TUTORIAL (/BROWSE?CATEGORIES=["7"])

Call an operationalized Microsoft Cognitive Toolkit model from an Android app

By Don Glover (Zensa Inc) (/Home/Author?authorId=125F0BE444D9B711F04C1CA68EDBD68256991DE1EE19B2836A81DFDCAA6E9411) for Microsoft (/Home/Author?authorId=72f988bf86f141af91ab2d7cd011db47) • April 6, 2017

(mailto:?subject=Call%20an%20operationalized%20Microsoft%20Cognitive%20Toolkit%20model%20from%20an%20Android%20app&body=Check%20out%20this%20link%3A%20https%3A%2F%2Fgallery.cortanaintelligence.com%2FTutorial%2FCall-an-operationalized-Microsoft-Cognitive-Toolkit-model-from-an-Android-app)

RELATED ITEMS

Cognitive Toolkit Evaluation on Azure (/Tutorial/Cognitive-Toolkit-Evaluation-on-Azure)
TUTORIAL by Microsoft (/Home/Author?authorId=72f988bf86f141af91ab2d7cd011db47)

Cognitive Toolkit 203: Reinforcement Learning Basics (/Tutorial/Cognitive-Toolkit-203-Reinforcement-Learning-Basics)
TUTORIAL by Microsoft (/Home/Author?authorId=72f988bf86f141af91ab2d7cd011db47)

Cognitive Toolkit Tutorial: Getting Started (/Tutorial/Cognitive-Toolkit-Tutorial-Getting-Started)
TUTORIAL by Microsoft (/Home/Author?authorId=72f988bf86f141af91ab2d7cd011db47)

See all related items (/browse?s=Call an operationalized Microsoft Cognitive Toolkit model from an Android app)

RELATED LINKS

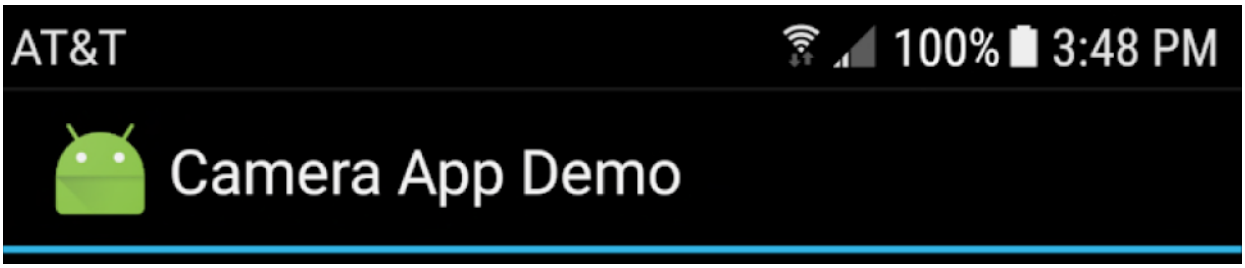
TAGS

Summary

This sample shows
you how to deploy a pretrained Microsoft Cognitive Toolkit image classification model (Resnet) as a real-time web service. You will then write a mobile app that can take a picture and pass it to the service for classification.

Description

Once you have deployed the model as a web service and built the mobile app you should get results similar to those shown in the following image:




Open Camera
Classify Image
{"result": [[["n03782006 monitor", 1407.412052154541],
["n03642806 laptop, laptop computer",
1347.8900909423828], ["n03584254 iPod",
1325.527286529541]]], "Computed in 823.68 ms"]}

To complete the sample, you will use the following tools and environments:

- Microsoft Visual Studio 2017 with the Xamarin extension. If do not have Visual Studio installed, you can install the free community edition (https://www.visualstudio.com/downloads/).
- Linux Data Science Virtual Machine. For more information on Data Science VMs, see Introduction to the cloud-based Data Science Virtual Machine for Linux and Windows (https://docs.microsoft.com/en-us/azure/machine-learning/machine-learning-data-science-virtual-machine-overview).
- Azure ML CLI for operationalization (http://aka.ms/o16ncli)
- To test the mobile app, either an emulator that can access your web cam or an android phone set up in developer mode. For more information on setting up your debugging environments, see Android SDK Emulator (https://developer.xamarin.com/guides/android/deployment,_testing,_and_metrics/on-emulator/android-sdk-emulator/) and Set Up Device for Development (https://developer.xamarin.com/guides/android/getting_started/installation/set_up

