Microsoft

Image Analysis Demo

Last updated: 10/2/2017

# Introduction

This document will walk through the architecture of the Energy demo, providing details about the proposed scenario and the different components that are part of it.

Image Analysis recreates a scenario that allows an energy company to improve their response time and reliability on the field, by performing automated analysis of the power lines status thanks to Azure, machine learning, and some drones. These drones will survey the power lines and take pictures that will be analyzed using a machine learning image detection system on Azure, allowing engineers to analyze them for anomalies easily, faster, and without having to be in the field.

This machine learning image detection system is powered by Azure and CNTK. To be able to build a business that handles and analyses images fast and reliably, they do not only rely on Azure’s proven scalability and availability but also its breadth of features to get to market quickly. By using services like HDInsight and CNTK, the company can focus on providing their engineers the best options when it comes to keeping the systems running smoothly.

Before continuing, please ensure you have installed the following:

* A web browser
* An SSH client (we use [MobaXterm](http://mobaxterm.mobatek.net/) for this document)
* An API client (we use [Postman](https://www.getpostman.com) for this document)

# SETUP

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
|  | 1. Open Powershell. | Before starting the demo, it is necessary to follow a couple of steps to deploy the Azure resources. |
| cd {PATH\_TO\_SOLUTION}\EnergyDemoARMDeploy | 1. Replace {PATH\_TO\_SOLUTION} with the path where the Demo solution is located. 2. Execute the command. |  |
| .\Deploy-Demo2.ps1 | 1. Execute the command |  |
|  | 1. Introduce your credentials and then click on Sign in. |  |
|  | 1. Introduce the option of your selected subscription 2. Press Enter | You will be displayed a list of available subscriptions for your account. Use whatever it fits best for you. |
|  | 1. Write a name for the deployment 2. Press Enter | Choose a name for the deployment. This name will be used to prefix all the deployed resources. **Make sure it meets the following requirements:**   * **Contains only letters or numbers (but not only numbers), as no other character is allowed**. * **It does not exceed 6 characters**.   Also make sure that is a unique name, not taken by any other resource. |
|  | 1. Write down the values of ClientId and TenantId | With that, the deployment of the resources will start.  It will start creating an Azure Active Directory application that is necessary to follow this demo. Write down the ClientId and TenantId values, as you will need them later. |
|  |  | Bear in mind that it could take up to 2 hours to be ready, since resources like HDInsight cluster take a long time to be created. |
|  | 1. Go to the Azure Portal 2. Login with your credentials 3. Click on Azure Active Directory | Once deployed, it is necessary to grant permissions for your user in the recently created Azure AD application. |
|  | 1. Click on App registrations |  |
|  | 1. In the search box, type the ClientId previously generated. 2. Click on the application found |  |
|  | 1. Click on Required permissions |  |
|  | 1. Click on Grant permissions |  |
|  | 1. Click on Yes | With this operation, your account has now permissions to access the PowerBI APIs. They are used later to send the processed information to PowerBI, so you can see the generated dashboard and consult the available information regarding the image analysis experiment. |
|  | 1. Open the [PowerBI webapp](https://app.powerbi.com/) 2. Sign in with your credentials | Now we need to create a workspace. This workspace will be used to store a dataset in which the data will be sent. |
|  | 1. Click on Workspaces |  |
|  | 1. In the opened blade, click on *Create app workspace* at the bottom. |  |
|  | 1. Enter a name for your workspace. Choose whatever you want. Write it down as you will use it later. 2. Leave the rest of options as they are. |  |
|  | 1. Click on Save |  |
|  |  | And that’s all!  Now you have everything you need to start the demo! |

