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.

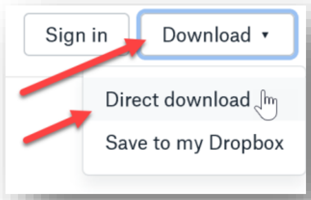
Contents

Create a custom vision model

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**:



3. Save the incoming .zip file on your computer.
4. 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

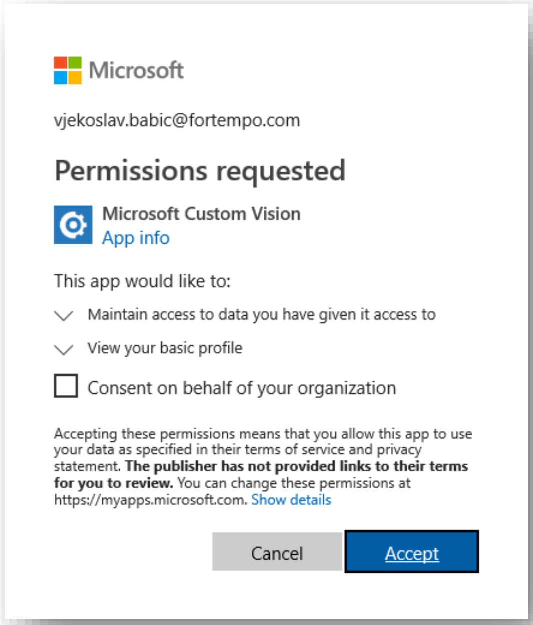
Inside the Train folder, there are 426 images are organized according to the following rules:

- 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:


Microsoft

vjekoslav.babic@fortempo.com

Permissions requested

Microsoft Custom Vision
App info

This app would like to:

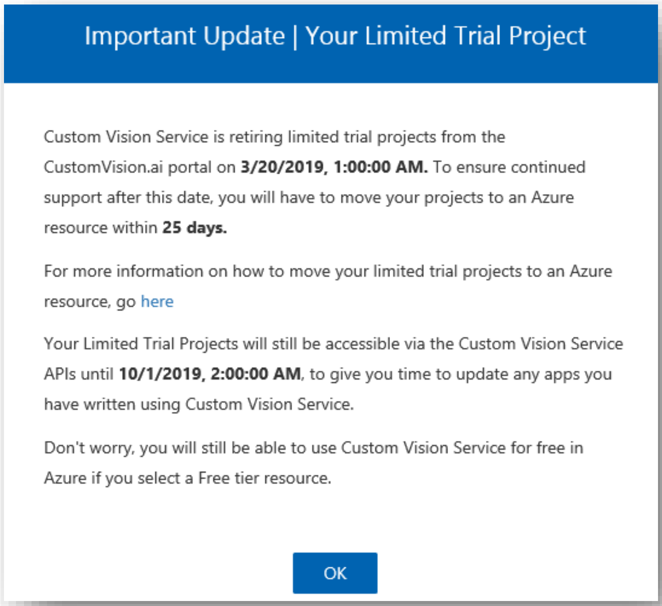
Maintain access to data you have given it access to

View your basic profile

☐ Consent on behalf of your organization

Accepting these permissions means that you allow this app to use your data as specified in their terms of service and privacy statement. **The publisher has not provided links to their terms for you to review.** You can change these permissions at https://myapps.microsoft.com. Show details

5. **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).
6. Click **Accept**.
7. If you are following this script before March 20, 2019, then the **Important Update** dialog will show:


Important Update | Your Limited Trial Project

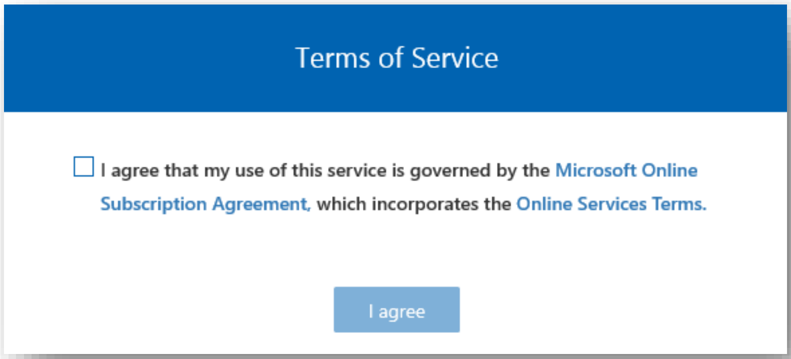
Custom Vision Service is retiring limited trial projects from the CustomVision.ai portal on **3/20/2019, 1:00:00 AM.** To ensure continued support after this date, you will have to move your projects to an Azure resource within **25 days.**

For more information on how to move your limited trial projects to an Azure resource, go here

Your Limited Trial Projects will still be accessible via the Custom Vision Service APIs until **10/1/2019, 2:00:00 AM**, to give you time to update any apps you have written using Custom Vision Service.

Don't worry, you will still be able to use Custom Vision Service for free in Azure if you select a Free tier resource.

