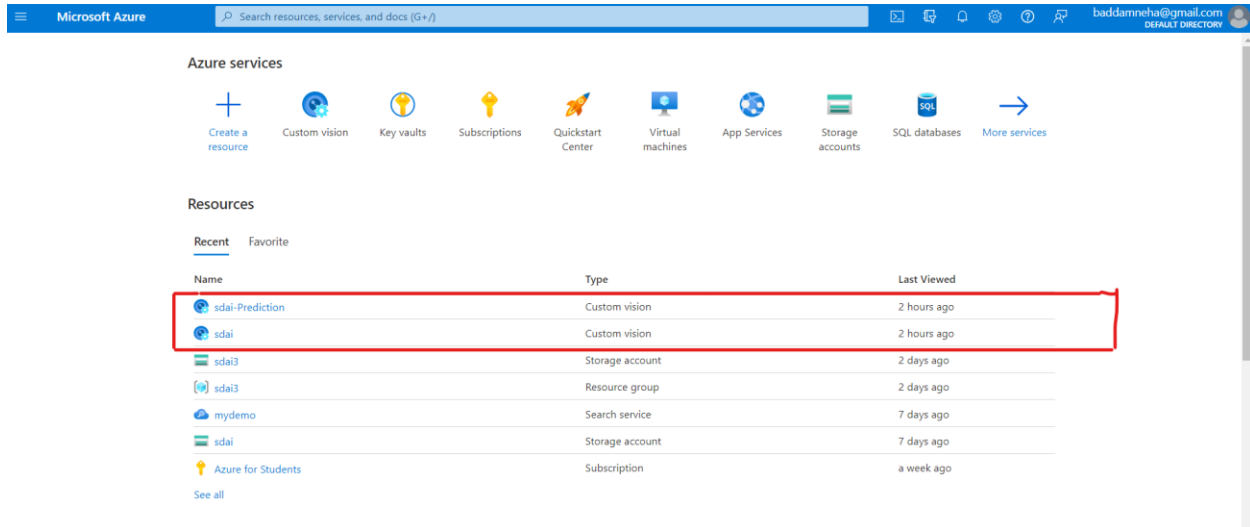


## Pre-requisites

### Azure Setup

Firstly, we must create a Custom Vision Project with training and prediction resources.

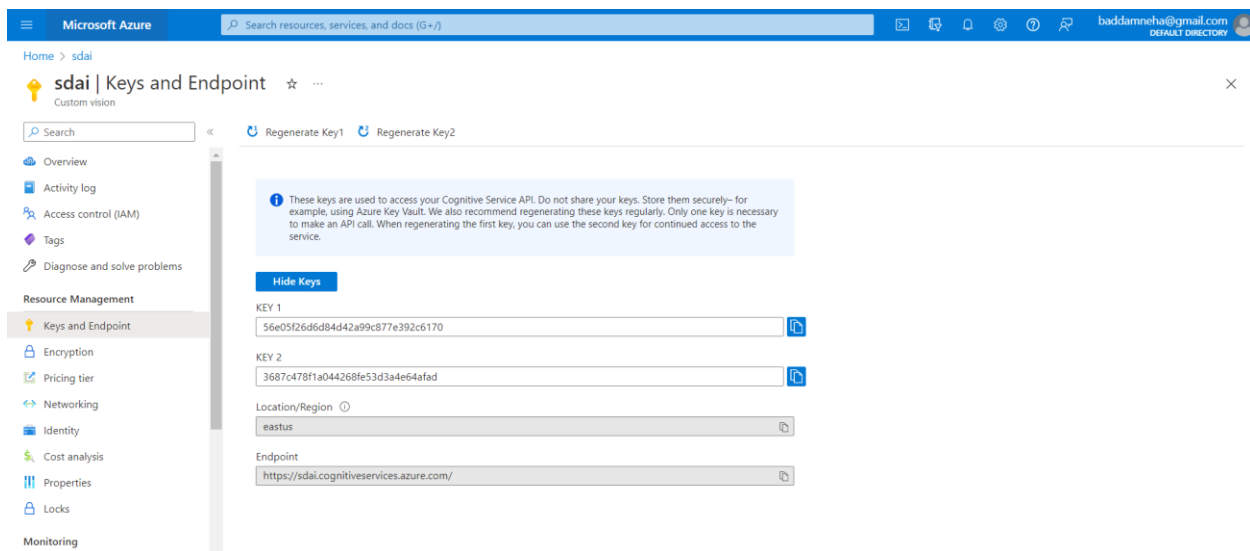


The screenshot shows the Microsoft Azure portal interface. At the top, there's a search bar and navigation icons. Below, the 'Azure services' section lists various services like Custom vision, Key vaults, Subscriptions, etc. The 'Resources' section is active, showing a table of resources. The first two rows are highlighted with a red box:

Name	Type	Last Viewed
sdai-Prediction	Custom vision	2 hours ago
sdai	Custom vision	2 hours ago

Then note down the keys and endpoints of training and prediction resources.

Training key and Endpoint:



The screenshot shows the 'sdai | Keys and Endpoint' page in the Azure portal. The page displays the keys and endpoint for the Custom vision resource. The keys are:

- KEY 1: 56e05f26d6d84d42a99c877e392c6170
- KEY 2: 3687c478f1a044268fe53d3a4e64afad

The Location/Region is set to 'eastus'. The Endpoint is 'https://sdai.cognitiveservices.azure.com/'.

