Custom Vision

In this hands-on lab, you explore the Custom Vision functionality of Azure Cognitive Services. This service allows you to define your own image analysis models that can recognize the attributes and classify your images according to your own business needs, rather than according to some generic rules.

Contents

[Create a custom vision model 2](#_Toc2011744)

[Download the model images 2](#_Toc2011745)

[Get started with Custom Vision 2](#_Toc2011746)

[Create a Custom Vision project 4](#_Toc2011747)

[Validating the model 9](#_Toc2011748)

[Improving the model 10](#_Toc2011749)

[Preparing for web services use 11](#_Toc2011750)

[Use Custom Vision from Business Central 13](#_Toc2011751)

[Create and configure a new workspace 13](#_Toc2011752)

[Create table and page extensions to set up Custom Vision 13](#_Toc2011753)

[Solution: TableExtension 50136 Image Analysis Setup Ext.al 13](#_Toc2011754)

[Solution: PageExtension 50136 Image Analysis Setup Ext.al 14](#_Toc2011755)

[Create table to contain temporary Custom Vision tags 15](#_Toc2011756)

[Solution: Table 50136 Custom Vision Tag.al 15](#_Toc2011757)

[Create codeunit to invoke Custom Vision web service 16](#_Toc2011758)

[Solution: Codeunit 50136 Custom Vision Management.al 21](#_Toc2011759)

[Customize Item Card page 26](#_Toc2011760)

[Solution: PageExtension 50137 Item Card Extension.al 26](#_Toc2011761)

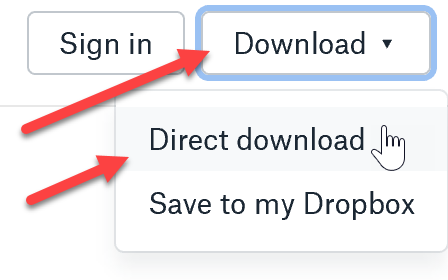
[Test Custom Vision integration 28](#_Toc2011762)

# Create a custom vision model

In this exercise you will create a new Custom Vision model, train that model with pictures, and then validate that model to see how well it works, where are its strengths and weaknesses, and then train it more to improve its performance.

## Download the model images

1. Open your browser and navigate to <http://bit.ly/RedCarpetCustomVision>  
   Hint: this URL is case-sensitive!
2. In the upper-right corner, click **Download > Direct Download**:



1. Save the incoming .zip file on your computer.
2. Unpack the .zip file. It will contain the following content:

Train: the folder contains pictures you’ll use to train your custom vision model

Validate: the folder contains pictures you’ll use to validate your custom vision model

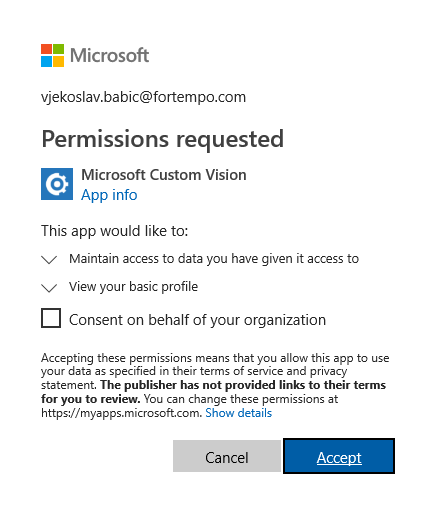
Inside the Train folder, there are 426 images are organized according to the following rules:

* There are 17 distinct Lego figurine identities, marked with **A** thru **Q**.
* The first word indicates whether the figurine represents a **Human** or an **Animal**.
* Humans are further classified as **adult** or **child** (the second word in the file name).
* Humans are further classified as **male** or **female** (the third word in the file name).
* Animals are classified as **horse**, **goat**, **dog**, **pig**, **sheep**, or **bear** (the second word in the file name).
* All figurines have a pose **Standing**, **Sitting**, or **Lying** (the first word after the dash).
* Some figurines have the setting of **city** or **nature** (the second word after the dash).

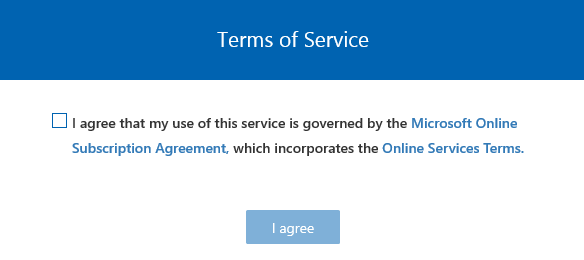
The validate folder has 150 unorganized images of 24 additional identities, with three additional settings, one additional pose, and a number of identities from the Train folder, but in unfamiliar poses or settings.

## Get started with Custom Vision

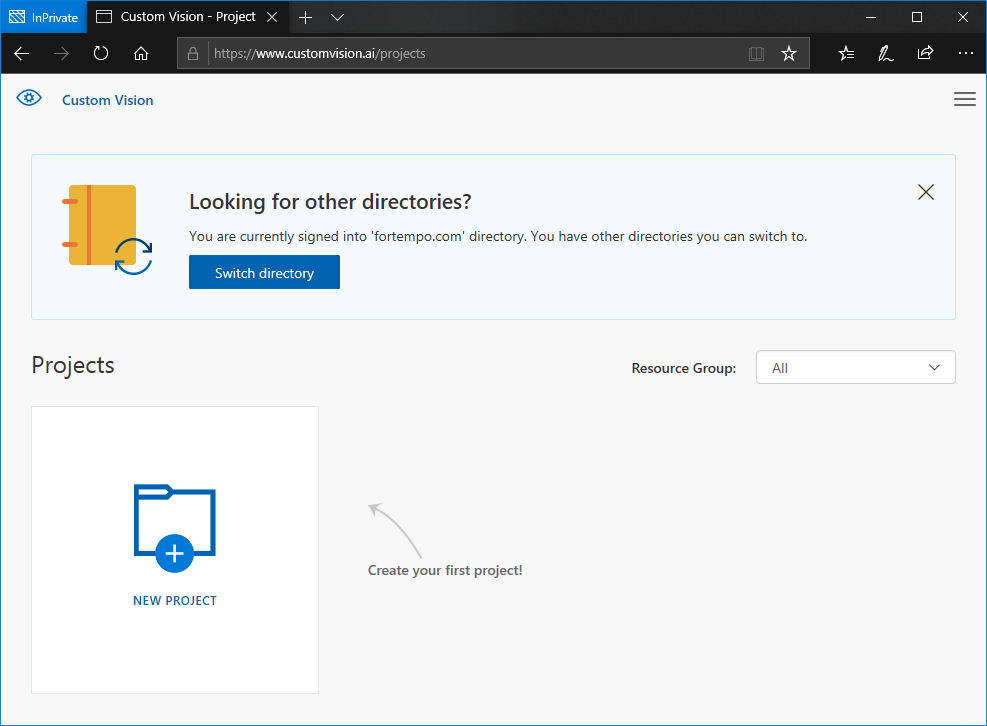
1. In your browser, navigate to <https://www.customvision.ai>
2. Click **Sign in**.
3. Use your Microsoft or Office 365 credentials.
4. If this is the first time you are using Custom Vision with this account, you’ll be presented with permissions dialog:



1. **Do not** select the **Consent on behalf of your organization** checkbox (unless you want to provide an organization-wide consent for Microsoft Custom Vision to access any pictures they upload or access their profile information).
2. Click **Accept**.
3. Click **OK**. The **Terms of Service** dialog shows:

