out the shape of the image. That way, I can confirm that an image has actually been read. So I'm going to add the n to the end of that file, and I'm going to print out the shape of this file. So, img.shape. Save this file, Ctrl+S and then run it from the command line. And from this, we know that the image has 360 rows, 480 columns, and it has three channels. These three channels correspond to B, G, and R. I'm going to just comment out this line. So if we want to display this image, I just type cv2 dot image show, give it the name Image, and then I want to display what's in the NumPy array img. As I had cv2.waitKey zero, and cv2.destroyAllWindows. We use the waitKey with argument zero, because this means execution is paused until a key is pressed. If we didn't have a waitKey, then you would see the image briefly, and it would disappear from your screen. So for example, if you pass an argument like 3000 to waitKey, then it waits for at least 3000 milliseconds for a keystroke before the window is closed. The cv2.destroyAllWindows simply destroys all windows that we've created. So let's save this file, and let's re-run our Python file. And you can see that we're able to display the image. And so when we press any key, this will close the window.

### **Using deep learning in Spark**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] There are three major ways to do deep learning in Spark. They are inference, transfer learning, and model training. Let's look at each of them in turn in a little more detail. We could take a model like ResNet 50, or Inception Version Three, or VGG16, these are deep learning models that have been trained on ImageNet dataset, and apply them to a large dataset in parallel using Spark. This will allow you to apply image classification to a large dataset of images very quickly and put them into one of the 1000 classes. You could take the YOLO Version Three model, for example, which is used for object detection, and apply it in parallel using a Spark function. You could also use PySpark's map function to get a distributed inference by calling TensorFlow, Keras, or PyTorch. The second way to use deep learning in Spark is via transfer learning. Transfer learning is using a trained neural network that would have been trained on a dataset that is similar to the one you're working on. So, for example, if you're looking to do some image classification, then a model that has been trained on ImageNet would be a good start. Transfer learning is particularly helpful when you don't have a large amount of training data. To give you an idea of the scale involved, when training a model for ImageNet, you require datasets with hundreds of thousands of images. You can get good results for transfer learning with only a few thousand images. The third option is where Spark is used to train a new deep learning model from scratch. There are two common approaches available. You can either use a Spark cluster to parallelize the training of a single model over multiple servers communicating updates between them, or you could train multiple instances of similar models in parallel to try very small architectures and hyper parameters. This parallel processing helps you tune your model faster.