# TRAINING THE MODEL

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
|  | 1. Open the Azure Portal 2. Login with your account 3. Click on Resource Groups | Now that everything is in place, we’re going to go over the image analysis system that has been created. In this scenario, we begin with a set of aerial images of the power lines, obtained via drones. Some of them are clearer than others, and they provide useful information in order to diagnose possible maintenance problems. Those images have been used to train a machine learning model, with CNTK on top of HDInsight, that can identify regions of the image: the mast, the top cover of the mast, the insulators… This greatly simplifies the work of the technicians that analyze these images, and opens the door to further improvements that allow the system to automatically detect anomalies on an image.  We’re going to analyze how this CNTK model works. |
|  | 1. Click on the deployed resource group 2. Click on the Spark cluster | We are going to review the scripts that train our CNTK model. Begin by connecting to the cluster via SSH. |
|  | 1. Click on Secure Shell SSH |  |
|  | 1. Copy the hostname |  |
|  | 1. Open your SSH client 2. Connect to the host previously copied | To connect using SSH, we need a SSH client like PuTTY or MobaXTerm. Make sure you have installed one of them. |
|  | 1. Use the following credentials:  * drwho * Patata.123456  1. If everything went correctly, you should see a welcome screen. |  |
| ls CNTK | 1. Execute the command 2. Review the available resources | The required scripts and resources have already been copied to the /home folder of the Spark cluster. Even though the model has already been initially trained (it’s a very time consuming operation, so we’re using a base AlexNet model), we’ll go over the steps we’ll perform to retrain it using our drone images.  First, let’s review what resources we have available, most of them being Python 3.5 scripts:   * fastRCNN folder: Contains fastRCNN libraries * selectivesearch folder: Contains a library that implements the Selective Search algorithm * cntk\_helpers.py and imdb\_data.py: Helper libraries * ImageDetectionModelCreator.py: Contains code to load, train and evaluate a FastRCNN CNTK model * ROIGenerator.py: Contains code to generate ROI (regions of interest) for a given image. * IncrementalTraining.py: Trains (or re-trains) the model, based on the images obtained by the drones. * PARAMETERS.py: Establishes common values used when training, re-training and evaluating images. * EvaluationOutput.py: Contains code to evaluate the results of analysing the images using the trained model.   Let’s review the most important files one by one |
| cat CNTK/PARAMETERS.py | 1. Execute the command | Take a look at the contents of the PARAMETERS.py file. In here, a set of variables that will be used throughout the work with the model are defined, including those relative to the ROI generation (number, size, aspect ratio…), size and number of images to be used, where to find those images, how many epochs will be run to train the model, which classes are defined and could be found in each image, etc.  Most of these parameters contain a brief comment that outlines their purpose, as well as a descriptive name. |
| cat CNTK/ROIGenerator.py | 1. Execute the command | Now let’s review the ROI generation system. It’s based on an implementation of the Selective Search algorithm (<https://www.koen.me/research/selectivesearch/>) and defines a function called generate\_input\_rois.  This function will be called either for training/re-training or for evaluation purposes (in which case the generate\_for\_evaluation parameter will be true). It applies the selective search method to the available images in order to define ROIs, or regions of interest, where an object could be found.  As before, there are a few comments in the code that help clarify the different steps. |
| cat CNTK/EvaluationOutput.py | 1. Execute the command | In EvaluationOutput.py there is an evaluate\_output function that reviews the results of the model training to provide a score that can be used to measure how good (or bad) the model is at identifying components in the image.  As before, there are a few comments in the code that help clarify the different steps. |
| cat CNTK/ImageDetectionModelCreator.py | 1. Execute the command | This is the biggest file, since it contains all the code required to load, train, re-train and save the CNTK FastRCNN. Let’s review the most important functions:   * train\_fast\_rcnn will take a model and either train it with a set of images, or re-train it if a train checkpoint has been previously created. To avoid the cost of training from scratch, a trained model has been stored in blob storage. * evaluate\_fast\_rcnn will run the model against a test dataset to see how accurate it is   As before, there are a few comments in the code that help clarify the different steps. |
| cat CNTK/IncrementalTraining.py | 1. Execute the command | The IncrementalTraining.py script will perform all the required steps to get data, train and evaluate the model in a sequential order so this will be the only script you need to execute directly.  As before, there are a few comments in the code that help clarify the different steps, and all of them will rely on the previously reviewed files. |