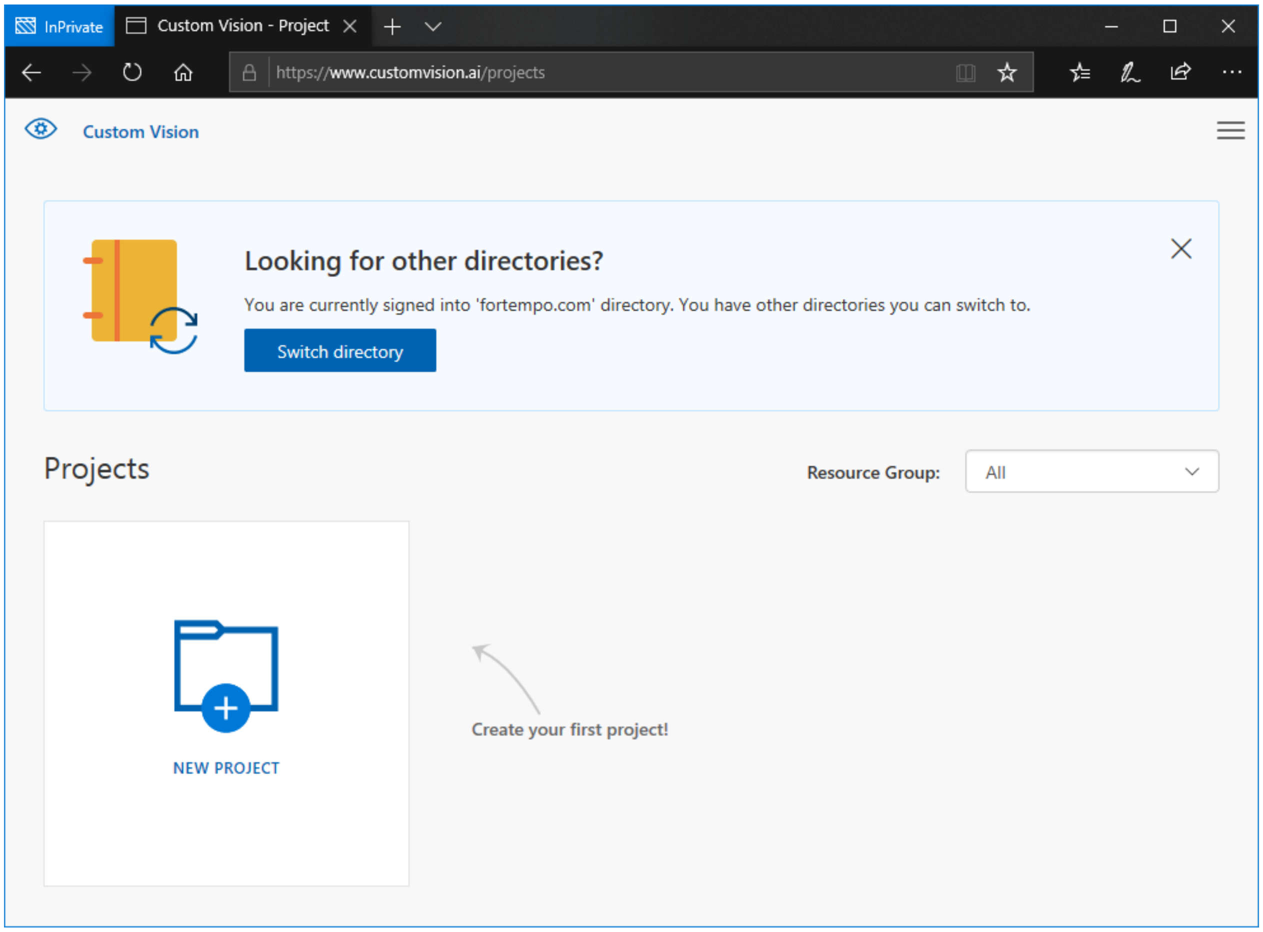
8. Click **OK**. The **Terms of Service** dialog shows:



Terms of Service

☐ I agree that my use of this service is governed by the Microsoft Online Subscription Agreement, which incorporates the Online Services Terms.

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



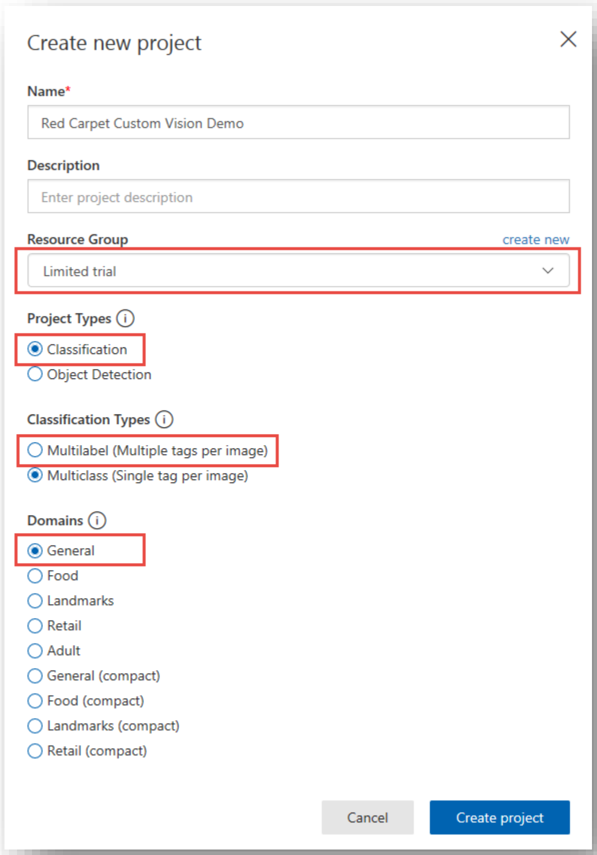
Looking for other directories?

You are currently signed into 'fortempo.com' directory. You have other directories you can switch to.

Create a Custom Vision project

1. Click **New Project**. The **Create new project** dialog opens.

2. Configure the Create new project dialog as follows:
    a. In the **Name** field, enter “Red Carpet Custom Vision Demo”.
    b. Leave the **Description** field empty.
    c. Make sure that **Resource Group** is set to **Limited Trial**.
    d. Make sure that **Project Type** is set to **Classification**.
    e. Set **Classification Types** to **Multilabel (Multiple tags per image)**.
    f. Make sure that **Domains** is set to **General**.


Create new project

Name*
Red Carpet Custom Vision Demo

Description
Enter project description

Resource Group — create new
Limited trial

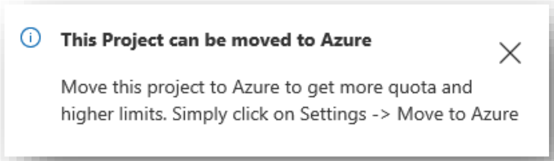
Project Types
[✓ Classification]
Object Detection

Classification Types
Multilabel (Multiple tags per image)
[✓ Multiclass (Single tag per image)]

Domains
[✓ General]
Food
Landmarks
Retail
Adult
General (compact)
Food (compact)
Landmarks (compact)
Retail (compact)

Cancel
Create project

3. Click **Create project**.
4. If you see the following notification, close it.


This Project can be moved to Azure

Move this project to Azure to get more quota and higher limits. Simply click on Settings -> Move to Azure

5. Under the **Looks like you don’t have any images here** caption, click **Add images**.
6. Browse to the **Train** folder.