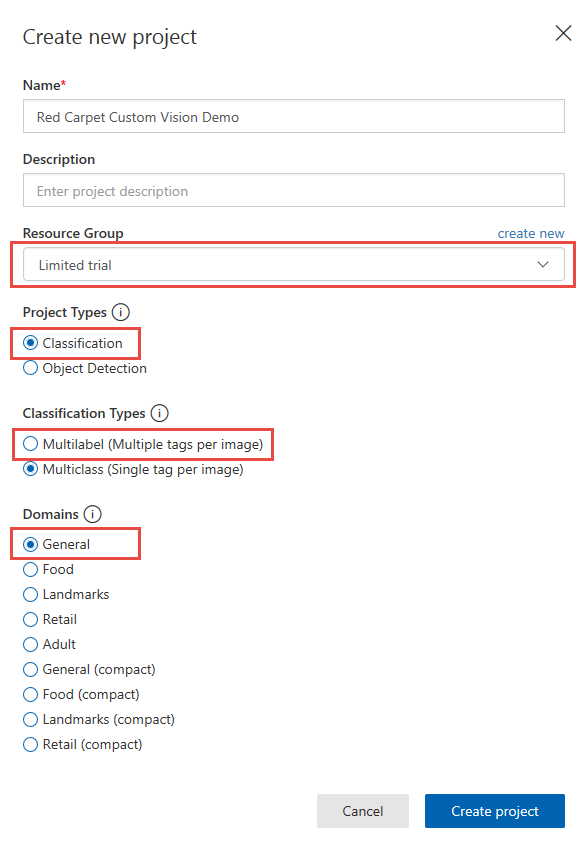


1. Select the **I agree** checkbox and then click the **I agree** button. You will see the Custom Vision dashboard.

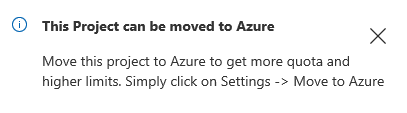


## Create a Custom Vision project

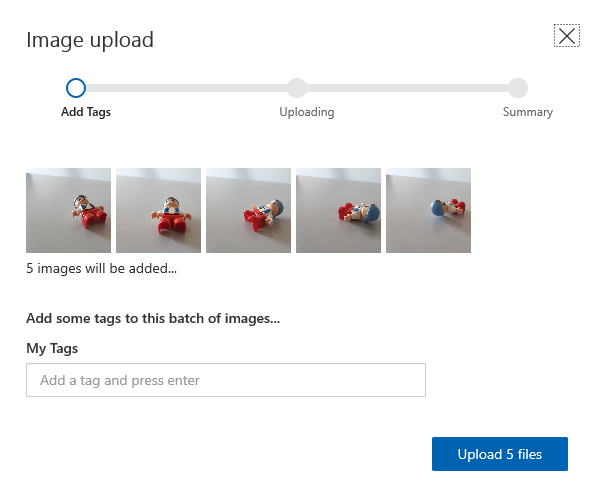
1. Click **New Project**. The **Create new project** dialog opens.
2. Configure the Create new project dialog as follows:
   1. In the **Name** field, enter “Red Carpet Custom Vision Demo”.
   2. Leave the **Description** field empty.
   3. Make sure that **Resource Group** is set to **Limited Trial**.
   4. Make sure that **Project Type** is set to **Classification**.
   5. Set **Classification Types** to **Multilabel (Multiple tags per image)**.
   6. Make sure that **Domains** is set to **General**.



1. Click **Create project**.
2. If you see the following notification, close it.



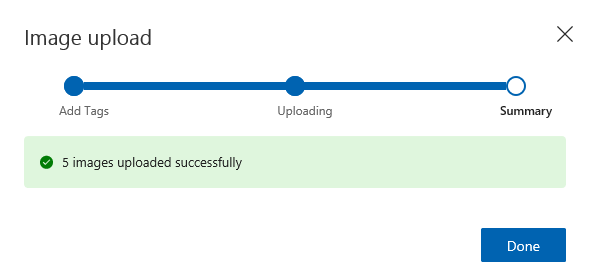
1. Under the **Looks like you don’t have any images here** caption, click **Add images**.
2. Browse to the **Train** folder.
3. Select five arbitrary pictures of **A Human child male – Lying** and click **Open**. The **Image upload** dialog shows:



1. In the **My Tags** field, enter “Human” (without the quotes!) and press Enter. Then use the same method to enter the following tags: **Child**, **Male**, **Lying**.

Hint: The tag system is case-insensitive. However, use the same case for all tags you enter.

1. Click **Upload 5 files**. The Image upload dialog will update:



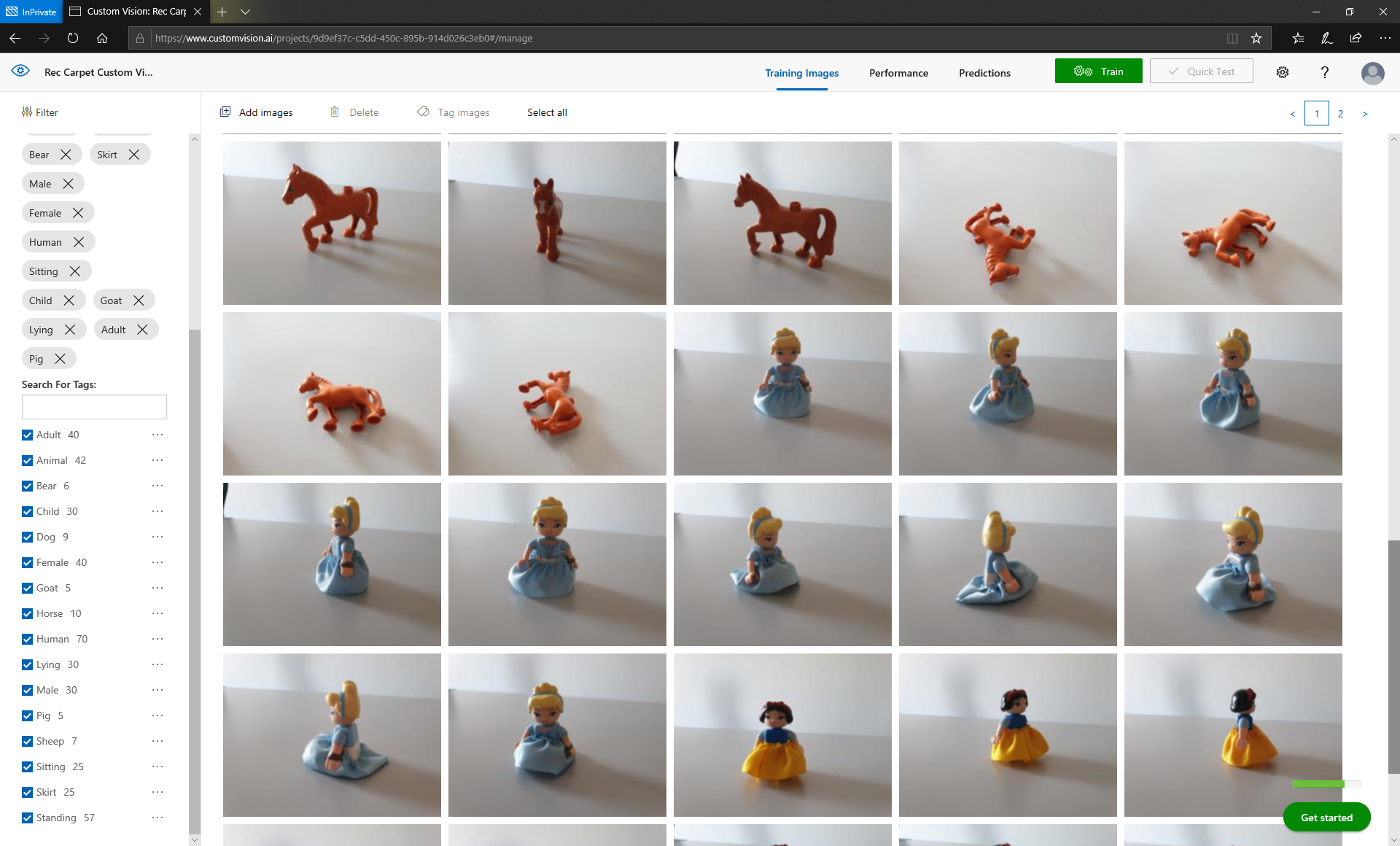
1. Click **Done**.
2. In the upper action bar, click **Add images**:



1. Repeat steps 6 thru 11 to add more images to your project. We recommend that you use identities A, B, C, D, I, K, L, M, N, O, and P. Select 5 (or all available, if there are fewer than 5) arbitrary images of each identity in each pose, and for now do not select images with city or nature setting. Tag the images according to their content, with the following tags:
   1. Human
   2. Adult
   3. Child
   4. Male
   5. Female
   6. Standing
   7. Sitting
   8. Lying
   9. Animal
   10. Horse
   11. Goat
   12. Dog
   13. Pig
   14. Sheep
   15. Bear

With identities D and I, use an additional tag: Skirt.

1. When you have completed step 12, you will have a workspace with more setup similar to this:

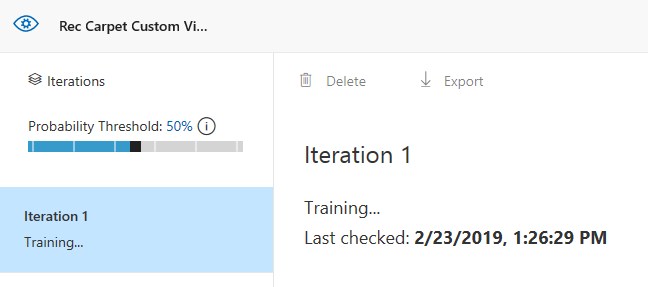


Hint: Pay attention to the distribution of tags – there will be a disbalance between different tags. This is not optimal, and for the model to provide optimal performance, you need a balanced number of tags, with minimum of 50 images per tag.

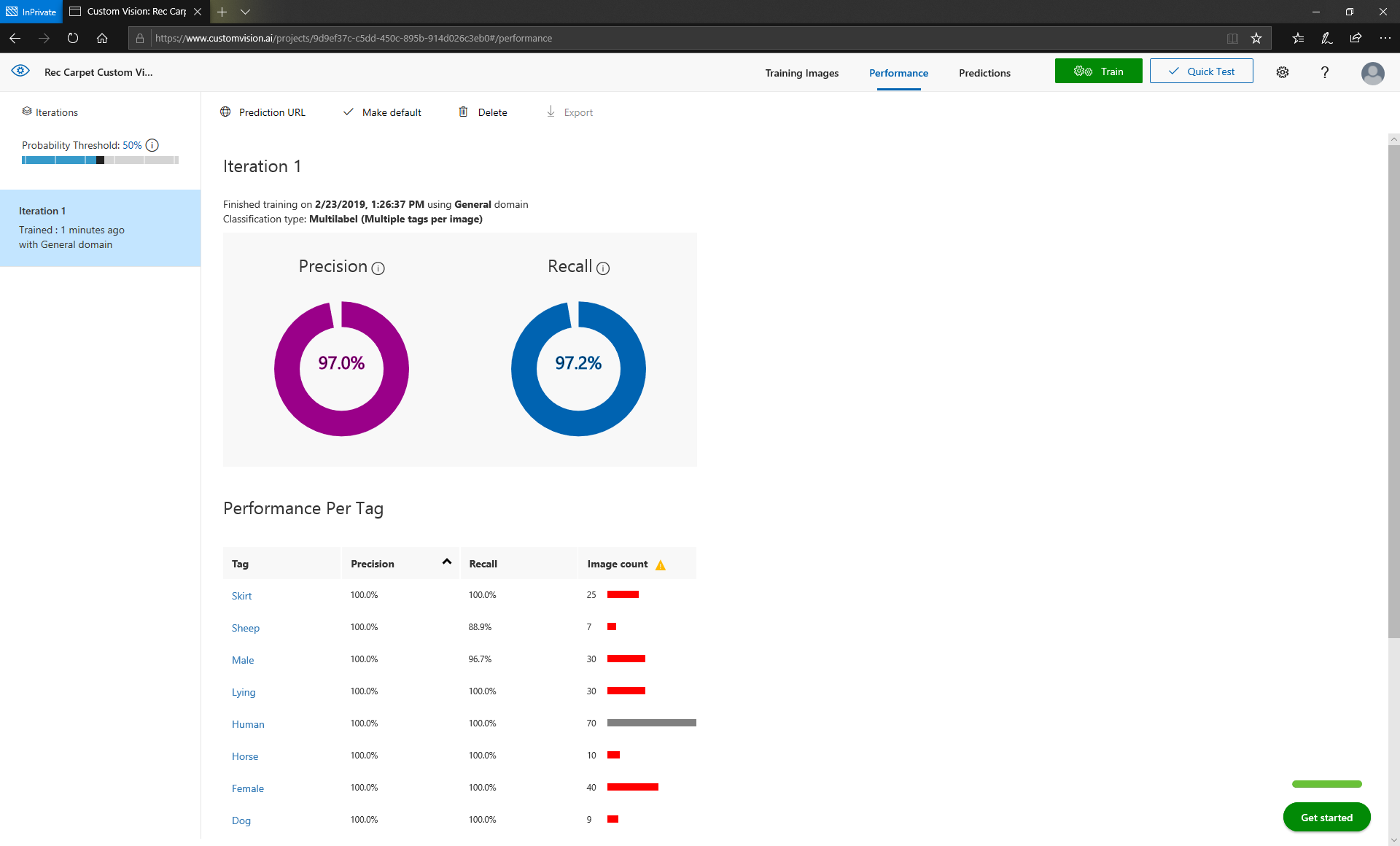
1. Click **Train**.



The model starts training and shows the following info:



1. After the model training completes, the **Performance** tab will open and show the following information:



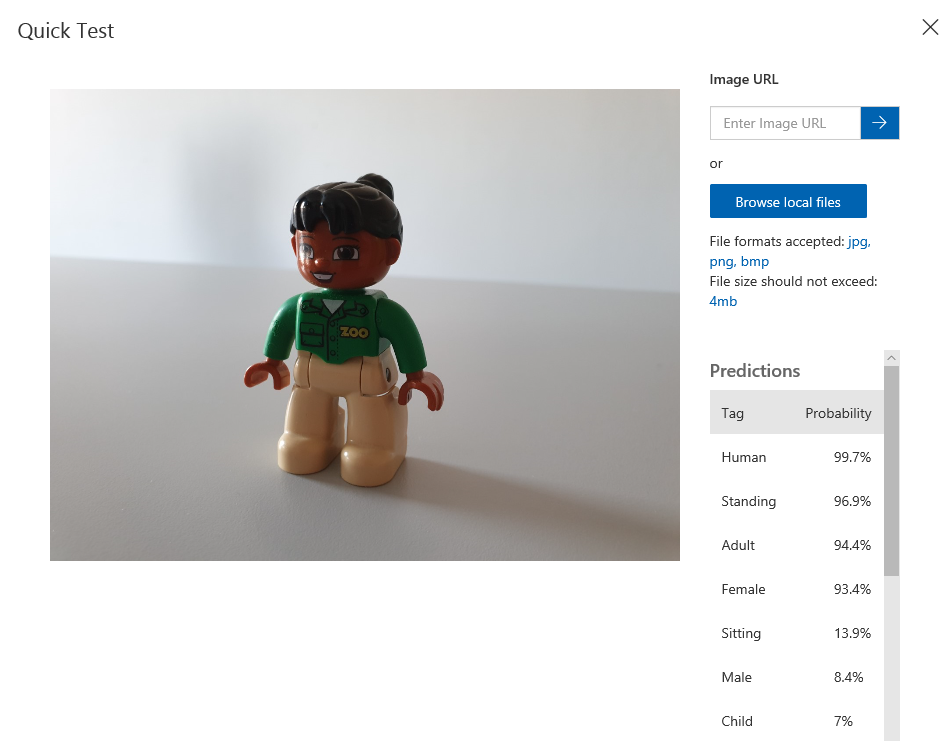
Pay attention to the the Performance Per Tag section, it shows you all your tags with their statistics, and recommendations. Tags that are marked with red in the Image count column will perform poorly as compared to tags that are gray. Also, tags with longer bars will be correctly recognized more often than tags with shorter bars.

## Validating the model

1. Click **Quick Test** to test your model.



1. In the **Quick Test** dialog, click **Browse local files**.
2. Go to the **Validate** folder, select image **Validation (32).jpg**, and then click **Open**. The model validates your image, and shows you results like this:



1. Click Browse local files again, and repeat step 18 for the following images:
   1. Validation (11).jpg
   2. Validation (19).jpg
   3. Validation (23).jpg
   4. Validation (39).jpg
   5. Validation (80).jpg
   6. Validation (142).jpg

If you used these images, you will notice that the model has accurately distinguishes between humans and animals, and can accurately identify whether the character is standing, sitting, or lying. It has a fair guess of gender but does not accurately tell adults from children. Also, different backgrounds obviously don’t influence the accuracy. Also, if you used the tag **Skirt**, and used to test the model with images of characters with skirts, you will see that the model doesn’t provide any false positives, and that it recognizes the presence of a skirt in the image.

1. Repeat tests with more images and take a note of any inaccurate tags. For each inaccurate tag, note the tags and the image that was inaccurately tag. You will come back to these images and tags later after you improve and re-train the model.

## Improving the model

1. Click **Training images**.

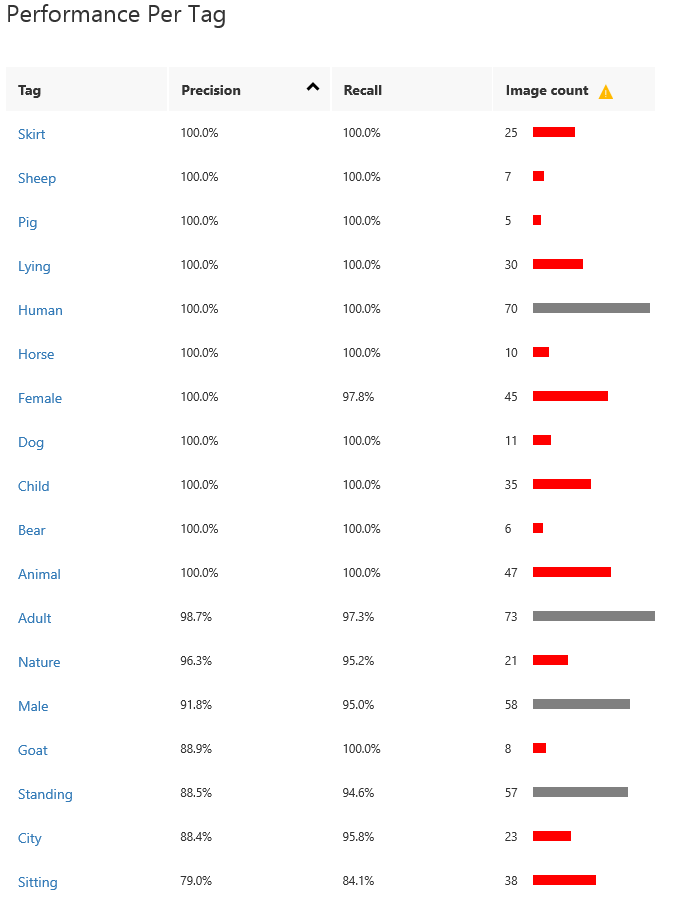


1. Upload and tag additional images of any characters with either nature or city setting. Tag those images only with the following tags:
   1. **City** or **Nature**.
   2. For humans specify **Adult** or **Child**, and **Male** or **Female**, but do not specify **Human**.
   3. For any characters specify pose if it is either **Sitting** or **Lying** (do not tag additional images with **Standing**).
   4. For animals, specify **Animal**, and specify which one.

If you tag images like this, you will improve statistics for certain tags. Any tags that you noticed to be consistently inaccurately recognized, make sure to include the in this iteration.

1. Click **Train**.

After Iteration 2 completes, the **Performance Per Tag** will show more tags as gray. These tags should exhibit better accuracy performance than in the first iteration. Also, a few tags have longer red bars, and the longer the bar, the more accurate the tag will perform.

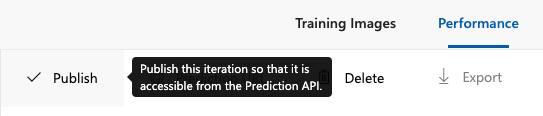


1. Repeat steps 17 and 18 for the following images:
   1. Validation (45).jpg
   2. Validation (47).jpg
   3. Validation (48).jpg
   4. Validation (52).jpg
   5. Validation (59).jpg
   6. Validation (70).jpg
   7. Validation (87).jpg

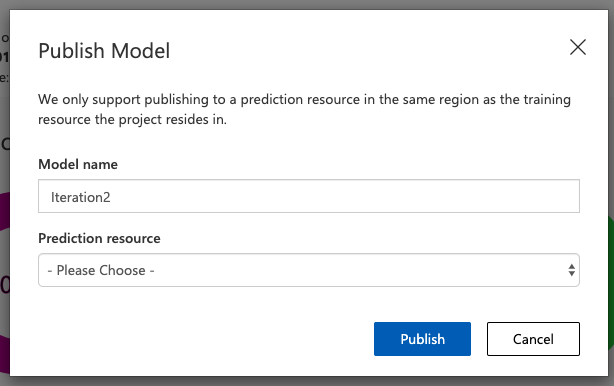
You can notice that city and nature are recognized, but not always accurately, and that recognition of other tags has probably improved. To further improve the model, you would have to upload much larger number of images and try to keep them as balanced as possible.

## Preparing for web services use

1. Click the Perfomance icon and then Publish.



1. Fill the Iteration name and choose your Azure resource group, where Web service will be hosted.



1. Click on Prediction URL. Copy the **url** under “If you have an image file… ” section. Copy the **Prediction Key** value. You will use them later when connecting to the Custom Vision web service from Business Central.

# Use Custom Vision from Business Central

In this exercise you will be a toy store owner that sells Lego figurines, and you’ll use your Custom Vison model to help you organize your item catalog and aid you in finding items for your customers.

You’ll start by building a Custom Vision extension to provide the necessary functionality. Then you’ll configure your application to support Custom Vision tagging, and finally you’ll test your extension and data.

