# Deploying our model

* From our project in customVisionAI make sure our project is open and go to the performance tab
* On the left we can see the iterations of our model. These are the different model versions. We should have only one at this time. Make sure it is selected and press “publish” at the top of the screen.

A screenshot of a cell phone

Description automatically generated

* In the popup window change the model name to **Production** and select the prediction resource to point to the Azure service we setup in lab 1.

# Calling the service

* The first thing we need is the information to call the prediction service. We can find this in the Custom Vision Portal at customvision.ia with our project open. In the Performance tab with the published iteration selected, press on the “Prediction URL” link at the top. From this we need the following information:
  + The First part of the URL before /customvision
  + The Prediction Key
  + The project ID from the URL
  + The name we gave to the deployment (should be Production) This can also be found in the URL
* In Visual Studio open the Lab2 project in the project folders.
* Open the Constants\AzureML.cs file. Paste the four items collected above into the following variables respectively:
  + PredictionEndpoint
  + PredictionAPIKey
  + ProjectId
  + PublishedId
* Ensure the following Nuget package has been added:
  + Microsoft.Azure.CognativeServices.Vision.CustomVision.Prediction
* Implement the following code in the GetEvaluationResult method of the HomeController class. This takes in a string for the location of an image and returns information from our model.

// Create a client object to communicate with our service

ICustomVisionPredictionClient client = new CustomVisionPredictionClient()

{

ApiKey = AzureML.PredictionAPIKey,

Endpoint = AzureML.PredictionEndpoint

};

// Load our image into a stream

var url = "https://" + HttpContext.Request.Host.ToString() + "/" + imageLocation;

var request = WebRequest.Create(url);

using (var response = await request.GetResponseAsync())

{

using (var stream = response.GetResponseStream())

{

// Classify the image

var results = await client.ClassifyImageAsync(AzureML.ProjectId, AzureML.PublishedId, stream);

// Get the prediction with the highest score

// Ensure the probabily is greater than 50%

var topPrediciton = results.Predictions.FirstOrDefault(p => p.Probability > .5);

// Return information about our prediction

// If no valid prediction return an empty "unknown" result

return topPrediciton != null ? new EvaluationResult

{

Confidence = Convert.ToInt32(topPrediciton.Probability),

ImageType = topPrediciton.TagName

} : new EvaluationResult();

}

}

* Try running the project and press the “Evaluate Images” button

One thing to notice, it likely classified the last image as a palm tree. It is not, it is a Joshua tree, but looks similar.

# Reinforcement Learning

* We now have a situation where a classification is done incorrectly. We want the user to be able to update the model with this new information.
* Go to the Custom Vision Portal at customvision.ia with our project open. Click on the gear icon in the upper right. Copy the following two pieces of information:
  + Key
  + Endpoint
* Open up the Lag2 project.
* Open the Constants\AzureML.cs file. Paste the two items collected above into the following variables respectively:
  + TrainingAPIKey
  + TrainingEndpoint
* Implement the following code in the IncorrectImagev method of the HomeController class. This takes in a string for the location of an image and adds it to the model data as an unknown negative result. The model is then updated and replaced as the current model.

// Create a client object to communicate with our service

var client = new CustomVisionTrainingClient

{

ApiKey = AzureML.TrainingAPIKey,

Endpoint = AzureML.TrainingEndpoint

};

// Get the list of tags in our model

var tags = await client.GetTagsAsync(AzureML.ProjectId);

// Find the unknown tag we want to add the image to

var tag = tags.Single(t => t.Name == "Unknown");

// Load our image into a stream

var url = "https://" + HttpContext.Request.Host.ToString() + "/" + imageLocation;

var request = WebRequest.Create(url);

using (var response = await request.GetResponseAsync())

{

using (var stream = response.GetResponseStream())

{

// Add the image to the unknown tag

await client.CreateImagesFromDataAsync(AzureML.ProjectId, stream, new List<Guid>() { tag.Id });

// Queue training to create another iteration of the model

var iteration = await client.TrainProjectAsync(AzureML.ProjectId, "Regualar");

// Wait until training is complete

do

{

iteration = await client.GetIterationAsync(AzureML.ProjectId, iteration.Id);

} while (iteration.Status == "Training");

// Fine the currently published iteration

var iterations = await client.GetIterationsAsync(AzureML.ProjectId);

var publishedIteration = iterations.SingleOrDefault(i => i.PublishName == AzureML.PublishedId);

// Unpublish the current iteration

await client.UnpublishIterationAsync(AzureML.ProjectId, publishedIteration.Id);

// Publish our new model under the same name

await client.PublishIterationAsync(AzureML.ProjectId, iteration.Id, AzureML.PublishedId, publishedIteration.OriginalPublishResourceId);

// Ensure the publish process is complete

do

{

iteration = await client.GetIterationAsync(AzureML.ProjectId, iteration.Id);

} while (iteration.PublishName != AzureML.PublishedId);

}

}

* Run the project and remove the last image
* See the new classifications

There is a chance that the first palm tree image will now end up being unknown. Why might this happen? What are the advantages and pitfalls of having anyone able to update the model and immediately training and publishing a new version? Is having the response from the web server wait for the model to be trained a good idea? How else could it be done?