7. Select five arbitrary pictures of **A Human child male – Lying** and click **Open**. The **Image upload** dialog shows:

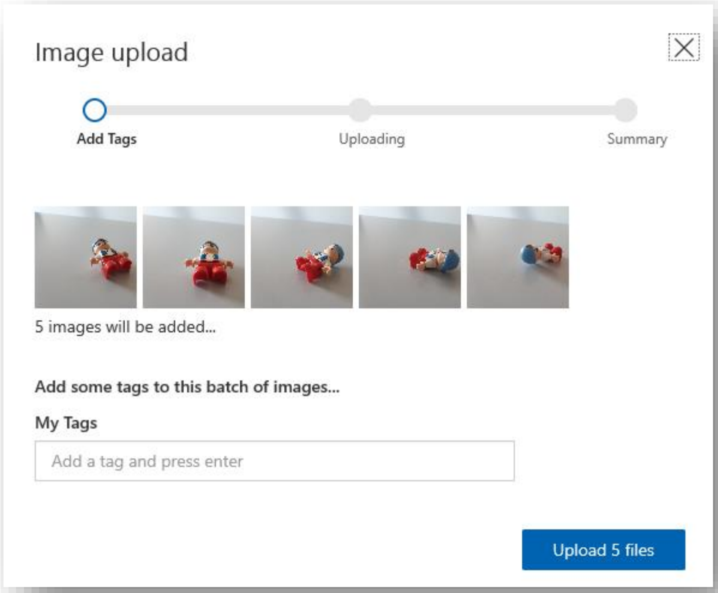
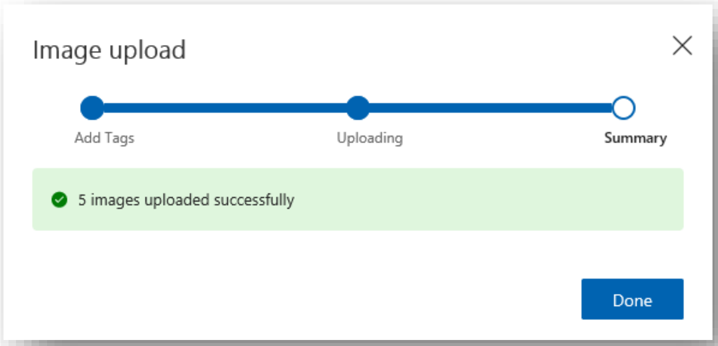
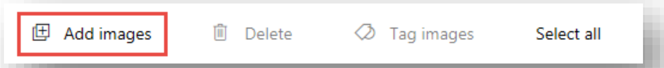


Image upload
Add Tags
Uploading
Summary
5 images will be added...
Add some tags to this batch of images...
My Tags
Add a tag and press enter
Upload 5 files

8. 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.
9. Click **Upload 5 files**. The Image upload dialog will update:

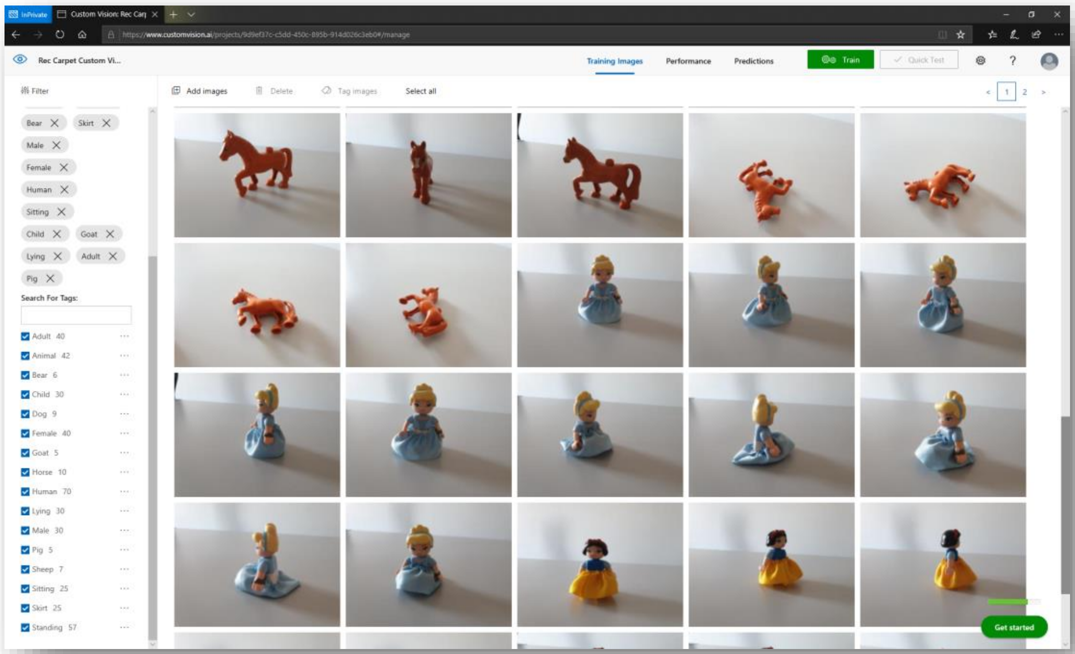


10. Click **Done**.
11. In the upper action bar, click **Add images**:



12. 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:
    a. Human
    b. Adult
    c. Child
    d. Male
    e. Female
    f. Standing
    g. Sitting
    h. Lying
    i. Animal
    j. Horse
    k. Goat
    l. Dog
    m. Pig
    n. Sheep
    o. Bear

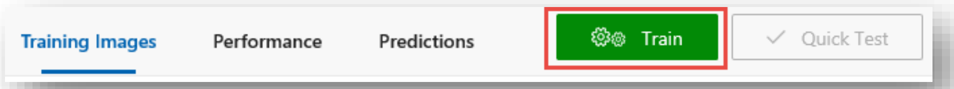
    With identities D and I, use an additional tag: Skirt.
13. When you have completed step 12, you will have a workspace with more setup similar to this:



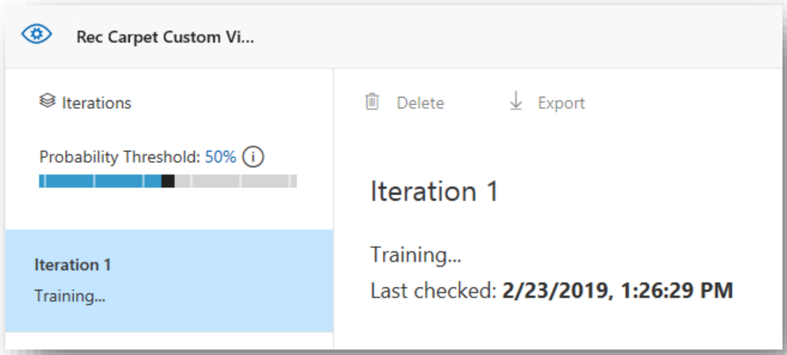
Adult	40
Animal	42
Bear	6
Child	30
Dog	9
Female	40
Goat	5
Horse	10
Human	70
Lying	30
Male	30
Pig	5
Sheep	7
Sitting	25
Skirt	25
Standing	57

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.

14. Click **Train**.

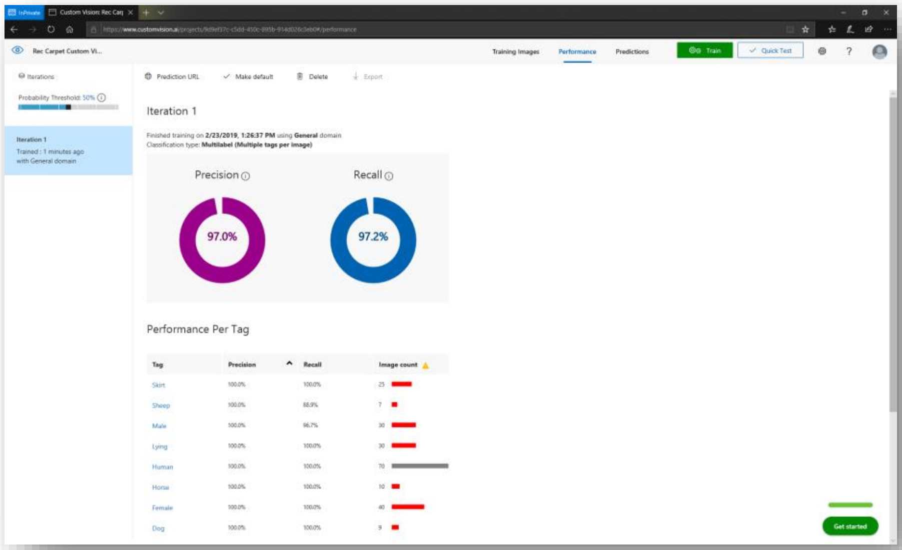


The model starts training and shows the following info:



Training...
Last checked: **2/23/2019, 1:26:29 PM**

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



Finished training on **2/23/2019, 1:26:37 PM** using **General** domain
Classification type: **Multilabel (Multiple tags per image)**

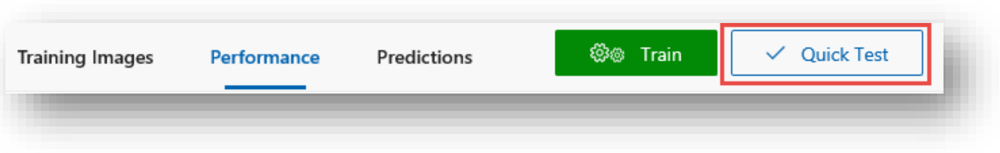
Performance Per Tag

Tag	Precision	Recall	Image count
Skirt	100.0%	100.0%	25
Sheep	100.0%	88.9%	7
Male	100.0%	96.7%	30
Lying	100.0%	100.0%	30
Human	100.0%	100.0%	70
Horse	100.0%	100.0%	10
Female	100.0%	100.0%	40
Dog	100.0%	100.0%	9

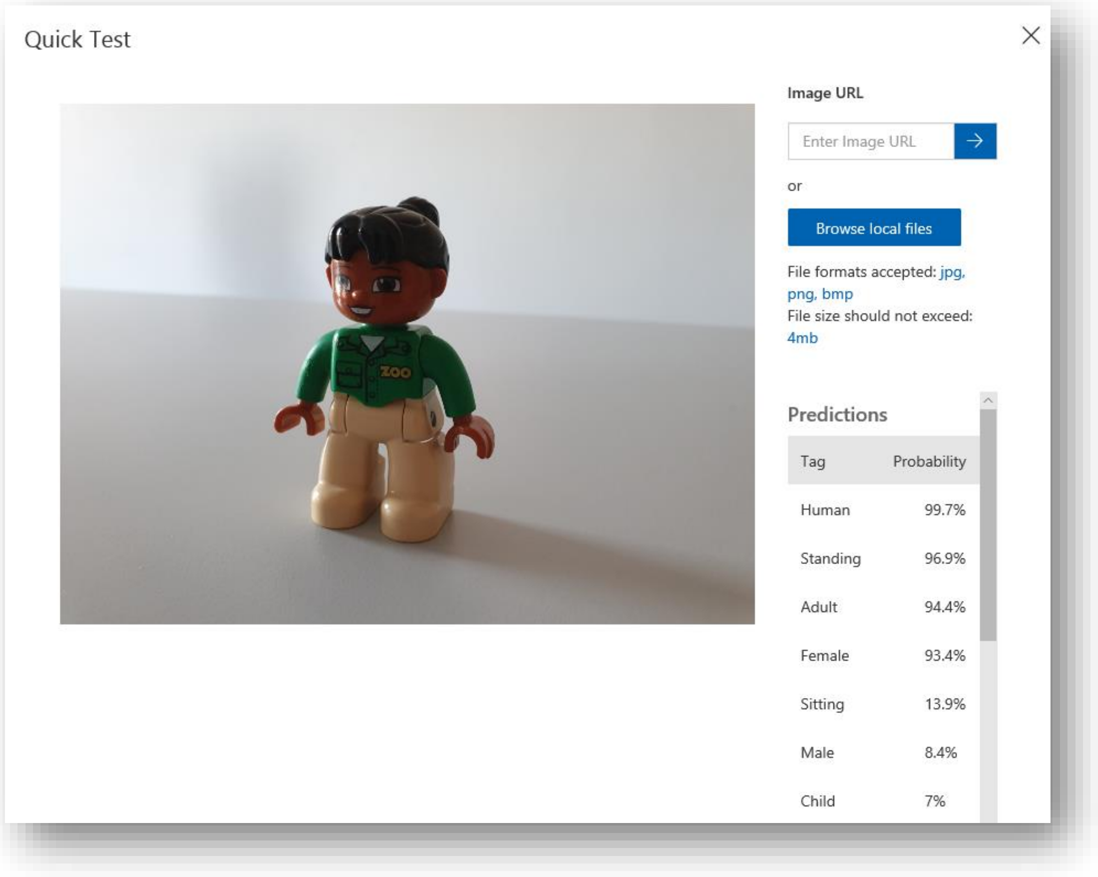
Pay attention to the the Performance Per Tag section, it shows you all your tags with their statistics, and recommendations. Tags that are marked with red in the Image count column will perform poorly as compared to tags that are gray. Also, tags with longer bars will be correctly recognized more often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