## Create and configure a new workspace

1. Create a new AL workspace and name it **Custom Vision Demo**.
2. In the app.json file, configure the ID range from 50136 to 50139.
3. In the launch.json file, set **startupObjectType** to “Page”, and **startupObjectId** to 31.

## Create table and page extensions to set up Custom Vision

1. Create a new file and name it TableExtension 50136 Image Analysis Setup Ext.al.
2. In this file, declare tableextension 50136 “Image Analysis Setup Ext.” that extends the **Image Analysis Setup** table.
3. Declare the fields section, and in it, declare the following new fields:

|  |  |  |
| --- | --- | --- |
| ID | Field | Type |
| 50136 | Custom Vision API URI | Text[250] |
| 50137 | Custom Vision API Key | Text[250] |
| 50138 | Tag Confidence Threshold | Decimal  Set InitValue to 0.5 |

1. Create a new file and name it PageExtension 50136 Image Analysis Setup Ext.al.
2. In this file, declare pageextension 50136 “Image Analysis Setup Ext.” that extends the **Image Analysis Setup** page.
3. Declare the layout section, and inside it add a new group named **Custom Vision** after the **General** group.
4. In the **Custom Vision** group, add the **Custom Vision API URI**, **Custom Vision API Key**, and **Tag Confidence Threshold** fields.

### Solution: TableExtension 50136 Image Analysis Setup Ext.al

tableextension 50136 "Image Analysis Setup Ext." extends "Image Analysis Setup"

{

fields

{

field(50136; "Custom Vision API URI"; Text[250])

{

Caption = 'Custom Vision API URI';

}

field(50137; "Custom Vision API Key"; Text[250])

{

Caption = 'Custom Vision API Key';

}

field(50138; "Tag Confidence Threshold"; Decimal)

{

Caption = 'Tag Confidence Threshold';

InitValue = 0.5;

}

}

}

### Solution: PageExtension 50136 Image Analysis Setup Ext.al

pageextension 50136 "Image Analysis Setup Ext." extends "Image Analysis Setup"

{

layout

{

addafter(General)

{

group("Custom Vision")

{

Caption = 'Custom Vision';

field("Custom Vision API URI"; "Custom Vision API URI")

{

ApplicationArea = All;

}

field("Custom Vision API Key"; "Custom Vision API Key")

{

ApplicationArea = All;

}

field("Tag Confidence Threshold"; "Tag Confidence Threshold")

{

ApplicationArea = All;

}

}

}

}

}

## Create table to contain temporary Custom Vision tags

1. Create a new file and name it Table 50136 Custom Vision Tag.al.
2. In this file, declare table 50136 "Custom Vision Tag".
3. Declare the fields section and in it declare the following fields:

|  |  |  |
| --- | --- | --- |
| ID | Field | Type |
| 1 | Entry No. | Integer |
| 2 | Attribute | Text[250] |
| 3 | Tag | Text[250] |
| 4 | Confidence | Decimal |

1. Declare the keys section and in it declare a primary key, and set it to the Entry No. field.

### Solution: Table 50136 Custom Vision Tag.al

table 50136 "Custom Vision Tag"

{

Caption = 'Custom Vision Tag';

fields

{

field(1; "Entry No."; Integer)

{

Caption = 'Entry No.';

}

field(2; Attribute; Text[250])

{

Caption = 'Attribute';

}

field(3; Tag; Text[250])

{

Caption = 'Tag';

}

field(4; Confidence; Decimal)

{

Caption = 'Confidence';

}

}

keys

{

key(Primary; "Entry No.") { }

}

}

## Create codeunit to invoke Custom Vision web service

1. Create a new file and name it Codeunit 50136 Custom Vision Management.al.
2. In this file, declare codeunit 50136 "Custom Vision Management". Make it single-instance.
3. Declare the following global variables:

|  |  |
| --- | --- |
| Variable | Type |
| ListType | List of [Text] |
| ListAge | List of [Text] |
| ListGender | List of [Text] |
| TypeIDs | List of [Integer] |
| AgeIDs | List of [Integer] |
| GenderIDs | List of [Integer] |
| TypeAttributeID | Integer |
| AgeAttributeID | Integer |
| GenderAttributeID | Integer |
| Initialized | Boolean |

1. Create a new local function named **GetAttributeValueID**. It returns a value of type Integer, and receives these parameters:

|  |  |
| --- | --- |
| Parameter | Type |
| AttrID | Integer |
| AttrVal | Text |

1. Declare the following local variable for the **GetAttributeValueID** function:

|  |  |
| --- | --- |
| Variable | Type |
| ItemAttrVal | Record “Item Attribute Value” |

1. Write the following logic for the **GetAttributeValueID** function:
   1. Try to find the first **Item Attribute Value** record where the **Attribute ID** field value matches the **AttrID** parameter and the **Value** field value matches the **AttrVal** parameter.
   2. If this record cannot be found:
      1. Reset any filters on **Item Attribute Value**.
      2. Try to find the last record, with ordering by the ID field, ascending.
      3. Initialize a new **Item Attribute Value** record.
      4. Set **Attribute ID** to the value of the **AttrID** parameter.
      5. Increase **ID** by 1.
      6. Set **Value** to the value of the **AttrVal** parameter.
      7. Insert the **Item Attribute Value** record.
   3. Exit the function with the value of the **ID** field from the **Item Attribute Value** as the return value.
2. Create a new local function named **GetAttributeID**. It returns a value of type Integer, and receives these parameters:

|  |  |
| --- | --- |
| Parameter | Type |
| Attr | Text |
| Values | List of [Text] |
| ValueIDs | List of [Integer] |

1. Declare the following local variables for the **GetAttributeID** function:

|  |  |
| --- | --- |
| Variable | Type |
| ItemAttr | Record “Item Attribute” |
| AttrValue | Text |

1. Write the following logic for the **GetAttributeID** function:
   1. Try to find the first **Item Attribute** record where the **Name** field value matches the **Attr** parameter.
   2. If this record cannot be found:
      1. Reset any filters on the **Item Attribute** table.
      2. Try to find the last record.
      3. Initialize a new **Item Attribute** record.
      4. Increase the **ID** by 1.
      5. Set **Name** to the value of the **Attr** parameter.
      6. Set **Type** to Option.
      7. Insert the Item **Attribute** record.
   3. Iterate through all values of the **Values** parameter. For each value, invoke the GetAttributeValueID function by passing the value of the **ID** field from the **Item Attribute** record and **AttrVal** as parameters, and add the resulting value of this invocation to the **ValueIDs** collection.  
      Hint: **Values** is a collection, take advantage of the **foreach** statement.
   4. Exit the function with the value of the **ID** field of the **Item Attribute** record as the return value.
2. Create a new local function named **InitializeAttributes**.
3. Declare the following local variables for the **InitializeAttributes** function:

|  |  |
| --- | --- |
| Variable | Type |
| LabelType | Label 'Figurine Type' |
| LabelAge | Label 'Figurine Age' |
| LabelGender | Label 'Figurine Gender' |

1. Write the following logic for the **InitializeAttributes** function:
   1. Set **TypeAttributeID** global variable to the result of the invocation of **GetAttributeID**, with **LabelType**, **ListType**, and **TypeIDs** passed as parameters.
   2. Set **AgeAttributeID** global variable to the result of the invocation of **GetAttributeID**, with **LabelAge,** **ListAge**, and **AgeIDs** passed as parameters.
   3. Set **GenderAttributeID** global variable to the result of the invocation of **GetAttributeID**, with **LabelGender**, **ListGender**, and **GenederIDs** passed as parameters.
2. Create a new global function named **Initialize**.
3. Declare the following local variables for the **Initialize** function:

|  |  |
| --- | --- |
| Variable | Type |
| LabelType | Label 'Human,Animal' |
| LabelAge | Label 'Adult,Child' |
| LabelGender | Label 'Male,Female' |