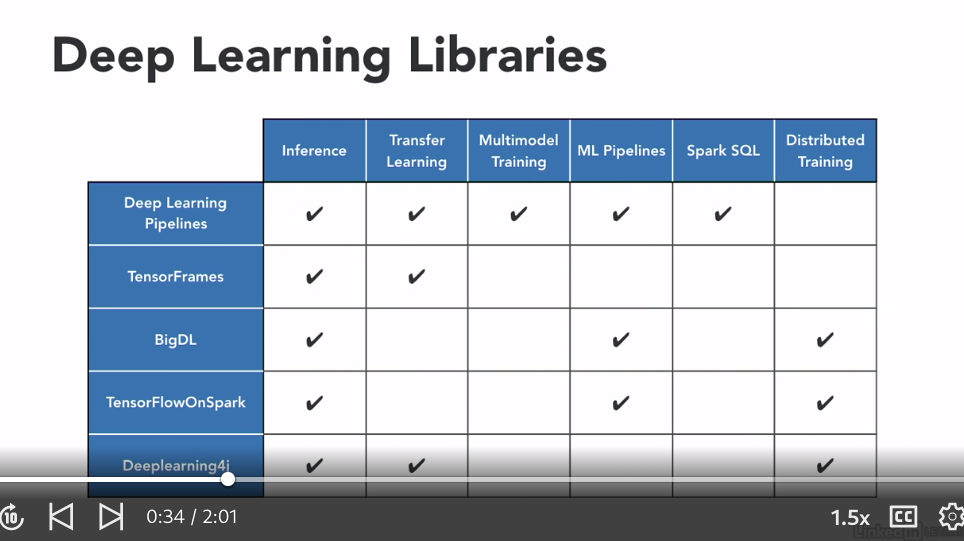




### **Deep learning libraries**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Now there are a couple of deep learning libraries available out there for Spark. In this course, we'll only look at Deep Learning Pipelines. This is a package developed by Databricks that integrates deep learning functionality into Spark's ML Pipeline's API. It currently also supports TensorFlow and Keras. It's used for inference, transfer learning, model training, and it integrates with Spark's SQL, and ticks most of the boxes. Let's take a look at some of the other deep learning frameworks available**. TensorFrames or TensorFlowOnSpark dataframes lets you manipulate Spark's dataframes with TensorFlow programs. TensorFrames main contribution is making it easy to pass data between Spark dataframes and TensorFlow. It's mainly an inference and transfer learning oriented library and supports both Python and Scala. BigDL is a distributed deep learning framework for Apache Spark, primarily developed by Intel and is modeled after Torch. Now one of the key advantages of BigDL over the other libraries is that it's optimized to use CPUs instead of GPUs. This means it is efficient to run on existing CPU based clusters, like an Apache Hadoop environment. TensorFlowOnSpark was developed by Yahoo for large scale distributed deep learning on Hadoop clusters in Yahoo's private cloud. It enables distributed deep learning on a cluster of GPUs and CPU servers.** It enables both distributed TensorFlow training and inferencing on Spark clusters and what's great is that the amount of change that you need to make to code to run existing TensorFlow programs is minimal. Deeplearning4j is an open source distributed deep learning project in Java and Scala that provides both single node and distributed training options. One of it's advantages over Python based deep learning frameworks is that it was designed for the Java Virtual Machine, or JVM. So it's great for those who don't want to have Python as part of their development process. It also has support for CPUs as well as GPUs.



## **Question 1 of 3**

One of the key advantages of BigDL over the other libraries is that it's optimized to use \_\_\_\_\_.

* GPUs instead of CPUs  
  Incorrect
* CPUs instead of GPUs  
  Correct
* Java Virtual Machine, or JVM
* distributed training options  
  Incorrect

## **Question 2 of 3**

Roxanne is looking to perform some image classification, without having a large amount of training data. Which way should Roxanne perform deep learning in Spark?

* She should use the deep learning community in Spark, where she can use the images of other users who have also created a Community Edition.
* She should use transfer learning, which implements a trained neural network on a dataset that is similar to the one she's working on.  
  Correct
* She should use inference learning, which pulls uploaded images from her desktop and automatically completes her image classification.
* She should use model training, which copies images from her phone onto her computer's hard drive, allowing instant access to all of her images.

## **Question 3 of 3**

Why would you also print out the shape of a file when using the imread function?

* to confirm that an image has not been read yet
* to confirm that an image will print in full color
* to confirm that an image has actually been read  
  Correct
* to confirm that an image is RGB in color and shape

### **Setting up a Databricks environment**

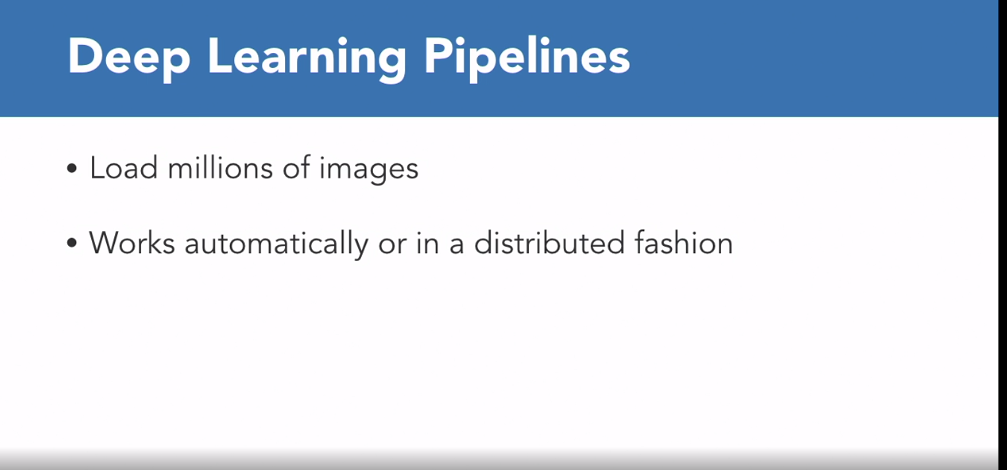
Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] So head over to Databricks and log in and let's head over to the Cluster and select Create Cluster and give it a name. Let's call it DL, for deep learning, and select Create Cluster. It takes a couple of seconds to create this cluster and once the cluster has been created, the state moves from pending to running. So hover over the cluster and select Libraries. **You now want to install the relevant libraries. So select Install New and from the Library Source select Maven,** select Search Packages, and wait for the list to populate. Sometimes the list doesn't come up immediately, so what I do is I close the window and select Search Packages again. So I now enter, **deep learning**. So that's deep learning and I can see that option is available to me. And we're going to be using release number 1.4.0. So select that and select the option using that link and now click on Install to install this package. We need to install a couple of other libraries, so head back to Install New and this time select **PyPi as a source and enter** TensorFlow in the packages. So, TensorFlow equals equals and we want to install version 1.12.0. Select Install. Head back to Install New. S**elect PyPi and this time you want to install Keras.** So that's Keras equals equals and we want to install Keras version 2.2.4. Select Install. Now TensorFlow and Keras are deep learning frameworks. **What we also need is the H5Py package, which is a Python interface to the HDF5 binary data format.** So I select Install New. Select PyPi and type H5Py and I want to install version 2.7.0. **And the last library we need to install is the wrapt and this is used for creating function wrappers and decorator functions. So select Install New, PyPi, and wrapT or wrapt and select Install. And once all five packages have been installed, we're now in a position to run deep learning in our Apache Spark environment.**