Validating the model

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



2. In the **Quick Test** dialog, click **Browse local files**.
3. 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:



File formats accepted: jpg, png, bmp
File size should not exceed: 4mb

Predictions

Tag	Probability
Human	99.7%
Standing	96.9%
Adult	94.4%
Female	93.4%
Sitting	13.9%
Male	8.4%
Child	7%

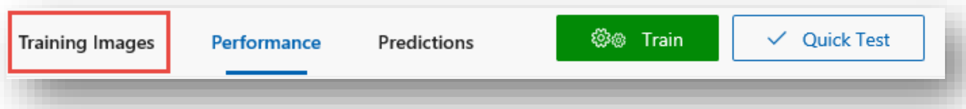
4. Click Browse local files again, and repeat step 18 for the following images:
   a. Validation (11).jpg
   b. Validation (19).jpg
   c. Validation (23).jpg
   d. Validation (39).jpg
   e. Validation (80).jpg
   f. Validation (142).jpg

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.

5. 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.

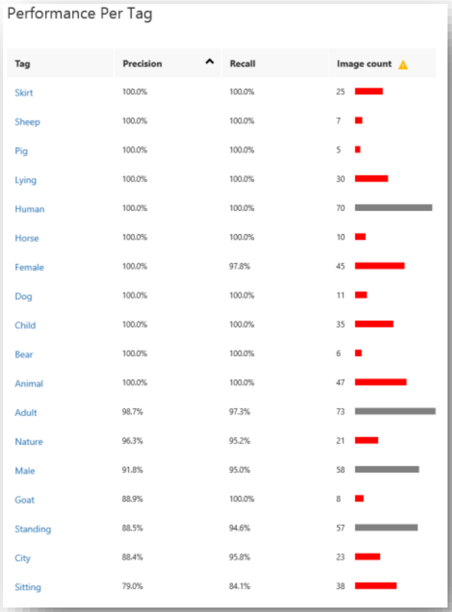
Improving the model

6. Click **Training images**.



7. Upload and tag additional images of any characters with either nature or city setting. Tag those images only with the following tags:
   a. **City** or **Nature**.
   b. For humans specify **Adult** or **Child**, and **Male** or **Female**, but do not specify **Human**.
   c. For any characters specify pose if it is either **Sitting** or **Lying** (do not tag additional images with **Standing**).
   d. 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.
8. 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.



Performance Per Tag

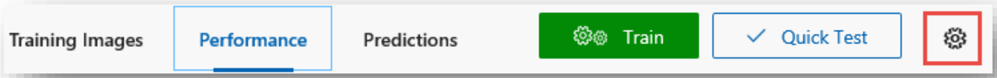
Tag	Precision	Recall	Image count
Skirt	100.0%	100.0%	25
Sheep	100.0%	100.0%	7
Pig	100.0%	100.0%	5
Lying	100.0%	100.0%	30
Human	100.0%	100.0%	70
Horse	100.0%	100.0%	10
Female	100.0%	97.8%	45
Dog	100.0%	100.0%	11
Child	100.0%	100.0%	35
Bear	100.0%	100.0%	6
Animal	100.0%	100.0%	47
Adult	98.7%	97.3%	73
Nature	96.3%	95.2%	21
Male	91.8%	95.0%	58
Goat	88.9%	100.0%	8
Standing	88.5%	94.6%	57
City	88.4%	95.8%	23
Sitting	79.0%	84.1%	38

9. Repeat steps 17 and 18 for the following images:
   a. Validation (45).jpg
   b. Validation (47).jpg
   c. Validation (48).jpg
   d. Validation (52).jpg
   e. Validation (59).jpg
   f. Validation (70).jpg
   g. Validation (87).jpg

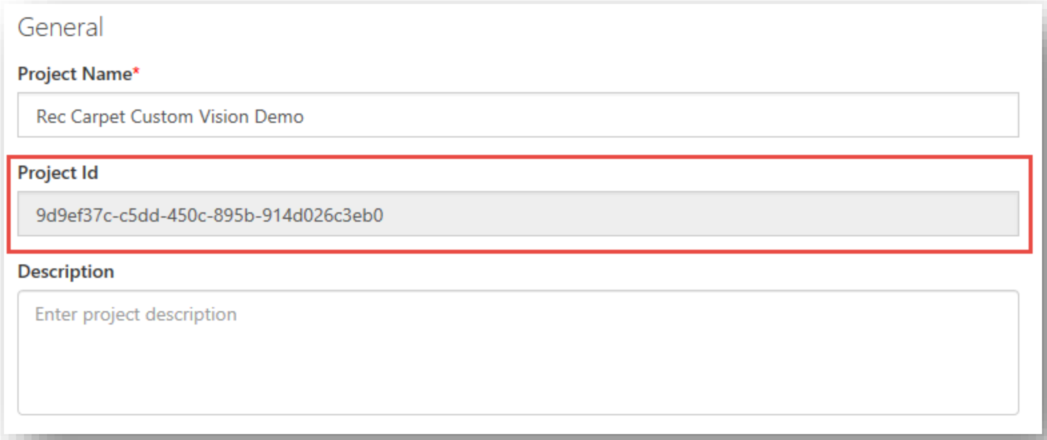
   You can notice that city and nature are recognized, but not always accurately, and that recognition of other tags has probably improved. To further improve the model, you would have to upload much larger number of images and try to keep them as balanced as possible.

