Machine Learning Blog

Intelligent Edge: Building a Skin Cancer Prediction App with Azure Machine Learning, CoreML & Xamarin

★★★★★

April 3, 2018 by ML Blog Team // 0 Comments

This post is authored by Anusua Trivedi, Carlos Pessoa, Vivek Gupta & Wee Hyong Tok from the Cloud AI Platform team at Microsoft.

Motivation

AI has emerged as one of the most disruptive forces behind digital transformation and it is revolutionizing the way we live and work. AI-powered experiences are augmenting human capabilities and transforming how we live, work, and play – and they have enormous potential in allowing us to lead healthier lives as well.

Introduction

AI is empowering clinicians with deep insights that are helping them make better decisions, and the potential to save lives and money is tremendous. At Microsoft, the Health NExT project is looking at innovative approaches to fuse research, AI and industry expertise to enable a new wave of healthcare innovations. The Microsoft AI platform empowers every developer to innovate and accelerate the development of real-time intelligent apps on edge devices. There are a couple of advantages of running intelligent real-time apps on edge devices – you get:

- Lowered latency, for local decision making.
- Reduced reliance on internet connectivity.

Imagine environments where there's limited or no connectivity, whether it's because of lack of communications infrastructure or because of the sensitivity of the operations and information involved. There the only alternative to cloud servers are proprietary data centers that cost a lot to set up and maintain. Remote locations that can benefit immensely from artificial intelligence may end up having limited access to AI applications because of their poor connectivity. As IoT moves into more unpredictable environments including disconnected ones, it becomes increasingly more important to support such hybrid environments of cloud and edge computing.

Take our skin cancer detection app as an example. Skin cancer is the most common type of cancer, globally accounting for at least 40% of all cases, and it is much better controlled when detected at an early stage. What if we create a real-time AI app which can quickly suggest whether or not a given individual needs to seek medical help? Such an app would flag a set of images which, in turn, could help doctors become more efficient by focusing most of their energies on their most critical patients.

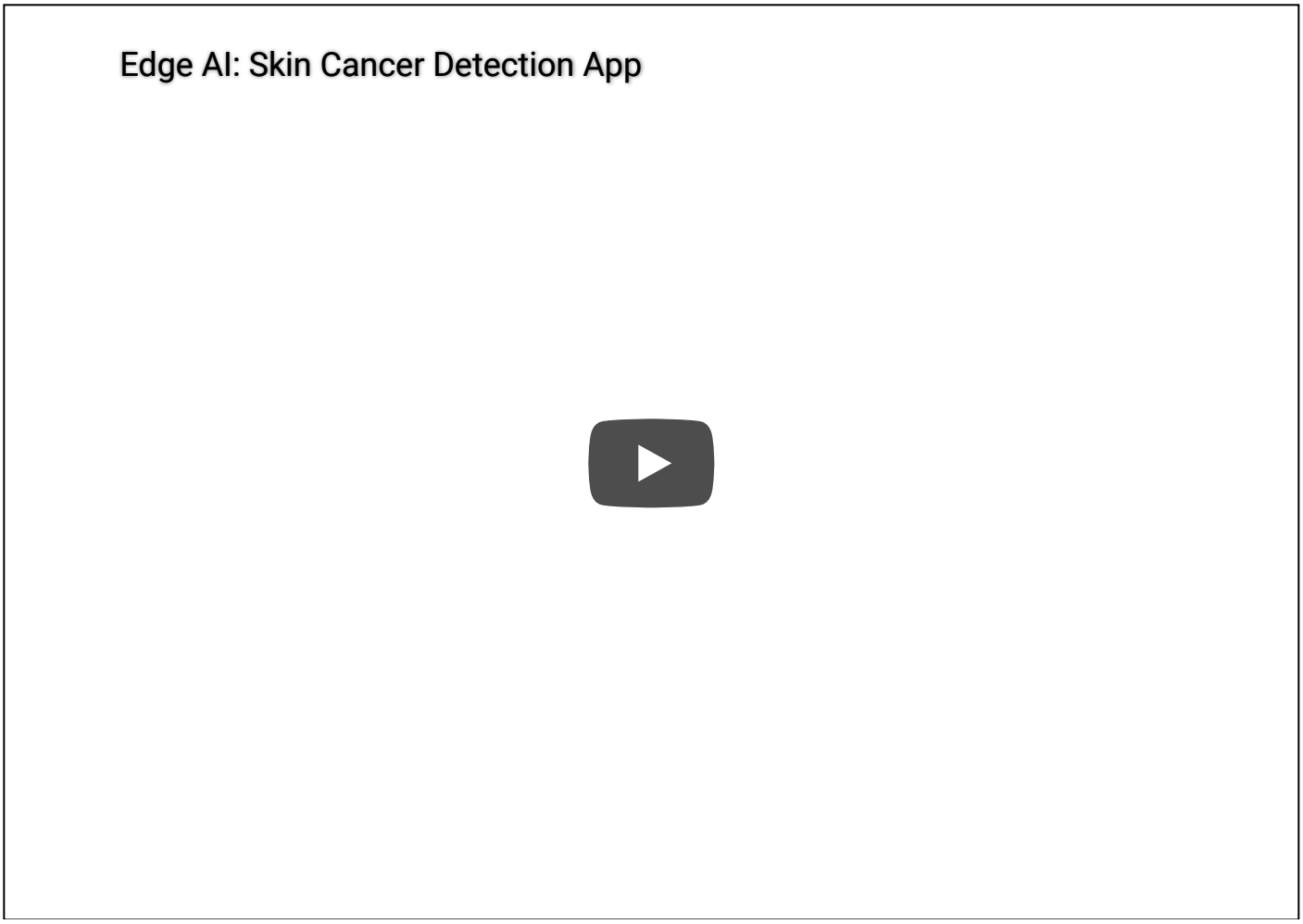
The model/application being proposed here is intended for research and development use only. The model/application is *not* intended for use in clinical diagnosis or decision-making or any other clinical use. The performance characteristics of the application for clinical use has not been established.