1. Write the following logic for the **Initialize** function:
   1. If **Initialized** is true, exit the function
   2. Split the **LabelType** by comma (',') and assign the resulting value to the **LabelType** global variable.  
      Hint: Use the **Split** function of the Text type. Since Label type is not equal to the Text type, you cannot directly invoke **Split** on a label variable, so you have to first convert the label to text by using the **Format** function.
   3. Split the **LabelAge** by comma (',') and assign the resulting value to the **ListAge** global variable.
   4. Split the **LabelGender** by comma (',') and assign the resulting value to the **ListGender** global variable.
   5. Invoke the **InitializeAttribute** function.
   6. Set **Initialized** to true.
2. Create a new local function named **UpdateItemAttribute**. It receives the following parameters:

|  |  |
| --- | --- |
| Parameter | Type |
| Item | Record Item |
| ID | Integer |
| ValueID | Integer |

1. Declare the following local variable for the **UpdateItemAttribute** function.

|  |  |
| --- | --- |
| Variable | Type |
| ItemAttrValMap | Record “Item Attribute Value Mapping” |

1. Write the following logic for the **UpdateItemAttribute** function:
   1. On the **Item Attribute Value Mapping** record variable, set **Table ID** to the ID of the **Item** table, **No.** to the value of the **No.** field of the **Item** record parameter, and **Item Attribute ID** to the value of the **ID** parameter.
   2. Try to find the **Item Attribute Value Mapping** record with matching primary key.
   3. If there is this record, delete it.
   4. If **ValueID** parameters is equal to 0, exit the function.
   5. Set the **Item Attribute Value ID** field on the **Item Attribute Value Mapping** record to the value of the **ValueID** parameter.
   6. Insert the **Item Attribute Value Mapping** record.
2. Create a new local function named **MatchAttribute**. It returns a value of type Integer, and receives the following parameters:

|  |  |
| --- | --- |
| Parameter | Type |
| CustomVisionTag | Record “Custom Vision Tag”, temporary, by reference |
| Tags | List of [Text] |
| Values | List of [Integer] |

1. Declare the following local variables for the **MatchAttribute** function:

|  |  |
| --- | --- |
| Variable | Type |
| TagConfidences | List of [Decimal] |
| Tag | Text |
| TagCount | Integer |
| BestConfidenceIndex | Integer |
| i | Integer |
| TagConfidence | Decimal |
| BestConfidence | Decimal |

1. Write the following logic for the **MatchAttribute** function:
   1. For each **Tag** in the **Tags** collection:
      1. Reset **TagCount** and **TagConfidence** to 0.
      2. Filter the **CustomVisionTag** temporary table to only those records where the **Tag** field value matches the **Tag** parameter value.
      3. Iterate through all **CustomVisionTag** records within the filter.
      4. For each record in iteration, increase the **TagCount** by 1, and then: if **TagCount** is 1, assign the **Confidence** field value from the **CustomVisionTag** temporary table to the **TagConfidence** variable; else multiply the **TagConfidence** variable by the **Confidence** field value.
      5. After completing the iteration over the **CustomVisionTag** temporary table, if **TagCount** is higher than 0, set the **TagConfidence** value to the result of taking **TagConfidence** to the power of 1 divided by **TagCount**.
      6. Add the value of the **TagConfidence** variable to the **TagConfidences** collection.
   2. In a for loop, iterate over the **TagConfidence** collection:
      1. If the value of the **TagConfidence** collection at the current index (of the for loop) is higher than the value of the **BestConfidence** variable, set the **BestConfidence** value to the **TagConfidence** value at the current index, and set the **BestConfidenceIndex** to the current index.
   3. If **BestConfidenceIndex** is higher than 0, exit the function with the value of the **Value** collection parameter at the index of the **BestConfidenceIndex**.
2. Create a new local function named **PerformTagAnalysis**. It receives the following parameters:

|  |  |
| --- | --- |
| Parameter | Type |
| Item | Record Item |
| CustomVisionTag | Record “Custom Vision Tag”, temporary, by reference |

1. Declare the following local variables for the **PerformTagAnalysis** function:

|  |  |
| --- | --- |
| Variable | Type |
| ImageAnalysisSetup | Record “Image Analysis Setup” |
| ImageAnalysis | Codeunit “Image Analysis Management” |
| Result | Codeunit “Image Analysis Result” |
| ErrorMessage | Text |
| IsUsageLimitError | Boolean |
| EntryNo | Integer |
| i | Integer |
| Ii | Integer |

1. Write the following logic for the **PerformTagAnalysis** function:
   1. If there are no pictures in the current Item record, throw an error.
   2. Retrieve the **Image Analysis Setup** record, and make sure the **Custom Vision API URI** and **Custom Vision API Key** fields have values.
   3. Invoke the **SetUriAndKey** function of the **Image Analysis Management** codeunit and pass the **Custom Vision API URI** and **Custom Vision API Key** values as its parameters.
   4. Invoke the **Initialize** function of the **Image Analysis Management** codeunit.
   5. In a for loop, iterate over each media in the **Picture** mediaset of the **Item** record. All remaining logic is a part of this loop.
   6. Invoke the **SetMedia** function of the **Image Analysis Management** codeunit and pass the media at the current index (of the for loop).
   7. Invoke the **AnalyzeTags** function of the **Image Analysis Management** codeunit, and if it returns false, retrieve the last error message, and then throw the error message.
   8. In a nested for loop, iterate over all tags retrieved by the **AnalyzeTags** function. All remaining logic is part of this nested loop.
   9. If tag confidence of the current index (of the nested loop) is higher than or equal to the **Tag Confidence Threshold** configured in the **Image Analysis Setup**, do the following:
      1. Increase **EntryNo** by 1.
      2. Initialize a new **Custom Vision Tag** record.
      3. Set the **Entry No.** field to the value of the **EntryNo** variable.
      4. Set the **Tag** field to the current tag name.
      5. Set the **Confidence** value to the current tag confidence.
      6. Insert the **Custom Vison Tag** record.
2. Create a new global function named **SuggestItemAttributes**. It receives the following parameter:

|  |  |
| --- | --- |
| Parameter | Type |
| Item | Record Item |

1. Declare the following local variables for the **SuggestItemAttributes** function:

|  |  |
| --- | --- |
| Variable | Type |
| CustomVisionTag | Record “Custom Vision Tag”, temporary, by reference |
| AttrTypeValue | Integer |
| AttrAgeValue | Integer |
| AttrGenderValue | Integer |

1. Write the following logic for the **SuggestItemAttributes** function:
   1. Invoke the **PerformTagAnalysis** local function.
   2. Invoke the **MatchAttribute** local function, pass the **CustomVisionTag**, **ListType** (global variable), and **TypeIDs** (global variable), and assign the returned value to the **AttrTypeValue** variable.
   3. Invoke the **MatchAttribute** local function, pass the **CustomVisionTag**, **ListAge** (global variable), and **AgeIDs** (global variable), and assign the returned value to the **AttrAgeValue** variable.
   4. Invoke the **MatchAttribute** local function, pass the **CustomVisionTag**, **ListGender** (global variable), and **GenderIDs** (global variable), and assign the returned value to the **AttrGenderValue** variable.
   5. Invoke the **UpdateItemAttribute** local function, pass the **Item**, **TypeAttributeID**, and **AttrTypeValue** variables.
   6. Invoke the **UpdateItemAttribute** local function, pass the **Item**, **AgeAttributeID**, and **AttrAgeValue** variables.
   7. Invoke the **UpdateItemAttribute** local function, pass the **Item**, **GenderAttributeID**, and **AttrGenderValue** variables.