### **Working with images**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] In the past, it's been a challenge working with images in Spark, as getting images into data frames was difficult. This has changed with the introduction of deep learning pipelines. Deep learning pipelines now include functions that can load millions of images into a Spark data frame. In typical Spark fashion, this can be done automatically in a distributed way. So let's load an image into a data frame on Spark and head over to our Databricks notebook. Now just to make it easier to see, I'm going to bump up the font on the screen. So let's upload an image to Databricks. So I select data. I select add data. I select browse. I head over to my exercise files to the images folder, and I'm going to select the typewriter. So we want to make a note of where this file is stored in Databricks. So I'm just going to copy that. So that's /FileStore/tables/typewriter.jpg. I'm going to copy that path, and now I'm going to go to workspace. I'm going to select the arrow that points down. I'm going to select import, and now I'm going to import the notebook from my chapter. So select browse. Select exercise files. 03\_02\_begin. I then select import, and the first couple of lines just ensure that we have installed the correct version of the packages. I run this by selecting shift and enter. Now the notebook isn't attached to a cluster. So I select attach and run, and I can ignore the deprecated warning in the bar here. The next section is about importing the image into a data frame. So this is where I need the path to that file. So I'm going to store that location of that file in the variable img. So img equals colon, and then I'm going to paste the location that I've stored of that file. Shift and enter to run that. We then want to import the image schema method from the ml.image module. **So from pyspark.ml.image import ImageSchema. I** then run that cell to import this method. Now ImageSchema.readImages, and we provide as a parameter the image will return a data frame. So df equals ImageSchema.readImages.img, and to show that, I do a df for my data frame, show, and I'm also going to do a df.printSchema to see the schema of my data frame, and I'm going to run that cell, and you can see that the image is now stored in a data frame. So you can see it's schema, and you can also see that it's got a column called image.



## **Question 1 of 2**

Which command allows you to import and run images in Databricks?

* Fn+V
* Control+P
* Shift+Enter  
  Correct
* Shift+Command

## **Question 2 of 2**

When are you ready to run deep learning in the Apache Spark environment?

* once all packages have been installed  
  Correct
* once half of the packages have been installed
* while you are installing packages
* before you have installed any packages