Preparing for web services use

1. Click the settings icon.



2. In the **General** section, take a note of the **Project Id**.



General

Project Name*

Rec Carpet Custom Vision Demo

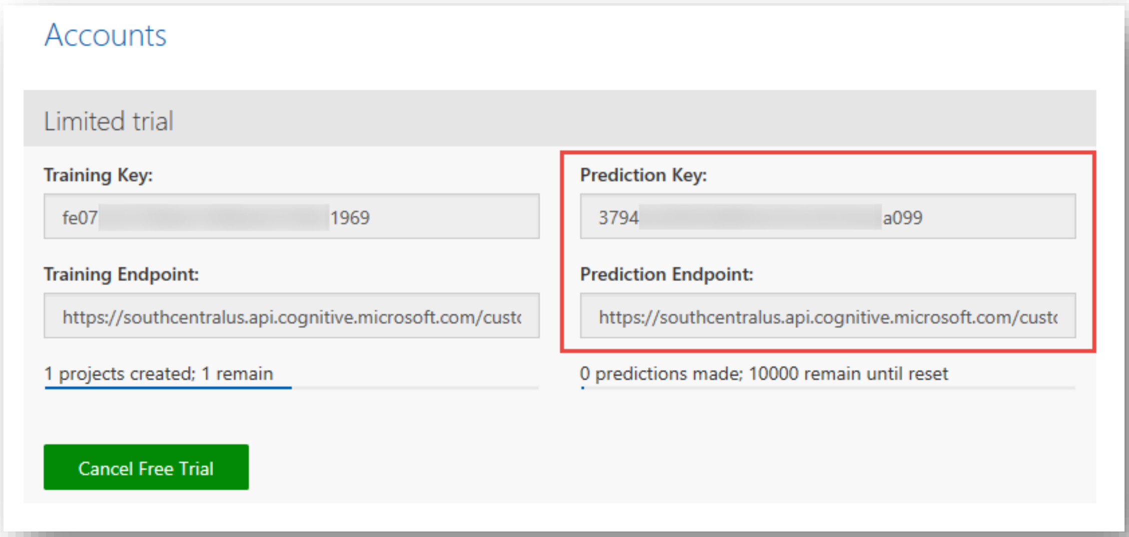
Project Id

9d9ef37c-c5dd-450c-895b-914d026c3eb0

Description

Enter project description

3. In the **Accounts** section, under **Limited trial**, take a note of the **Prediction Key** and **Prediction Endpoint**.



Accounts

Limited trial

Training Key:

fe07 1969

Training Endpoint:

https://southcentralus.api.cognitive.microsoft.com/custo

1 projects created; 1 remain

Prediction Key:

3794 a099

Prediction Endpoint:

https://southcentralus.api.cognitive.microsoft.com/custo

0 predictions made; 10000 remain until reset

4. Copy the **Project Id**, **Prediction Key** and **Prediction Endpoint** values. 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

4. Create a new file and name it PageExtension 50136 Image Analysis Setup Ext.al.
5. In this file, declare pageextension 50136 "Image Analysis Setup Ext." that extends the **Image Analysis Setup** page.
6. Declare the layout section, and inside it add a new group named **Custom Vision** after the **General** group.
7. 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

4. 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

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

Parameter	Type
AttrID	Integer
AttrVal	Text

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

Variable	Type
ItemAttrVal	Record “Item Attribute Value”

6. Write the following logic for the **GetAttributeValueID** function:
    a. 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.
    b. If this record cannot be found:
        i. Reset any filters on **Item Attribute Value**.
        ii. Try to find the last record, with ordering by the ID field, ascending.
        iii. Initialize a new **Item Attribute Value** record.
        iv. Set **Attribute ID** to the value of the **AttrID** parameter.
        v. Increase **ID** by 1.
        vi. Set **Value** to the value of the **AttrVal** parameter.

        vii. Insert the **Item Attribute Value** record.
    c. Exit the function with the value of the **ID** field from the **Item Attribute Value** as the return value.

7. 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]

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

Variable	Type
ItemAttr	Record “Item Attribute”
AttrValue	Text

9. Write the following logic for the **GetAttributeID** function:
    a. Try to find the first **Item Attribute** record where the **Name** field value matches the **Attr** parameter.
    b. If this record cannot be found:
        i. Reset any filters on the **Item Attribute** table.
        ii. Try to find the last record.
        iii. Initialize a new **Item Attribute** record.
        iv. Increase the **ID** by 1.
        v. Set **Name** to the value of the **Attr** parameter.
        vi. Set **Type** to Option.
        vii. Insert the Item **Attribute** record.
    c. 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.
    d. Exit the function with the value of the **ID** field of the **Item Attribute** record as the return value.
10. Create a new local function named **InitializeAttributes**.
11. Declare the following local variables for the **InitializeAttributes** function:

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

12. Write the following logic for the **InitializeAttributes** function:

    a. Set **TypeAttributeID** global variable to the result of the invocation of **GetAttributeID**, with **LabelType**, **ListType**, and **TypeIDs** passed as parameters.
    b. Set **AgeAttributeID** global variable to the result of the invocation of **GetAttributeID**, with **LabelAge, ListAge**, and **AgeIDs** passed as parameters.
    c. Set **GenderAttributeID** global variable to the result of the invocation of **GetAttributeID**, with **LabelGender**, **ListGender**, and **GenederIDs** passed as parameters.