|  |  |  |
| --- | --- | --- |
|  | 1. Go to the Azure Portal 2. Go to the deployed resource group 3. Click on the Azure Storage account | To execute the script we need to get a couple parameters: the storage account name and key, and the Azure Cosmos DB account endpoint and key.  Let’s see how we can take them. |
|  | 1. Click on Access keys |  |
|  | 1. Write down both the Storage account name and the key 2. Close this window |  |
|  | 1. Go back to the deployed resource group 2. Click on the Azure Cosmos DB account |  |
|  | 1. Click on keys |  |
|  | 1. Copy both the URI and Primary key, and write them down |  |
|  | 1. Click on Blobs | Before we execute the script, let’s take a look at the files we’ll be working with. |
|  | 1. Click on imageinbox |  |
|  | 1. Take a look at the available files 2. Click on a \*.bboxes.labels.tsv file | Here we have all the files that will be used for the training. They are divided in three types: \*.jpg, \*.bboxes.tsv and \*.bboxes.labels.tsv |
|  | 1. Click on Download 2. Repeat the same steps to download the corresponding bboxes.tsv and .jpg file | Let’s download one of each and see what they contain |
| C:\Users\fmartinezmiranda\AppData\Local\Microsoft\Windows\INetCache\Content.Word\1000093196.jpg | 1. Open the .jpg file | The image shows an aerial picture of a power line mast |
|  | 1. Open the bboxes.tsv file | This file contains the identified regions of interest in the image. Each column of the file contains a coordinate, and each row corresponds to a single region |
|  | 1. Open the bboxes.labels.tsv file | This file contains the labels that correspond to each region in the previous file. With both files and the image, we can provide the model with a training set of images. |

|  |  |  |
| --- | --- | --- |
| export PYSPARK\_PYTHON= /usr/bin/anaconda/envs/py35/bin/python3.5 | 1. Execute the command | Let’s go back to the Spark cluster connection, and execute this command. It will tell Spark to execute with the Python 3.5 version, instead of using 2.7 |
| spark-submit CNTK/IncrementalTraining.py '{STORAGE\_ACCOUNT\_NAME}' '{STORAGE\_ACCOUNT\_KEY}' '{AZURE\_COSMOS\_DB\_ENDPOINT}' '{AZURE\_COSMOS\_DB\_KEY}' | 1. Replace {STORAGE\_ACCOUNT\_NAME} with the name of the storage account 2. Replace {STORAGE\_ACCOUNT\_KEY} with the key of the storage account 3. Replace {AZURE\_COSMOS\_DB\_ENDPOINT} with the URI from Cosmos DB 4. Replace {AZURE\_COSMOS\_DB\_KEY} with the Cosmos DB primary key 5. Execute the command | Now, replace the two parameters we obtained in a previous step, and let’s execute the IncrementalTraining.py script and see the results step by step |
|  | 1. Check the screen and follow the script | The script begins by downloading from the provided blob storage, the images that will be used to train the model |
|  | 1. Check the screen and follow the script | The ROI generation method will be called for each of the downloaded images, as a necessary step before training the model with them. The images will be split between test and train images, with 70% of them used for training. |
|  | 1. Check the screen and follow the script | Since this is the first run, no checkpoint is found, and no checkpoint will be used, so the base model will be retrained using our image set. |
|  | 1. Check the screen and follow the script | The script will run for the number of epochs defined, improving the model precision with each step. |
|  | 1. Check the screen and follow the script | Once each epoch is complete, it will output a summary of it before continuing. When it has completed, it will validate the trained model using the provided test images. |
|  | 1. Check the screen and follow the script | After evaluating the test images, we get the results: the mean AP (average precision) that we obtain after comparing the results of our model’s evaluation of the test images, with the ground truth results we provided (by means of the \*.bboxes.tsv and \*.bboxes.labels.tsv files).  Now the model is ready, and both the trainer checkpoint and the model itself have been uploaded to blob storage, so we can proceed with the next steps: evaluation of a single image and re-training of the model. |