Provision a Linux DSVM

For information on provisioning a DSVM, see Provision the Linux Data Science Virtual Machine (https://docs.microsoft.com/en-us/azure/machine-learning/machine-learning-data-science-linux-dsvm-intro).

Once the DSVM is provisioned, note the IP address. You will use it to sign in to the DSVM to configure it and to call the web service you are creating.

Note: The information in this document pertains to DSVMs provisioned after February 1st, 2017.

To configure the AML CLI environment sign into the DSVM, run the following commands and follow the prompts:

```
$ wget -q http://amlsamples.blob.core.windows.net/scripts/amlupdate.sh -O - | sudo bash -
$ sudo /opt/microsoft/azureml/initial_setup.sh
```

Important: You must log out and log back in to your SSH session for the changes to take effect.

Next, enter the AML environment setup command.

Note: The following items are important when completing the environment setup:

- Enter a name for the environment. Environment names must be 20 or fewer characters in length and can only consist of numbers and lowercase letters.

- You will be prompted to sign in to Azure. To sign in, use a web browser to open the page https://aka.ms/devicelogin (https://aka.ms/devicelogin) and enter the provided code to authenticate.
- During the authentication process, you are prompted for an account to authenticate with. Use the account under which you created the DSVM.
- When the sign in is complete, your subscription information is presented, and you are prompted whether you wish to continue with the selected account.

Environment setup command:

```
$ az ml env setup
```

Once the setup command has finished, it outputs environment export commands for the AML CLI environment. It also saves these export commands to a file in your home directory. Source the file to set up your environment variables:

```
$ source ~/.amlenvrc
```

To always set these variables when you log in, copy the export commands into your .bashrc file:

```
$ cat < ~/.amlenvrc >> ~/.bashrc
```

Operationalize the Image Classification Model

For this sample, you are using a pretrained Cognitive Toolkit imagine classification model (Resnet). To Operationalize the model, you use the Azure Machine Learning service create command and supply three files:

- A model file.
- A driver file which contains functions to initialize and run the web service. The initialization function loads the model and defines the inputs and outputs. The run function takes the input as a JSON object and passes it to the web service.
- A resource file that contains text identifying classifications of the images.

The files for the sample were created for a blog post (https://github.com/ilkarman/Blog/blob/master/rndm/ACS%20Deploy.ipynb) by Ilia Karmanov, a Data Scientist with Microsoft.

From the blog post, download the following files:

- The model file: https://migonzastorage.blob.core.windows.net/deep-learning/models/cntk/imagenet/ResNet_152.model

(https://migonzastorage.blob.core.windows.net/deep-learning/models/cntk/imagenet/ResNet_152.model)
- The Resource file: https://ikcompuvision.blob.core.windows.net/acs/synset.txt (https://ikcompuvision.blob.core.windows.net/acs/synset.txt)
- The driver file: https://ikcompuvision.blob.core.windows.net/acs/driver.py (https://ikcompuvision.blob.core.windows.net/acs/driver.py)

You must update the driver file to reference the correct model.

Update the load model call to load the ResNet_152 model as follows:

```
trainedModel = load_model('ResNet_152.model')
```

The _152 model uses larger images for the training and classification.

Next you will operationalize the model as a web service to which you can submit images for classification:

1. Sign in to the DSVM and navigate to the notebooks>azureml folder.
2. Create a new folder named cntkservice.
3. Open the folder and upload the ResNet_152.model, synset.txt, and driver.py file.
4. From the command line, navigate to the notebooks> azureml>cntkservice folder.
5. Run the following commands create the local real-time web service based on the model and uses the cntk run-time:
    1. az ml env local
    2. az ml service create realtime -r cntk-py -f driver.py -m ResNet_152.model -d synset.txt -n cntkservice.