Dataset and Preprocessing

For this work, we us the ISIC Skin Cancer Research dataset. We have split up the ISIC Dataset for training and testing – 80% of the images for training and 20% of the images for scoring. ISIC dataset contains 2000 images as training data, including 374 “melanoma” images and 254 “seborrheic keratosis” images, with the remaining 1372 being benign images. The training data is provided as a ZIP file containing dermoscopic lesion images in JPEG format and a CSV file with some clinical metadata for each image. Here we use only JPEG images for training our AI model. The ISIC dataset has far fewer melanoma examples than seborrheic keratosis and nevus. Only about 20% of the default ISIC dataset is malignant, a total of 374 images. The skewed distribution has a big impact on how we judge our classifier and how we train it. We apply different augmentation techniques such as rotation, cropping etc. to balance the dataset.

Building an Intelligent Skin Cancer Prediction App

Listed below are the steps we took to build our intelligent app. To access this code, please refer to this GitHub link. For additional details regarding this app, be sure to check out the video clip below.



1. *Training the AI model for the app:*
   We build the AI Model using the Microsoft Azure Machine Learning Workbench. Azure ML is a cross-platform application, which makes the modelling and model deployment process much faster versus what was possible before. We create a deep learning model using open-source packages supported in Azure ML. We use Keras with a Tensorflow backend to build the model.

   We first tried the transfer learning approach for training the AI models. Applying transfer learning on a standard ImageNet pretrained ResNet-50 model was not giving good results on smaller domain-specific datasets such as ISIC. As seen in Figure 1, we used a smaller 2 convolution layer network with a sigmoid classifier. We did some hyperparameter tuning and Relu activation with Adam optimizer worked best for this model. This model, when trained on ISIC dataset, gives ~97% training accuracy on our training set and ~89% test accuracy on our test set.

2. *Deploy trained AI model as an intelligent app:*
   We want to run this trained model on our iPhone. We use CoreML to convert the trained Keras model to an iPhone compatible format (CoreML brings machine learning to iOS). Apps can take advantage of trained machine learning models to perform all sorts of tasks, from problem solving to image recognition. We pip installed the CoreML package in our Azure ML environment. We can run the CoreML converter on the Azure ML Workbench and create an mlmodel (see Figure 2 below).

3. *Using Xamarin to develop an intelligent application:*

A key benefit of Xamarin is that the UI uses native controls on each platform – this helps us create apps that are indistinguishable from other iOS or Android apps. We start with the sample Xamarin app at this GitHub link. Next, we change the name of the model in the view controller file and load the compiled CoreML model. In the view controller file, we change the result extraction function to output the messages we want the app to spit out (see Figure 3). By changing only the highlighted lines of code in our sample Xamarin app, we can run any AI model on the phone.