# EVALUATING A SINGLE IMAGE

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
|  | 1. Open the Azure Portal 2. Login with your account 3. Click on Resource Groups | With the model already trained, we can now see how it fares by evaluating a single image. This is part of the standard behaviour: once the model is trained, we want to use it to obtain results from the images we’re obtaining in real time from our drones in the field.  To start, let’s check the website where the API has been deployed. |
|  | 1. Click on EnergyDemo resource group 2. Click on the demo website |  |
|  | 1. Copy the website url | This is the URI of the website where the API has been deployed. This API has an evaluate method that receives an image, and returns the results of evaluating it as a JSON: what objects have been found and where.  Let’s get an image to evaluate and then use Postman to call and test the API |
|  | 1. Go back to the EnergyDemo resource group 2. Click on the Azure Storage account |  |
|  | 1. Click on Blobs |  |
|  | 1. Click on images |  |
|  | 1. Click on evaluationImages.zip | This file contains a set of images that can be used for evaluation. In the real world, they will be obtained from the drone and directly submitted to the API. |
|  | 1. Click on Download 2. Unzip the contents in the Desktop | Once we have the files to be evaluated, let’s open Postman and call the API |
|  | 1. Open Postman 2. Paste the API url in the “Enter request URL” box, and append “/api/evaluate” |  |
|  | 1. From the dropdown menu, select POST |  |
|  | 1. Select Body and form-data |  |
|  | 1. Select File as Key |  |
|  | 1. Click on Choose Files 2. Select one of the previously downloaded image files | This image is the one we want to provide to be evaluated by the model |
|  | 1. Press Send | Postman will call the API and submit the provided image. |
|  | 1. Wait a moment and check the result | There you go, that JSON contains the info of the objects detected by the model: to what class they belong and the coordinates of the detected object.  Now let’s see how to improve these results by retraining the model |