### **Using pretrained models**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Deep learning pipelines support running pretrained models in a distributed manner with Spark. You can do this with both batch and streaming data processing. The ImageNet data set has 1,000 different categories of objects. The different categories are here in the synset file. I open this with Wordpad. Now each of these rows corresponds to one of the categories or classes in the ImageNet data set. The first part is the ID and the second part is one or more words to describe the class. We can write code using well-known models that have won competitions classifying the 1,000 different objects. These models are ResNet 50 and Inception version three amongst others. You can then input any image that belongs to one of these 1,000 categories and the deep image predictor method will predict which of these objects have been loaded. This prediction, of course, is done in parallel with all the benefits that come with Spark. Let's head over to the databricks notebook. I'll select workspace. So I'll select the down arrow. I select import. I select browse. I select the notebook 0401 begin. And I select import. This notebook should look familiar as this is where we finished up when working with images. So let's use this as our starting point. The first thing we want to do is confirm that we've got the right packages installed, including the deep learning package. I select attach and run to attach this notebook to the cluster. So I can ignore this error message. You want to then import the image into a data frame. And we then want to use the deep image predictor method from the Spark dl module. **This Spark dl module comes from the deep learning package that we attached to the cluster as part of our setup. From sparkdl import DeepImagePredictor**. And let's say predictor equals DeepImagePredictor. The input column will have the column name image as we've seen in the data frame. The output column is predicted labels. We're going to be using the ResNet 50 model. So modelName equals ResNet50. We're going to be decoding the predictions. So this is making sense off the class from that synset file. And we want to see just the top five predictions. So I've got an error here because it needs to be a top with a capital K. Now the deep image predictor is doing a couple of things. Once an image is input, it needs to determine which of the 1,000 classes of ImageNet the object and the image corresponds to. And by default, each of the objects are given a probability so that the deep image predictor sorts them based on probability and since we've specified a topK equals five, it displays the objects with the top five highest probabilities. Let's give the predictor data frame the name predictor. Predictor\_df equals predictor.transform. And we'll use the transform method to perform a prediction on our data frame which holds the image. So that's df and then predictor\_df.select. And then you want to select the predicted labels column. So predicted labels. And we do a show truncate equals false. And let's run that cell. Now remember that our original image was a typewriter. And we can see that the topmost prediction with a probability of 97% is typewriter keyboard. The next one is a space bar with 2.7%. The next one is a computer keyboard and so on. And the first part of this prediction is the synset ID that we've seen in that file. So we can see that we've been able to get pretty accurate results using a pretrained model.

### **What is transfer learning?**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] One of the best ways to get started with deep learning, is to use transfer learning. Transfer learning is reusing a trained neural network. This trained neural network would have been trained on a data set that is similar to the one you're working on. Many image classification models are trained on ImageNet, this has 1000 classes of objects and animals. Now if you're tryna tell the difference between different types of flowers, then transfer learning using a model that was trained on ImageNet would be a very good start. This is because the edges, colors and textures of objects that would have been activated in the first few layers of the model, will also be relevant with the data set with different kinds of flowers. When you're tying to distinguish between flowers, this will only take place at later layers of your deep learning model. The first thing we do in transfer learning is to remove the top of our pre-train network. We then replace this with our own custom classifier. We need to do this because if the original image classification model was trained on ImageNet, then it will provide us as output the probability for 1000 classes of objects. In the example we will look at, we only want to distinguish between two varieties of flowers, and so we only want to have two output classes. The next thing that is done is freezing the first couple of layers. All this means is that when we need to carry out training for the new classifier that we've added on, the weights for the first couple of layers will not be updated. If we don't do this, then whatever features and textures that were captured as a result of training on ImageNet is lost. This is because as part of training, we start off with random weights. We then go ahead and train the classifier that was added. We'll next look at implementing this in the deep learning package.