13. Create a new global function named **Initialize**.
14. Declare the following local variables for the **Initialize** function:

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

15. Write the following logic for the **Initialize** function:
    a. If **Initialized** is true, exit the function
    b. 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.
    c. Split the **LabelAge** by comma (',') and assign the resulting value to the **ListAge** global variable.
    d. Split the **LabelGender** by comma (',') and assign the resulting value to the **ListGender** global variable.
    e. Invoke the **InitializeAttribute** function.
    f. Set **Initialized** to true.
16. Create a new local function named **UpdateItemAttribute**. It receives the following parameters:

Parameter	Type
Item	Record Item
ID	Integer
ValueID	Integer

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

Variable	Type
ItemAttrValMap	Record “Item Attribute Value Mapping”

18. Write the following logic for the **UpdateItemAttribute** function:
    a. 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.
    b. Try to find the **Item Attribute Value Mapping** record with matching primary key.

c. If there is this record, delete it.
d. If **ValueID** parameters is equal to 0, exit the function.
e. Set the **Item Attribute Value ID** field on the **Item Attribute Value Mapping** record to the value of the **ValueID** parameter.
f. Insert the **Item Attribute Value Mapping** record.

19. 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]

20. 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

21. Write the following logic for the **MatchAttribute** function:
    a. For each **Tag** in the **Tags** collection:
        i. Reset **TagCount** and **TagConfidence** to 0.
        ii. Filter the **CustomVisionTag** temporary table to only those records where the **Tag** field value matches the **Tag** parameter value.
        iii. Iterate through all **CustomVisionTag** records within the filter.
        iv. 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.
        v. 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**.
        vi. Add the value of the **TagConfidence** variable to the **TagConfidences** collection.
    b. In a for loop, iterate over the **TagConfidence** collection:
        i. 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.
    c. If **BestConfidenceIndex** is higher than 0, exit the function with the value of the **Value** collection parameter at the index of the **BestConfidenceIndex**.

22. 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

23. 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

24. Write the following logic for the **PerformTagAnalysis** function:
    a. If there are no pictures in the current Item record, throw an error.
    b. Retrieve the **Image Analysis Setup** record, and make sure the **Custom Vision API URI** and **Custom Vision API Key** fields have values.
    c. 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.
    d. Invoke the **Initialize** function of the **Image Analysis Management** codeunit.
    e. 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.
    f. Invoke the **SetMedia** function of the **Image Analysis Management** codeunit and pass the media at the current index (of the for loop).
    g. 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.
    h. In a nested for loop, iterate over all tags retrieved by the **AnalyzeTags** function. All remaining logic is part of this nested loop.
    i. 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:
        i. Increase **EntryNo** by 1.
        ii. Initialize a new **Custom Vision Tag** record.
        iii. Set the **Entry No.** field to the value of the **EntryNo** variable.
        iv. Set the **Tag** field to the current tag name.
        v. Set the **Confidence** value to the current tag confidence.
        vi. Insert the **Custom Vison Tag** record.
25. Create a new global function named **SuggestItemAttributes**. It receives the following parameter:

Parameter	Type
Item	Record Item

26. 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

27. Write the following logic for the **SuggestItemAttributes** function:
    a. Invoke the **PerformTagAnalysis** local function.
    b. Invoke the **MatchAttribute** local function, pass the **CustomVisionTag**, **ListType** (global variable), and **TypeIDs** (global variable), and assign the returned value to the **AttrTypeValue** variable.
    c. Invoke the **MatchAttribute** local function, pass the **CustomVisionTag**, **ListAge** (global variable), and **AgeIDs** (global variable), and assign the returned value to the **AttrAgeValue** variable.
    d. Invoke the **MatchAttribute** local function, pass the **CustomVisionTag**, **ListGender** (global variable), and **GenderIDs** (global variable), and assign the returned value to the **AttrGenderValue** variable.
    e. Invoke the **UpdateItemAttribute** local function, pass the **Item**, **TypeAttributeID**, and **AttrTypeValue** variables.
    f. Invoke the **UpdateItemAttribute** local function, pass the **Item**, **AgeAttributeID**, and **AttrAgeValue** variables.
    g. 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.