Once the aml create command has completed, the CLI displays information that shows you how to call the service locally. From this, you need to note the port number.

You can verify that your service is running the following command:

```
az ml service list
```

To test the web service you can run the aml service run command and supply it with the base64 representation of an image to compare.

```
az ml service run realtime -n cntkservice -d '{"input": "[\"<base64 image represent
ation>\"]"}'
```

A sample image that you can use is of a British Airways Airbus 380 (https://www.britishairways.com/assets/images/information/about-ba/fleet-facts/airbus-380-800/photo-gallery/240x295-BA-A380-exterior-2-high-res.jpg).

You can convert it to a base64 representation using the services at imagetobase.com

(http://imagetobase64.com/).

Open the port on the DSVM

To call the web service from the mobile app, you must open the port on DSVM to incoming and outgoing internet traffic.

To open the port on DSVM:

1. Sign into the Azure portal (https://portal.azure.com).
2. Open the Network group for your DSVM.
3. Add the port to the rules.

For complete instructions on opening ports on an Azure VM, see Opening ports to a VM in Azure using the Azure portal (https://docs.microsoft.com/en-us/azure/virtual-machines/windows/nsg-quickstart-portal),

Build the mobile app

For this sample, you are going to start from the Xamarin sample Take a Picture and Save Using Camera App (https://github.com/xamarin/recipes/tree/master/android/other_ux/camera_intent/take_ found in the Xamarin sample repository on GitHub.

For more information on the sample, see the commentary in Take a Picture and Save Using Camera App (https://developer.xamarin.com/recipes/android/other_ux/camera_intent/take_a_picture_ in the Xamarin documentation.

Download the sample app from GitHub and open the solution in Visual studio and make the following additions and changes to the code to call the web service.

You can download the completed code (https://aka.ms/cntkimg) from the samples folder in the Machine Learning Operationlization GitHub repository.

Call the classification web service

Add the following method to call the classification web service. The method resizes the bitmap since there is a limitation on the amount of data that can be sent to the web service. Internally, the classification model further resizes the image to 224 by 224 for analysis. It then converts image to its base64 representation, constructs the request, and sends it to the web service.

```
    private async void classifyImage()
    {
        if (App.bitmap != null)
        {
            // Resize the bitmap so that it is small enough to upload to the web se
rvice
            Bitmap bitmap = App.bitmap;
            Bitmap bitmapScaled = Bitmap.CreateScaledBitmap(bitmap, 360, 262, tru
e);
            MemoryStream stream = new MemoryStream();
            bitmapScaled.Compress(Bitmap.CompressFormat.Jpeg, 100, stream);

            // Convert it to base64 string
            byte[] bitmapArrayImage = stream.ToArray();
            string bitmapArrayImageStr = Convert.ToBase64String(bitmapArrayImage);

            // Construct the json request body. Note that the quotes surrounding th
e
            // base64 data must be escaped.
            string jsonRequest =
                      String.Format(@"{{ ""input"":""[\""{0}\""]"" }}", bitmapArray
ImageStr);

            // Send the request
            string json = await FetchAsync.CallWebService("POST",
                            "http://<your host address>:<your service port>", jsonR
equest);

            // Once the call returns with data, display it.
            if (json != null)
            {
                TextView tv = FindViewById<TextView>(Resource.Id.resultsText);
                tv.Text = json;
            }
        }
        else
        {
            TextView tv = FindViewById<TextView>(Resource.Id.resultsText);
            tv.Text = "No image to classify.";
        }
    }
```

Update the call to `FetchAsync.CallWebService` to reference your DSVM and the port on which your web service is accepting requests.

To support the call to the web service, there a few additional updates you must make to the sample code.