### **Transfer learning in action**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] So, we're going to do transfer learning using the ResNet50 model. Now, the ResNet50 model has been trained on the ImageNet data set, which has 1,000 classes. So, we've got a couple of varieties of flowers in the flower data set that we'll use. We've got daisies, dandelions, roses, sunflowers, and tulips. Now, it so happens that daisies are one of the 1,000 categories in ImageNet, so we won't pick them. Instead we'll use transfer learning to distinguish between two classes of flowers, tulips and sunflowers. So, let's head over to the notebook, so I select Workspace, I select the down arrow, I select Import, Browse, I select the notebook 04\_03\_begin. I select Import, so we'll do our usual checks, confirm that we've got the correct libraries installed. I select Attach and Run to attach this notebook to the cluster. And then what you're going to do is to download the flower data set. Enter and unzip it, go into the flower photos and just confirm that we've got the different folders with the flower names, so we've got the daisy, dandelion, roses, sunflowers, and tulips, and the path to the flower photos is going to be stored in the image directory, so that's image\_dir. So, we want to load the images into the data frame. Now we've seen how we can do that using Spark's ImageSchema. There's another way of doing that using a custom image library where we use the method imageIO and we can use the readImagesWithCustom function and we can decode the image using Python's pillow library for images. We now come to the bit where we're actually going to be doing the transfer learning, and there are a couple of things we need to do. We want to add our own classifier to our already trained network, and in the deep learning library we use for Spark the DeepImageFeaturizer automatically removes the last layer of a pre-trained neural network. So, let's call this "featurizer," and we'll use the method DeepImageFeaturizer. We want as the input column the image. We want as the output column the features. And we're going to be using the ResNet50 model, so the model name is ResNet50. Now, the logistic regression algorithm would be a good choice as our new classifier because it allows us to distinguish between two classes. So, let's have lr for logistic regression. We're going to have 10 passes through our data, so a max iteration of 10, and these are some of the hyperparameters that you can specify, so the regParam is going to be 0.05, and you can try out other hyperparameters to see if you can get better accuracies, and elasticNetParam equals 0.3, and the label column is going to be called 'label.' Now, we want to make sure that we only train the logistic regression classifier that we've added. We can do this using pipelines. So, let's call our pipeline flo for flower, so pipeline, we want to first go through the featurizer, and then the logistic regression. Now, the next thing we want to do is to do the training for the sunflower and the tulip data set, but first we need to create our training data set. Right, and I need to include the parameter stages, stages equals the featurizer and the logistic regression, and that's cleared that up for us. Now, creating the train and test class is pretty straightforward. We import the ImageSchema methods and the imageIO, which allows us to create the custom function, so on the first line we have the tulips data frame, and we give it the label zero, that's what we do by the lit(0). So, from the sunflowers data frame let's do things a little bit differently. Let's use the readImagesWithCustom function, provide the path to the sunflowers data set, and this time the sunflowers will have a label of one, and that's specified with a lit(1). Now, normally when you do training and testing you have 60% for your training data set, 20% for your test data set, and 20% for your validation data set, but because we're using the community addition clusters we don't have a massive amount of compute power, so let's scale this down to only using 8% of our data set for training and testing, and we can randomly split them up using the randomSplit method here, so we have the tulips\_train and tulips\_test data set using only 8% of the data set each, and we do exactly the same thing for the sunflower training data set. In line five of our code we combine the tulip and the sunflower training data set and call it train\_df. We do something very similar with the test dataset combining both the sunflower and the tulip data set, and we call that test\_df. Now, under the hood each of the partitions is fully loaded in memory, which can be quite expensive, so to ensure that the partitions have a smaller size we use the repartition function there. So, let me run this block of code. I need to first import the functions that we'll be using, so I import the functions and then I rerun that block of code. So, we're now in a position to train only the classify that we've added, so let's call our model f for flower, so f\_model equals flo.fit, and you want to use our training data set. Now, this should take between five and six minutes to run, so go ahead and grab a tea or a coffee. Now, let's see how accurate our model is using the multi-class classification evaluator. Now, we've seen this bit of code in the previous video, so I'm just going to reuse it here. Now, this should also take between five and six minutes to run. Now, that was worth waiting for. That's pretty impressive. By using transfer learning we have achieved an accuracy of over 90% to distinguish between tulips and sunflowers. Now, your value of the accuracy will differ from mine because we've randomly split our training and test data set. Next we'll train our new transfer learning model on two images that are not in our train or test folder, one of a sunflower and one of a tulip. Now, don't close this notebook because we'll be using our new trained model, f\_model, to test our two new images.

### **Testing your new model**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Now in order to test how good our newly-trained model f\_model is, I'm going to use an image of a tulip and a sunflower. So let's upload these images to Databricks. So I select Data, I select Add Data, I select Browse, I select the flower image, which is in my Exercise files in the Images folder. I click on the Databricks icon again, I select Data, I select Add Data, I select Browse, I select the tulip file and this time I make a note of where this file was stored. So I copy that. I select the Databricks icon again, and then I go back to my Notebook by just clicking Back. And let's store the path to the tulip file in the variable t. So t equals tulip.jpg and in the same way, s for sunflower and the name of my image is sunflower.jpg. Now what I need to do is to take these image files and convert them into DataFrames. So from pyspark.ml.image import ImageSchema. And this method should already be exported based on what we've done in the previous section of the file. So let's call our DataFrame df equals ImageSchema, readImages, s for sunflower and we'll do a df.show. So I haven't run my cell with the tulip and sunflower so let me run that first. And then let's go back to the ImageSchema and let's run this again. So we can see that the DataFrame has got the column name image. And now let's run our model on this image, so f\_model.transform and we pass out our DataFrame df. And we do a show. And let's see what the result is. Now remember that sunflowers were labeled with a one and tulips with a zero. So our model has returned a value of one, which is a prediction of one, so it's predicted that this indeed is a sunflower but it doesn't seem to be very convinced of it, we've only got a 5% probability of that. Let's see if we do any better with the tulip model. So I'm just going to rerun the same couple of cells but this time I'm going to use the tulip image as the data so in the same DataFrame df. And I'm going to run this again. So we can see that with a tulip image, the model has accurately predicted that it is a tulip with a value of zero and it seems far more convinced of this, with a probability of almost 99%.