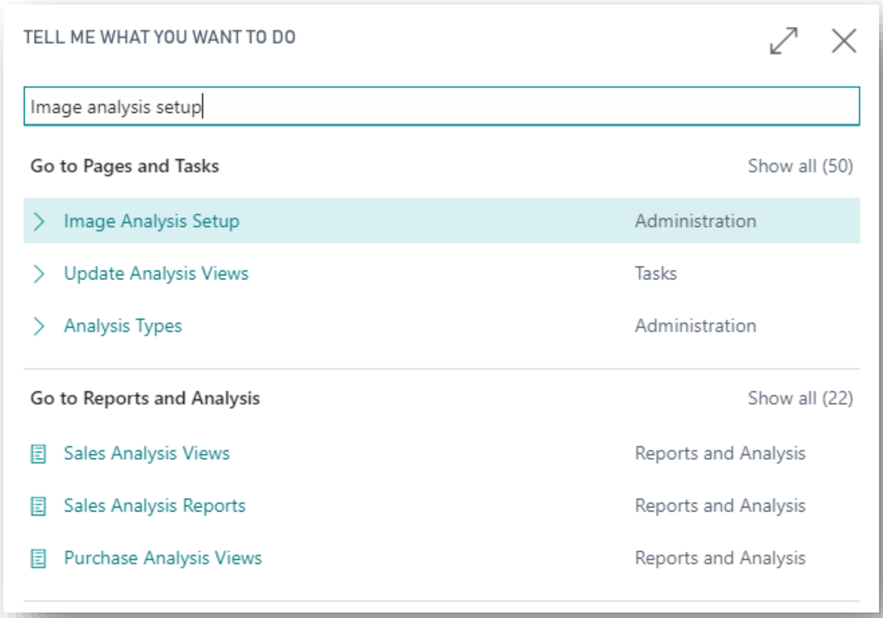
Solution: PageExtension 50137 Item Card Extension.al

```
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:



TELL ME WHAT YOU WANT TO DO

Image analysis setup

Go to Pages and Tasks	Show all (50)
Image Analysis Setup	Administration
Update Analysis Views	Tasks
Analysis Types	Administration

Go to Reports and Analysis	Show all (22)
Sales Analysis Views	Reports and Analysis
Sales Analysis Reports	Reports and Analysis
Purchase Analysis Views	Reports and Analysis

3. In the **Image Analysis Setup** page, enter the following values:
    a. **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
    b. **Custom Vision API Key**: it's the **Prediction Key** that you copied from your Red Carpet Custom Vision project.
    c. **Tag Confidence Threshold**: enter the value of 0.5.

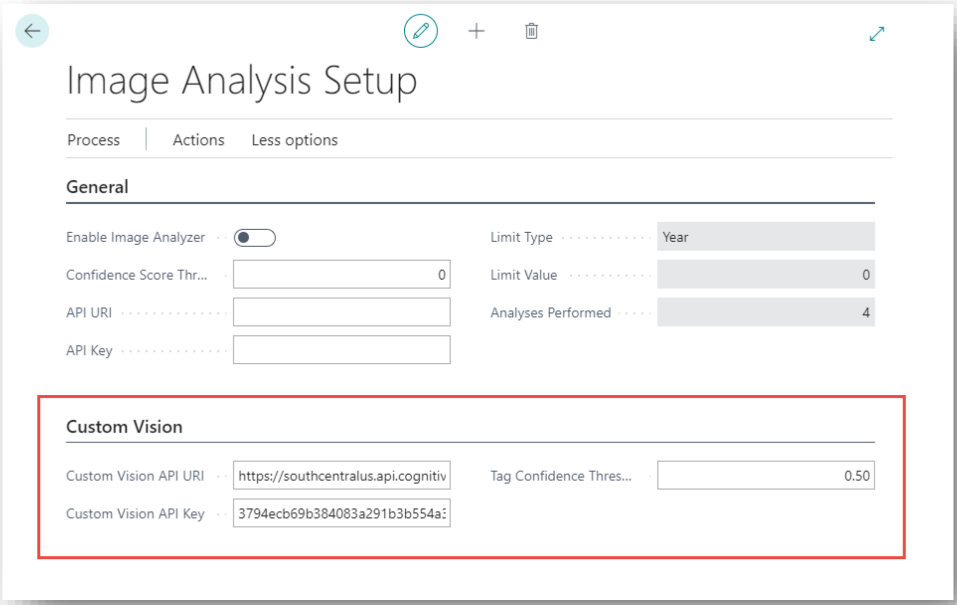


Image Analysis Setup

Process | Actions | Less options

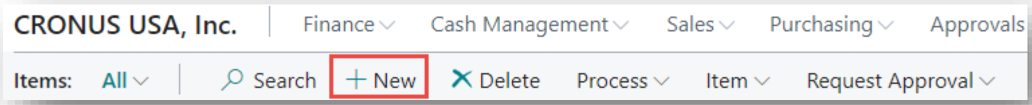
General

Enable Image Analyzer		Limit Type	Year
Confidence Score Thr...	0	Limit Value	0
API URI		Analyses Performed	4
API Key			

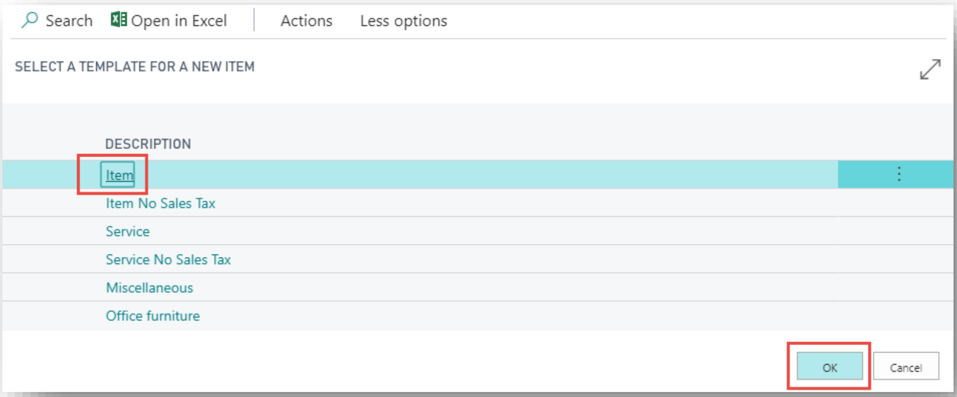
Custom Vision

Custom Vision API URI	https://southcentralus.api.cognitiv	Tag Confidence Thres...	0.50
Custom Vision API Key	3794ecb69b384083a291b3b554a:		

4. Close the **Image Analysis Setup** page.
5. 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.
6. Click **New**.



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



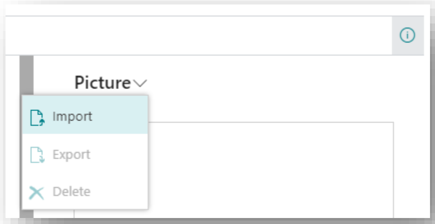
Search | Open in Excel | Actions | Less options

SELECT A TEMPLATE FOR A NEW ITEM

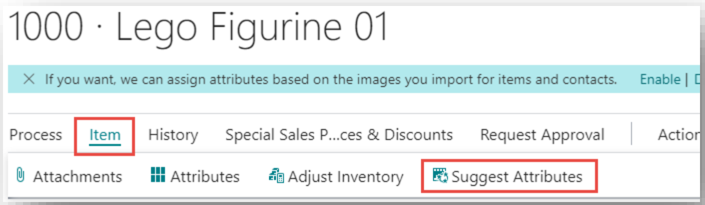
DESCRIPTION
Item
Item No Sales Tax
Service
Service No Sales Tax
Miscellaneous
Office furniture

OK | Cancel

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

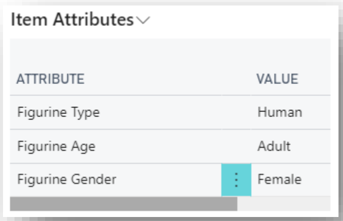


10. In the **Select a picture to upload** dialog, click **Choose**.
11. Browse to the **Validate** folder that you downloaded earlier in this hands-on lab, select the **Validation (32).jpg** image, and then click **Open**.
12. Click **Item > Suggest attributes**.


1000 · Lego Figurine 01

If you want, we can assign attributes based on the images you import for items and contacts. Enable |

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


Item Attributes

ATTRIBUTE	VALUE
Figurine Type	Human
Figurine Age	Adult
Figurine Gender	Female

13. Repeat steps 6 thru 12 for:
    a. Validation (23).jpg
    b. Validation (26).jpg
    c. Validation (101).jpg