# EVALUATING A SINGLE IMAGE using databricks

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
|  | 1. Open the Azure Portal 2. Login with your account 3. Click on Resource Groups 4. Click on the EnergyDemo resource group | Apart from the evaluation using the HDInisght cluster, we are going to show how to perform it using a platform called Databricks, which simplifies the cluster management.  Databricks is a unified approach that makes data analytics simpler, with a fully managed cloud infrastructure and a collaborative workspace to allow data scientists and data engineers work together.  And now it is available in Azure! |
|  | 1. Click on Add |  |
|  | 1. Search databricks 2. Click on Azure Databricks (preview) |  |
|  | 1. In the opening blade, click on Create |  |
|  | 1. Fill the field Workspace name with the name you want. 2. Leave the rest as it is. |  |
|  | 1. Click on create | Now wait until the resource is created. |
|  | 1. Access to <https://westeurope.azuredatabricks.net/> 2. Click on Sign in with Azure AD 3. Login with your account | Once created, let’s access to the portal. |
|  |  | Now we are inside the Databricks portal. The core of Databricks is Spark, so we are going to create a Spark cluster so we can run our evaluation script on it.  We could do it using the portal, but at the time of writing this document, Spark clusters that use Python 3 are only available through the API. We have prepared a script to do so. But first is necessary to create a token to access the Databricks API. Let’s do it. |
|  | 1. On the top right corner, click on the user icon 2. Then, click on Admin Console. |  |
|  | 1. Click on Access Control |  |
|  | 1. In the Personal Access Tokens section, click on Enable | This will allow users to use personal access tokens instead of passwords to access the Databricks REST API. |
|  | 1. On the top right corner, click on the user icon 2. Then, click on User Settings. | Now let’s create the access token. |
|  | 1. Click on Generate New Token |  |
|  | 1. Fill the comment field, and leave the Lifetime as it is 2. Click on Generate |  |
|  | 1. Make sure you copy the token generated, as it will not be available once you click on Done 2. Click on Done | Now it’s time to use the script and create the cluster. |
|  | 1. Go to EnergyDemo 🡪 CNTK 🡪 Databricks 2. Open CreatePython3Cluster.py |  |
|  | 1. Replace *{REPLACE\_WITH\_CLUSTER\_NAME}* with the cluster name you want. 2. Replace *{REPLACE\_WITH\_YOUR\_TOKEN}* with the generated token. | This python scripts basically creates a POST request against the Cluster Management API from Databricks, using the Create cluster endpoint.  The body contains the configuration of the Spark cluster. We are going to use the version 3.2 of the cluster (that uses Scala 2.11). Also, we are setting some configuration in order to make the CNTK scripts work.  Finally, we are setting the PYSPARK\_PYTHON environment variable, so Spark uses Python 3 when submitting jobs. |
|  | 1. Open the Command Prompt |  |
| cd {PATH\_TO\_ENERGY\_DEMO}\EnergyDemo\CNTK\Databricks | 1. Replace {PATH\_TO\_ENERGY\_DEMO} with the path to your EnergyDemo solution 2. Execute the command in the cmd |  |
| python CreatePython3Cluster.py | 1. Execute the command | If the request is done correctly, the response will be the id of the cluster. |
|  | 1. In the Databricks portal, click on Clusters. | The cluster is now being created. It may take up to 20 minutes to be ready.  In the portal, going to the cluster section, you will see the state of the cluster. When it is ready to be used, the state will pass from Pending to Running. |
|  | 1. Click on workspaces | Now we need to attach some libraries to the cluster. Databricks allows to add what are called Shared Libraries, which can be used across all the cluster that we create in the platform, and are deployed in all the nodes of the cluster. The platform automatically manages this, so we just need to point what libraries we want to use. |
|  | 1. Click on the arrow next to Shared 2. Click on Create 3. Click on Library | Databricks allows to add packages that are available on [PyPI](https://pypi.python.org/pypi) (Python Package Index). |
|  | 1. Select Upload Python Egg or PyPI |  |
|  | 1. Copy and paste the list of libraries in the field:   future, Pillow, xmltodict, wheel, matplotlib, EasyDict, numpy, scikit-learn, scipy, scikit-image, Cython, pandas, statsmodels, urllib3, opencv, azure, opencv-python   1. Click on Install Library | It is easy to add multiple libraries at the same time. Just separate them with commas and you are done. |
|  | 1. Click on the Attach checkbox | Now we need to select which clusters are going to use that libraries. We need to attach them to the clusters.  In our case, we only have one, but we could attach the library at multiple running clusters.  It may take a couple of minutes to attach the libraries. Be sure that the status is changed to Attached before continuing. |
|  | 1. Go to the workspaces section 2. Click on the arrow next to *Workspace* 3. Click on Import | Now we need to install CNTK. We have to install it using a different method since it is not available through PyPI.  Databricks has a method to execute custom scripts on the cluster called Init Scripts. These are Shell scripts that are executed when the cluster is being initialized. These scripts are located in a special directory in the Databricks File System (DBFS), which is a distributed file system installed on all Spark clusters in this platform.  Let’s see how to create a Init script. |