### Solution: Codeunit 50136 Custom Vision Management.al

codeunit 50136 "Custom Vision Management"

{

SingleInstance = true;

var

ListType: List of [Text];

ListAge: List of [Text];

ListGender: List of [Text];

TypeIDs: List of [Integer];

AgeIDs: List of [Integer];

GenderIDs: List of [Integer];

TypeAttributeID: Integer;

AgeAttributeID: Integer;

GenderAttributeID: Integer;

Initialized: Boolean;

procedure Initialize();

var

LabelType: Label 'Human,Animal';

LabelAge: Label 'Adult,Child';

LabelGender: Label 'Male,Female';

begin

if Initialized then

exit;

ListType := Format(LabelType).Split(',');

ListAge := Format(LabelAge).Split(',');

ListGender := Format(LabelGender).Split(',');

InitializeAtributes();

Initialized := true;

end;

procedure SuggestItemAttributes(Item: Record Item);

var

CustomVisionTag: Record "Custom Vision Tag" temporary;

AttrTypeValue: Integer;

AttrAgeValue: Integer;

AttrGenderValue: Integer;

begin

PerformTagAnalysis(Item, CustomVisionTag);

AttrTypeValue := MatchAttribute(CustomVisionTag, ListType, TypeIDs);

AttrAgeValue := MatchAttribute(CustomVisionTag, ListAge, AgeIDs);

AttrGenderValue := MatchAttribute(CustomVisionTag, ListGender, GenderIDs);

UpdateItemAttribute(Item, TypeAttributeID, AttrTypeValue);

UpdateItemAttribute(Item, AgeAttributeID, AttrAgeValue);

UpdateItemAttribute(Item, GenderAttributeID, AttrGenderValue);

end;

local procedure GetAttributeValueID(AttrID: Integer; AttrVal: Text): Integer;

var

ItemAttrVal: Record "Item Attribute Value";

begin

with ItemAttrVal do begin

SetRange("Attribute ID", AttrID);

SetRange(Value, AttrVal);

if not FindFirst() then begin

Reset();

SetCurrentKey(ID);

if FindLast() then;

Init();

"Attribute ID" := AttrID;

ID += 1;

Value := AttrVal;

Insert();

end;

exit(ID);

end;

end;

local procedure GetAttributeID(Attr: Text; Values: List of [Text]; ValueIDs: List of [Integer]): Integer;

var

ItemAttr: Record "Item Attribute";

AttrVal: Text;

begin

with ItemAttr do begin

SetRange(Name, Attr);

if not FindFirst() then begin

Reset();

if FindLast() then;

ID += 1;

Init();

Name := Attr;

Type := ItemAttr.Type::Option;

Insert();

end;

foreach AttrVal in Values do

ValueIDs.Add(GetAttributeValueID(ID, AttrVal));

exit(ID);

end;

end;

local procedure InitializeAtributes();

var

LabelType: Label 'Figurine Type';

LabelAge: Label 'Figurine Age';

LabelGender: Label 'Figurine Gender';

begin

TypeAttributeID := GetAttributeID(LabelType, ListType, TypeIDs);

AgeAttributeID := GetAttributeID(LabelAge, ListAge, AgeIDs);

GenderAttributeID := GetAttributeID(LabelGender, ListGender, GenderIDs);

end;

local procedure PerformTagAnalysis(Item: Record Item; var CustomVisionTag: Record "Custom Vision Tag" temporary);

var

ImageAnalysisSetup: Record "Image Analysis Setup";

ImageAnalysis: Codeunit "Image Analysis Management";

Result: Codeunit "Image Analysis Result";

ErrorMessage: Text;

IsUsageLimitError: Boolean;

EntryNo: Integer;

i: Integer;

ii: Integer;

begin

if Item.Picture.Count = 0 then

Error('No media available for this item.');

ImageAnalysisSetup.Get();

ImageAnalysisSetup.TestField("Custom Vision API URI");

ImageAnalysisSetup.TestField("Custom Vision API Key");

ImageAnalysis.SetUriAndKey(

ImageAnalysisSetup."Custom Vision API URI",

ImageAnalysisSetup."Custom Vision API Key");

ImageAnalysis.Initialize();

for i := 1 to Item.Picture.Count do begin

ImageAnalysis.SetMedia(Item.Picture.Item(i));

if not ImageAnalysis.AnalyzeTags(Result) then begin

ImageAnalysis.GetLastError(

ErrorMessage,

IsUsageLimitError);

Error('Invocation error: %1', ErrorMessage);

end;

for ii := 1 to Result.TagCount do begin

if Result.TagConfidence(ii) >= ImageAnalysisSetup."Tag Confidence Threshold" then begin

EntryNo += 1;

CustomVisionTag.Init();

CustomVisionTag."Entry No." := EntryNo;

CustomVisionTag.Tag := Result.TagName(ii);

CustomVisionTag.Confidence := Result.TagConfidence(ii);

CustomVisionTag.Insert();

end;

end;

end;

end;

local procedure MatchAttribute(var CustomVisionTag: Record "Custom Vision Tag" temporary; Tags: List of [Text]; Values: List of [Integer]): Integer;

var

TagConfidences: List of [Decimal];

Tag: Text;

TagCount: Integer;

BestConfidenceIndex: Integer;

i: Integer;

TagConfidence: Decimal;

BestConfidence: Decimal;

begin

foreach Tag in Tags do begin

TagCount := 0;

TagConfidence := 0;

CustomVisionTag.SetRange(Tag, Tag);

if CustomVisionTag.FindSet() then

repeat

TagCount += 1;

if TagCount = 1 then

TagConfidence := CustomVisionTag.Confidence

else

TagConfidence \*= CustomVisionTag.Confidence;

until CustomVisionTag.Next() = 0;

if TagCount > 0 then

TagConfidence := Power(TagConfidence, 1 / TagCount);

TagConfidences.Add(TagConfidence);

end;

for i := 1 to TagConfidences.Count do begin

TagConfidence := TagConfidences.Get(i);

if TagConfidence > BestConfidence then begin

BestConfidence := TagConfidence;

BestConfidenceIndex := i;

end;

end;

if BestConfidenceIndex > 0 then

exit(Values.Get(BestConfidenceIndex))

end;

local procedure UpdateItemAttribute(Item: Record Item; ID: Integer; ValueID: Integer);

var

ItemAttrValMap: Record "Item Attribute Value Mapping";

begin

with ItemAttrValMap do begin

"Table ID" := Database::Item;

"No." := Item."No.";

"Item Attribute ID" := ID;

if Find() then

Delete();

if ValueID = 0 then

exit;

"Item Attribute Value ID" := ValueID;

Insert();

end;

end;

}

## Customize Item Card page

1. Create a new file and name it PageExtension 50137 Item Card Extension.al.
2. In this file, declare pageextension 50137 "Item Card Extension" that extends the **Item Card** page.
3. Declare the actions section.
4. In the actions section, add an action as the last action in the action group **Item**. This action invokes the **SuggestItemAttribute** function of the **Custom Vision Management** codeunit, and then updates the current page. Promote this action to the Category4 category.
5. Create the **OnOpenPage** trigger, and from its body invoke the **Initialize** function of the **Custom Vision Management** codeunit.