## **Question 1 of 2**

Jesse wants to make sure he only trains the logistic regression classifier that he's added. To do this, Jesse would use \_\_\_\_\_.

* a featurizer
* pipelines  
  Correct
* logistic regression
* a training data set

## **Question 2 of 2**

In transfer learning, why do you need to remove the top of your pretrain network and replace it with your own custom classifier?

* If you did not, it would only provide output probability for 1,000 classes of objects.  
  Correct
* If you did not, the first couple of layers would be frozen and the probability would always return an error.
* If you did not, it would only provide output probability for 100 classes of objects.
* If you did not, then all of your images could only be used once.

### **Deploying models as SQL functions**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Now one way to take a deep learning model to production, is to deploy it as a Spark SQL UDF or User-defined Function. This means that anyone who knows SQL can use it. How cool is that? So let's head over to our Notebook. I select Workspace, I select the Down Arrow, I select Import, File, Browse, Exercises, 05\_01... \_begin, and I select Import. Now we need to check we have the correct versions of the libraries installed, including the Deep Learning Library. So attach the notebook to the cluster. We're going to use the flower dataset again, but if you've downloaded it to your cluster you don't have to do that again. So you've got the directories daisy, dandelion, roses, sunflowers, and tulips under the flower\_photos directory. And this time we're going to create a directory called sample\_images and copy three of the daisy images across, and we're going to load the three images of the daisy into the dataframe. Now in the dataframe you can see the column named Image, with the three rows corresponding to the path to the three images. Now this time 'round we're going to use the Inception Version 3 Deep Learning Model. Databrick's Deep Learning Pipelines allows you to take a deep learning model and register a Spark SQL User-defined Function or UDF. Now, if you've worked with Keras and images before, you'll know it's common to have pre-processing steps to an image before passing it to a model. Well, we can do that here. So we create a Keras function called keras\_load\_img that converts the images to dimensions of 299 by 299 and then returns an unsigned integer. Now as part of our workflow, we can provide this pre-processing function to the UDF registration. So let's register our UDF, this should take a couple of moments. Now, because we haven't used the Inception Version 3 Model before, this is being downloaded from the Keras library on Github. Once a UDF has been registered, it can be used as an SQL query. Now if you remember, we had three images in our folder and each of these numbers correspond to the 1000 classes. So, I'm just going to copy the data for one of the images and if I put it into the length function, as this is a list, we can see that there are 1000 elements. Unfortunately, there seems to be a bug in the code because regardless of what image I use the 110th index always has the highest probability. What we would expect is that the highest value will correspond to the relevant class in the image in that dataset, and we can then easily decode this and map it to the image in that class. So let's do that. We'll use the decode\_predictions method from the Inception Version 3 Module, and we'll store the output of those 1000 classes which is a list in out. We'll convert to a NumPy array. Now, because decode\_predictions is expecting a batch of predictions of the format samples comma 1000, we'll reshape out list to the required format and then provided the output. Now unfortunately, regardless of what image you use, you will always get the output of flatworm. However once Databricks fixes this code, this means anyone who knows how to use SQL can use this deep learning model in this way. And what's great, is that this works on both batches of images or streams of images.

## **Question 1 of 1**

What allows you to take a deep learning model and register a Spark SQL user-defined function (UDF)?

* Databrick's classification evaluator
* Spark's image schema
* Databrick's deep learning pipelines  
  Correct
* Spark's training dataset

### **Future project ideas**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Jonathan] Now we've looked at Databricks' Deep Learning package for Spark which is very feature-rich. With the Databricks Community Edition that we've been using as our lab environment, we haven't been able to run our processing in parallel. Go ahead and process these images in a standalone Spark environment with multiple workers or in a cloud environment and see the difference when Spark processes these models and images in parallel. This is why companies use Spark for processing millions of images at a time. Now what if your use case is distributed training? Then check out some of the other packages such as Intel's BigDL, Yahoo's TensorFlowOnSpark, or Deeplearning4j if you also know Scala. I hope you found this course helpful. Thanks for watching, and I'd love to hear back from you and to connect via LinkedIn. Enjoy, and (speaking foreign language).