|  | 1. Click on the box to upload a file | We have a notebook prepared that setups a Init script. Let’s import it. |
|  | 1. The notebook is in EnergyDemo\CNTK\Databricks\Notebooks 2. Double click on InstallCNTK.ipynb |  |
|  | 1. Click on Import |  |
|  | 1. Replace {REPLACE\_WITH\_CLUSTER\_NAME} with the name of your cluster. | The imported notebook will be opened automatically.  The shell script is stored in a variable that will be later dumped in a file.  The biggest advantage of using Init scripts is that you can do whatever you would do in an actual cluster. So, if you seem limited by the Databricks platform, you can always use the potential of the bash to execute any tasks that otherwise would be impossible.  All the notebooks have access to the DBFS API. There is a list of functions to easily operate with it, like list files, create directories, remove files/directories, and so on.  We are using the DBFS API here to put a shell script in a folder called **init**, which store all the initialization scripts for the cluster. Inside the init folder, there is a directory for each cluster created in the platform. Be sure put your cluster name correctly. Otherwise, the script won’t be executed. |
|  | 1. Click on Detached 2. Select the created cluster to attach the notebook to it | We need to attach this notebook to a running cluster in order to execute it. |
|  | 1. Run the notebook |  |
|  |  | Ok, now the Init script is installed. But we need to execute it. To do so, we need to restart the cluster. |
|  | 1. Go to Clusters |  |
|  | 1. Click on the restart button of your cluster. | The restart may take 20 minutes or even something more. |
|  | 1. Go to the workspace 2. Import the Notebook ModelEvaluationNotebook.ipynb, which is in the same directory as the InstallCNTK.pynb one. | We are almost done. We have the necessary libraries installed, so we just need to import the notebook that is in charge of the evaluation. |
|  | 1. Click on Jobs | The evaluation is done using the available nodes in the cluster.  The Spark framework will create workers depending on the number of images it is told to evaluate. Each of these workers will create an output for the image with the results, and later they will be uploaded to an Azure Storage account.  The notebook will be run through a job in Databricks. Jobs can be attached to particular notebooks and clusters that run the notebooks. One of the advantages of using jobs is that they can be triggered through the Databricks API.  That’s how we are going to do the evaluation. So let’s create a job first. |
|  | 1. Click on Create Job |  |
|  | 1. Give it a name | Let’s attach a notebook and the cluster to this job. |
|  | 1. Click on Select Notebook |  |
|  | 1. Pick the ModelEvaluationNotebook 2. Click OK |  |
|  | 1. Click on Edit |  |
|  | 1. In cluster type, select Existing Cluster 2. Then select the cluster created 3. Click Confirm |  |
|  | 1. Write down the Job ID as you will need it later. | Our Job is ready. Now it can be run using the API. |
|  | 1. Click on EnergyDemo resource group 2. Click on the demo website | Now, let’s check the website where the API has been deployed. |
|  | 1. Copy the website url | This is the URI of the website where the API has been deployed. This API has an evaluate method that receives an image, and returns the results of evaluating it as a JSON: what objects have been found and where.  The evaluate method will call the Databricks API to run the job. It then will wait until the job is completed and will retrieve the job results from an Azure Storage container. |
|  | 1. Cllick on Application settings | Before going further, let’s set a couple of application settings for the API to work correctly. |
|  | 1. Add the following settings:  * evaluationJobId: The job id that you copied previously * token: The API token that you generated at the beginning |  |
|  | 1. Click on Save |  |
|  | 1. Go back to the EnergyDemo resource group 2. Click on the Azure Storage account | Let’s get an image to evaluate and then use Postman to call and test the API |
|  | 1. Click on Blobs |  |
|  | 1. Click on images |  |
|  | 1. Click on evaluationImages.zip | This file contains a set of images that can be used for evaluation. In the real world, they will be obtained from the drone and directly submitted to the API. |
|  | 1. Click on Download 2. Unzip the contents in the Desktop | Once we have the files to be evaluated, let’s open Postman and call the API |
|  | 1. Open Postman 2. Paste the API url in the “Enter request URL” box, and append “/api/runnotebook” |  |
|  | 1. From the dropdown menu, select POST |  |
|  | 1. Select Body and form-data |  |
|  | 1. Select File as Key |  |
|  | 1. Click on Choose Files 2. Select one of the previously downloaded image files | This image is the one we want to provide to be evaluated by the model. |
|  | 1. Press Send | Postman will call the API and submit the provided image. |
|  |  | Just out of curiosity, if you go to the Databricks portal, in the Jobs section, you will see the job running that we have requested. You can see here if it executes correctly and even take a look at the logs it generates. |
|  | 1. Wait a moment and check the result | There you go, that JSON contains the info of the objects detected by the model: to what class they belong and the coordinates of the detected object.  Now let’s see how to improve these results by retraining the model |