### Solution: PageExtension 50137 Item Card Extension.al

pageextension 50137 "Item Card Extension" extends "Item Card"

{

actions

{

addlast(Item)

{

action(SuggestAttributes)

{

Caption = 'Suggest Attributes';

Image = SuggestTables;

ApplicationArea = All;

trigger OnAction();

begin

CustomVisionManagement.SuggestItemAttributes(Rec);

CurrPage.Update(false);

end;

}

}

}

var

CustomVisionManagement: Codeunit "Custom Vision Management";

trigger OnOpenPage();

begin

CustomVisionManagement.Initialize();

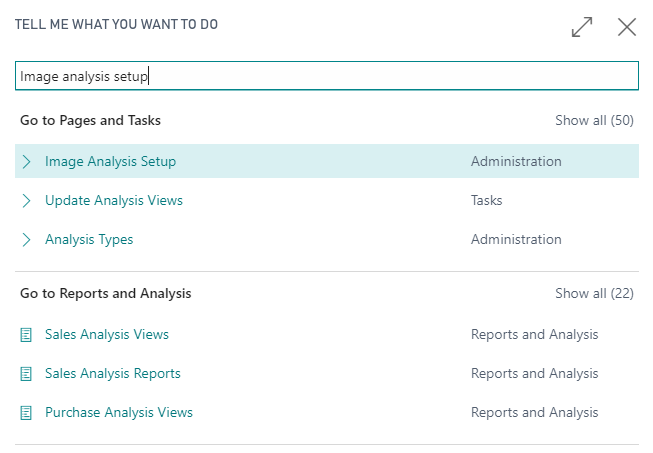
end;

}

# Test Custom Vision integration

In this exercise you test the Custom Vision integration you have just developed.

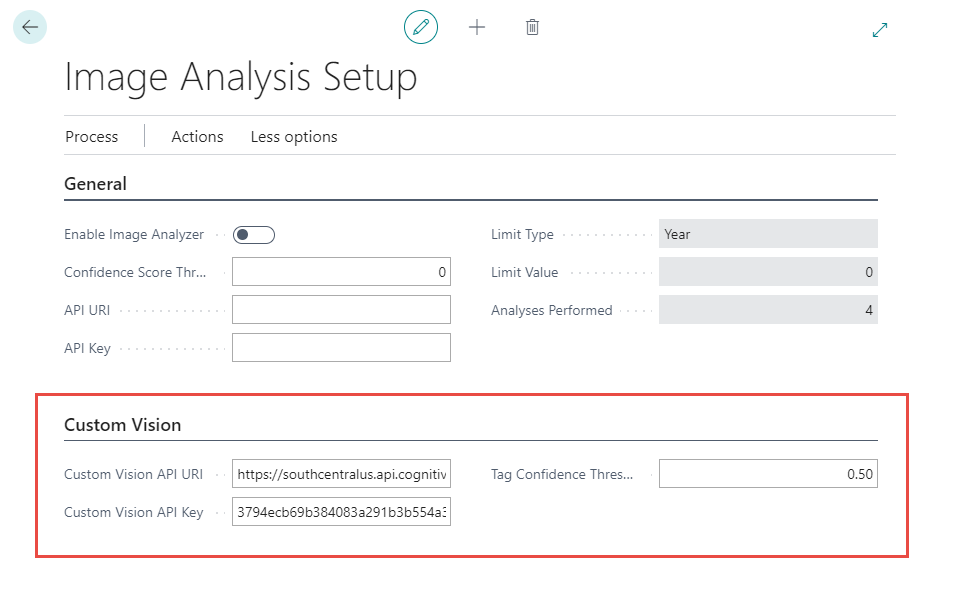
1. Build, deploy, and run the **Custom Vision Demo** extension.
2. In the **Tell me** box, search for and then run the **Image Analysis Setup** page:



1. In the **Image Analysis Setup** page, enter the following values:
   1. **Custom Vision API URI**: it’s a combination of the **Prediction Endpoint** and **Project Id** values that you copied from your **Red Carpet Custom Vision** project earlier during this hands-on lab. It is formed like this: <Prediction\_Endpoint>/<Project\_Id>/image.  
      For example:

https://southcentralus.api.cognitive.microsoft.com/customvision/v2.0/Prediction/9c3280c7-3002-4c3f-ab00-c478fa5ed474/image

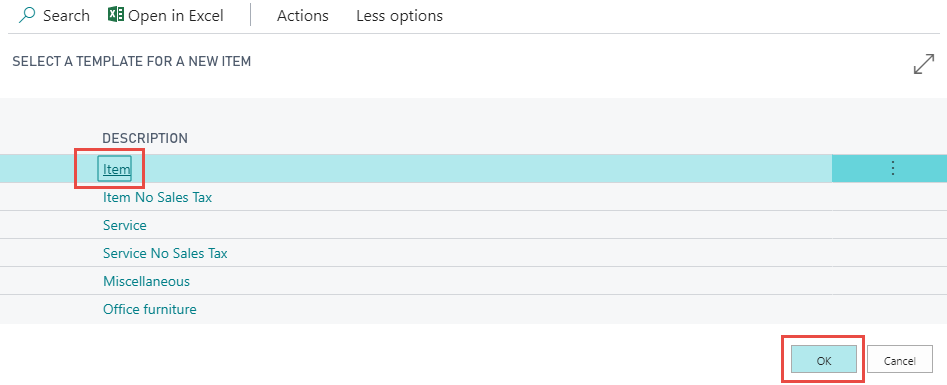
* 1. **Custom Vision API Key**: it’s the **Prediction Key** that you copied from your Red Carpet Custom Vision project.
  2. **Tag Confidence Threshold**: enter the value of 0.5.



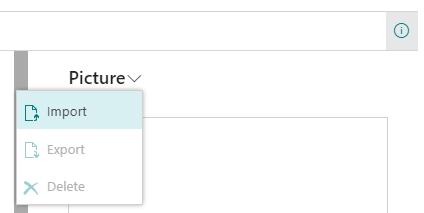
1. Close the **Image Analysis Setup** page.
2. If you configured your workspace in Visual Studio Code correctly, you should now be in the **Items** list place. If you are not, use the **Tell me** box to search for and run the **Items** list.
3. Click **New**.



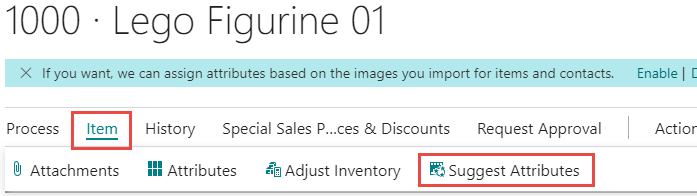
1. In the **Select a template for a new Item**, make sure that the **Item** row is selected, and then click **OK**.



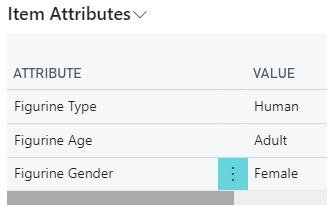
1. In the **Description** field, enter “Lego Figurine 01”.
2. In the **Picture** factbox, click **Picture > Import**.



1. In the **Select a picture to upload** dialog, click **Choose**.
2. Browse to the **Validate** folder that you downloaded earlier in this hands-on lab, select the **Validation (32).jpg** image, and then click **Open**.
3. Click **Item > Suggest attributes**.



After a few seconds, the **Item Attributes** factbox updates and displays the following values:



1. Repeat steps 6 thru 12 for:
   1. Validation (23).jpg
   2. Validation (26).jpg
   3. Validation (101).jpg