Modify the UI

In the resources>layout>Main.axml file, after the `myButton` declaration, add markup for

a button to call the classification web service.

```
<Button
    android:id="@+id/classifyImageButton"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Classify Image" />
```

After the `ImageView` declaration, add a text view field to display the results.

```
<TextView
    android:id="@+id/resultsText"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text=""/>;
```

Update the MainActivity

In MainActivity.cs, add the following using statement to support .NET IO calls:

```
using System.IO;
```

To address conflicts between the .NET and Java IO resources, remove the following using statement.

```
using Java.IO;
```

Next, add explicit `Java.io` resource references to the file definitions in App class.

```
public static class App
{
    public static Java.IO.File _file;
    public static Java.IO.File _dir;
    public static Bitmap bitmap;
}
```

Add explicit resource references to the file creation calls in `CreateDirectoryForPictures` and `TakeAPicture` methods.

```
private void CreateDirectoryForPictures()
{
    App._dir = Java.IO.File(
    Environment.GetExternalStoragePublicDirectory(
        Environment.DirectoryPictures), "CameraAppDemo");
    if (!App._dir.Exists())
...



private void TakeAPicture(object sender, EventArgs eventArgs)
{
    Intent intent = new Intent(MediaStore.ActionImageCapture);
    App._file = new Java.IO.File(App._dir, String.Format("myPhoto_{0}.jpg", Guid.Ne
wGuid()));
...
```

In the `OnActivityResult` method, comment out the line which sets the stored bitmap to null, you are going to keep the bitmap around so that you can send it to the classification web service.

```
if (App.bitmap != null) {
    _imageView.SetImageBitmap (App.bitmap);
    //App.bitmap = null;
}
```

To wire up the classify button to call the classification method, add the following lines in the `OnCreate` method:

```
     Button button2 = FindViewById<Button>(Resource.Id.classifyImageButton);
     button2.Click += delegate { classifyImage(); };
```

Finally, to call the web service, you must add code to make http calls.

Add a new class file called callwebservice and add the following code:

```
using System;
using System.Text;
using System.Threading.Tasks;
using System.Net;
using System.IO;

namespace CameraAppDemo
{
    public static class FetchAsync
    {
        public static async Task<string> CallWebService(string verb, string url, st
ring requestBody = "")
        {
            string jsonDoc = "";
            // Create an HTTP web request using the URL:
            HttpWebRequest request = (HttpWebRequest)HttpWebRequest.Create(new Uri
(url));

            request.Method = verb;
            if (requestBody != "")
            {
                request.ContentType = "application/json";
                ASCIIEncoding encoding = new ASCIIEncoding();

                byte[] data = encoding.GetBytes(requestBody);
                request.ContentLength = data.Length;
                Stream myReqStream = null;
                try
                {
                    myReqStream = request.GetRequestStream();
                    myReqStream.Write(data, 0, data.Length);
                    myReqStream.Close();
                }
                catch (Exception e)
                {
                    Console.Write(e.Message);
                }
            }

            try
            {
                using (WebResponse response = await request.GetResponseAsync())
                {
                    if (response.ContentLength > 0)
                    {
                        // Get a stream representation of the HTTP web response:
                        using (Stream stream = response.GetResponseStream())
                        {
                            // Use this stream to build a JSON document object:
                            jsonDoc = await Task.Run(() =>
                            {
                                byte[] bytes = new byte[response.ContentLength + 1
```

```
0];
                                int numBytesToRead = (int)response.ContentLength;
                                int numBytesRead = 0;
                                do
                                {
                                    // Read may return anything from 0 to 10.
                                    int n = stream.Read(bytes, numBytesRead, 10);
                                    numBytesRead += n;
                                    numBytesToRead -= n;
                                } while (numBytesToRead > 0);
                                return System.Text.Encoding.Default.GetString(byte
s);
                            });
                        }
                    }
                }
            }
            catch (WebException e)
            {
                Console.Write(e.Message);
            }
            return jsonDoc;
        }
    }
}
```

Microsoft

FAQ (http://azure.microsoft.com/en-us/documentation/articles/machine-learning-faq/) Privacy and Cookies (http://www.microsoft.com/privacystatement/en-us/core/default.aspx) Terms of Use (http://azure.microsoft.com/en-us/support/legal/)