# Re-TRAINING THE MODEL to improve predictions

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
|  | 1. Open the Azure Portal 2. Login with your account 3. Click on Resource Groups | With the model already trained an being used to evaluate images on the fly, we’re going to see how we can improve the model.  We’ve already seen that the training generates both a trained model and a checkpoint that can be used to continue training. Let’s take a look and ensure everything is where it should be. |

|  |  |  |
| --- | --- | --- |
|  | 1. Click on the Azure Storage account |  |
|  | 1. Click on Blobs | Before we execute the script, let’s take a look at the files we’ll be working with. |

|  |  |  |
| --- | --- | --- |
|  | 1. Click on models |  |
|  | 1. Check that both imageDetection.model and trainer.checkpoint exist | The imageDetection.model is the model file we’ve used to evaluate the images taken from the drone via the web API. The trainer.checkpoint allows us to re-train the existing model, and update it when new images arrive, hopefully improving the performance and accuracy.  With this in place, we now need to set up which images we’re going to use in our re-training. |
|  | 1. Go back to the blob storage container list 2. Click on images |  |
|  | 1. Click on retrainImages.zip | This file contains a set of images to use for retraining. In the real world, they will be obtained from the drone and automatically placed in blob storage. |
|  | 1. Click on download 2. Unzip the contents of the file in the desktop |  |
|  | 1. Go back to the blob storage container list 2. Click on imageinbox |  |
|  | 1. Click on Upload |  |
|  | 1. Click on the file picker and select all the files you unzipped from the retrainImages.zip file 2. Click on Upload |  |
|  | 1. Go back to the deployed resource group 2. Click on the Spark cluster | After setting up the images, we’re ready to run the re-train script. Let’s connect to the cluster via SSH. |
|  | 1. Click on Secure Shell SSH |  |
|  | 1. Copy the hostname |  |
|  | 1. Open your SSH client 2. Connect to the host previously copied | To connect using SSH, we need a SSH client like PuTTY or MobaXTerm. Make sure you have installed one of them. |
|  | 1. Use the following credentials:  * drwho * Patata.123456  1. If everything went correctly, you should see a welcome screen. |  |

|  |  |  |
| --- | --- | --- |
|  | 1. Go to the Azure Portal 2. Go to the deployed resource group 3. Click on the Azure Storage account | To execute the script we need to get a couple parameters: the storage account name and key. If you already have them at hand from section 3, you can skip the following two steps.  Otherwise, let’s see how we can obtain them. |
|  | 1. Click on Access keys |  |
|  | 1. Write down both the Storage account name and the key 2. Close this window |  |

|  |  |  |
| --- | --- | --- |
| export PYSPARK\_PYTHON= /usr/bin/anaconda/envs/py35/bin/python3.5 | 1. Execute the command | Let’s go back to the Spark cluster connection, and execute this command. It will tell Spark to execute with the Python 3.5 version, instead of using 2.7 |
| spark-submit CNTK/IncrementalTraining.py '{STORAGE\_ACCOUNT\_NAME}' '{STORAGE\_ACCOUNT\_KEY}' | 1. Replace {STORAGE\_ACCOUNT\_NAME} with the name of the storage account 2. Replace {STORAGE\_ACCOUNT\_KEY} with the key of the storage account 3. Execute the command | Now, replace the two parameters we obtained in a previous step, and let’s execute the IncrementalTraining.py script and see the results step by step. |
|  | 1. Check the screen and follow the script | The script begins, as with the training, by downloading from the provided blob storage, the images that we just uploaded. |
|  | 1. Check the screen and follow the script | Once again, the ROI generation method will be called for each of the downloaded images, as a necessary step before using them to re-train the model. The images will be split between test and train images, with 70% of them used for training. |
|  | 1. Check the screen and follow the script | This time, a trained model and a training checkpoint exist, so they will be picked up and used for the retraining. |
|  | 1. Check the screen and follow the script | The script will run for the number of epochs defined, improving the model precision with each step. |
|  | 1. Check the screen and follow the script | Once each epoch is complete, it will output a summary of it before continuing. When it has completed, it will validate the re-trained model using the provided test images. |
|  | 1. Check the screen and follow the script | After evaluating the test images, we get the results: the mean AP (average precision) that we obtain after comparing the results of our model’s evaluation of the test images, with the ground truth results we provided (by means of the \*.bboxes.tsv and \*.bboxes.labels.tsv files).  If everything went as expected, we should have improved the performance and accuracy of our model, and we can repeat this process as many times we need provided we get more images! |