At the end of these steps, we have our intelligent skin cancer prediction app for iOS (see Figure 4).

```
from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D
from keras.layers import Dropout, Flatten, Dense
from keras.models import Sequential

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=(224, 224, 3)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(2, activation='sigmoid'))

model.summary()


#Compile the Model
opt = keras.optimizers.Adam(lr=0.0001, beta_1=0.9, beta_2=0.999, epsilon=1e-08,
decay=0.0)
model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
```

Figure 1. Training DNN in AML

```
import coremltools
import h5py
scale = 1/255.
output_labels = ['benign', 'malignant']
coreml_model = coremltools.converters.keras.convert('path_to_model.hdf5')

coreml_model.author = 'anusua'
coreml_model.short_description = 'Model to classify skin cancer'
print(coreml_model)
coreml_model.save(path_to_model.mlmodel')
```

Figure 2. Convert Keras model to CoreML in AML

```
public VNRequest[] ClassificationRequest
        {
            get
            {
                if (model == null)
                {
                    var modelPath = NSBundle.MainBundle.GetUrlForResource("CoreML_Model_Name",
"mlmodelc");

                    NSError createErr, mlErr;
                    var mlModel = MLModel.Create(modelPath, out createErr);
                    model = VNCoreMLModel.FromMLModel(mlModel, out mlErr);
                }
                if (classificationRequestAry == null)
                {
                    var classificationRequest = new VNCoreMLRequest(model,
handleClassification);
                    classificationRequestAry = new VNRequest[] { classificationRequest };
                }
                return classificationRequestAry;
            }
        }


public void handleClassification(VNRequest request, NSError error)
        {
            _
            else
            {
                unknownCounter = 0;
                var className = best.Identifier.Trim();
                Console.WriteLine("Identified: " + className);

                DispatchQueue.MainQueue.DispatchAsync(() =>
                {
                    var confidenceString = best.Confidence.ToString("P0", new NumberFormatInfo
{ PercentPositivePattern = 1 });
                    if (className == "benign")
                    {
                        bubbleLayer.BackgroundColor = new UIColor(0, 1, 217 / 255, 1).CGColor;
                        bubbleLayer.String = $"All clear ({confidenceString})";
                    }
                    else
                    {
                        bubbleLayer.BackgroundColor = new UIColor(1, 0, 217 / 255, 1).CGColor;
                        bubbleLayer.String = $"See doctor! ({confidenceString})";
                    }
                });
            }
        }
```

Figure 3. Intelligent Xamarin app

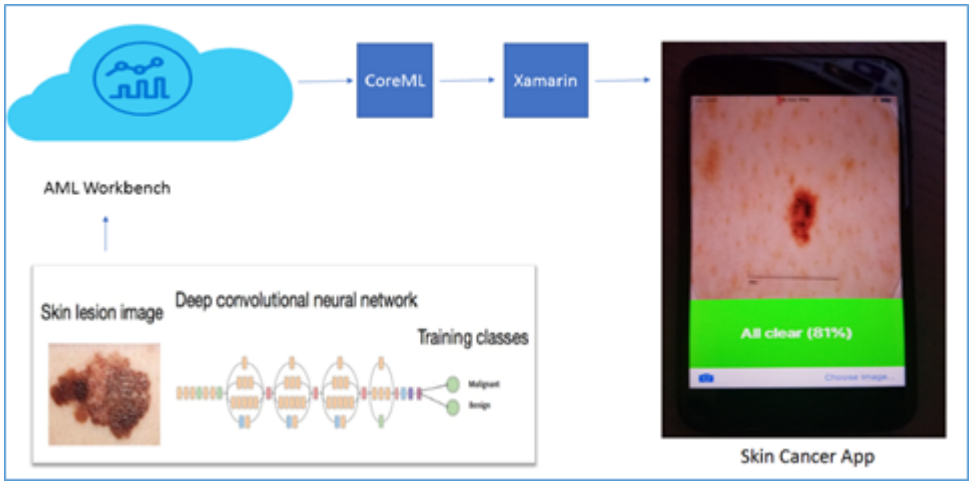


Figure 4. Skin Cancer Prediction App Architecture

Conclusion

In this blog post, we showed how to use Azure Machine Learning to train and test an AI model and create an intelligent iOS app. Such apps can help with time-critical decisions at the edge, referring to the cloud only if more intensive computation or historical analysis is needed.

We hope you are inspired to use the combination of intelligent cloud and intelligent edge in your own scenarios and build a bunch of cool AI-powered apps for your business.

Anusua, Carlos, Vivek & Wee Hyong
(You can email Anusua at antriv@microsoft.com with any questions pertaining to this post.)

Related Links

Azure Big Data Blog

Azure IoT Suite

Azure Machine Learning

Cortana Intelligence Suite

Cortana Intelligence Gallery

Data Science at Microsoft Research

SQL Server Blog

Python Engineering at Microsoft