Prediction key, endpoint, and Resource ID:

## ICE-4

Home > sdai-Prediction

### sdai-Prediction | Keys and Endpoint

Custom vision

Search

Regenerate Key1 Regenerate Key2

Overview  
Activity log  
Access control (IAM)  
Tags  
Diagnose and solve problems

Resource Management

Keys and Endpoint

Encryption  
Pricing tier  
Networking  
Identity  
Cost analysis  
Properties  
Locks

Monitoring

Alerts

These keys are used to access your Cognitive Service API. Do not share your keys. Store them securely—for example, using Azure Key Vault. We also recommend regenerating these keys regularly. Only one key is necessary to make an API call. When regenerating the first key, you can use the second key for continued access to the service.

Hide Keys

KEY 1  
e9eb4c1a158c4bfc8da8397bb161ef2c

KEY 2  
d67f14c3c51b4abca9c979fec3b5b11

Location/Region  
eastus

Endpoint  
https://sdai-prediction.cognitiveservices.azure.com/

Home > sdai-Prediction

### sdai-Prediction | Properties

Custom vision

Search

Overview  
Activity log  
Access control (IAM)  
Tags  
Diagnose and solve problems

Resource Management

Keys and Endpoint

Encryption  
Pricing tier  
Networking  
Identity  
Cost analysis  
Properties  
Locks

Monitoring

Alerts

Status  
Active

Pricing tier  
Standard

Subscription name  
Azure for Students  
[Change subscription](#)

Subscription ID  
5cf92398-d8a0-400a-9d32-05fcb62b046b

Resource group  
sdai3  
[Change resource group](#)

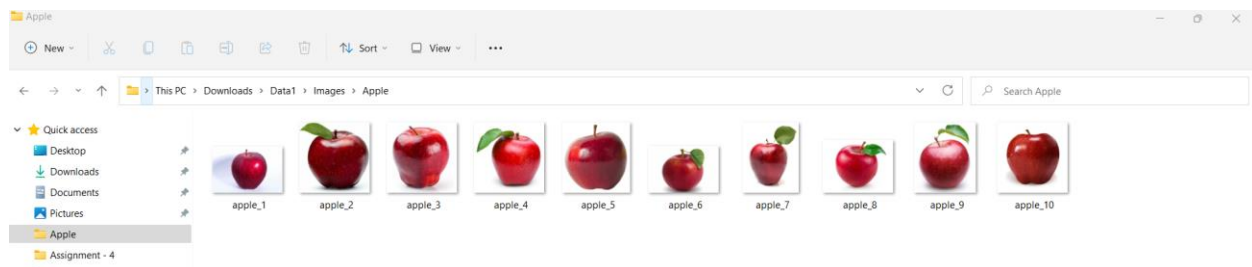
Resource ID  
/subscriptions/5cf92398-d8a0-400a-9d32-05fcb62b046b/resourceGroups/sdai3/providers/Microsoft.CognitiveServices/accounts/sdai-Prediction

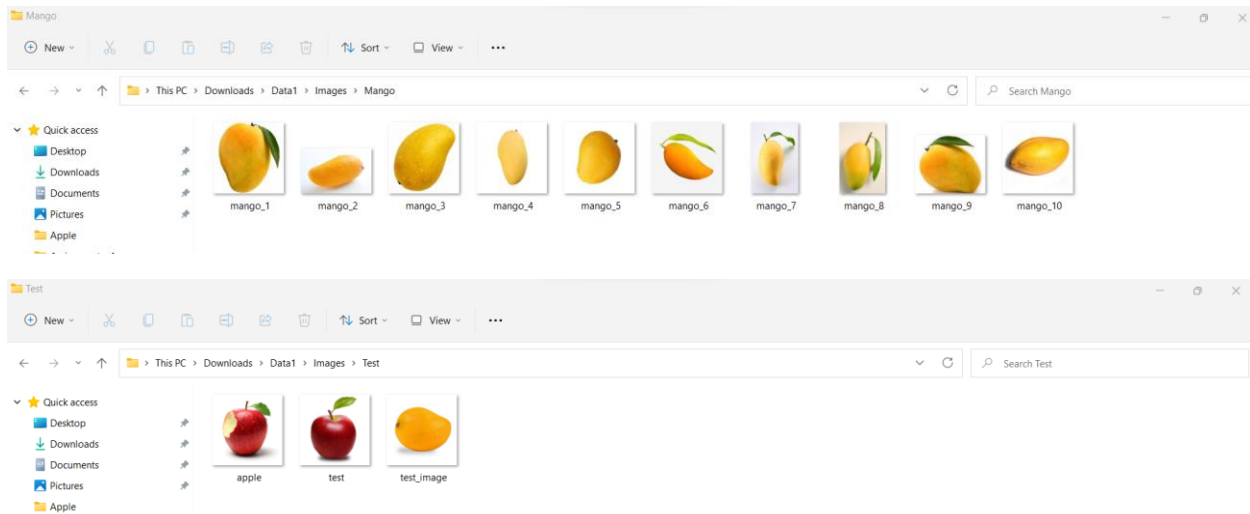
Date Created  
9/23/2022, 2:53:41 PM

Location  
East US

## Dataset setup

On your local storage create a Data folder and upload images for training and prediction.





## Package Setup

Run the below command in the command prompt to install custom vision packages for code execution.

```
pip install azure-cognitiveservices-vision-customvision
```

## Image classification with the Azure Custom Vision

Now, import all the packages required for image classification. Paste all the keys and endpoints in the code.

```
In [27]: from azure.cognitiveservices.vision.customvision.training import CustomVisionTrainingClient
from azure.cognitiveservices.vision.customvision.prediction import CustomVisionPredictionClient
from azure.cognitiveservices.vision.customvision.training.models import ImageFileCreateBatch, ImageFileCreateEntry, Region
from msrest.authentication import ApiKeyCredentials
import os, time, uuid

In [28]: # Replace with valid values
ENDPOINT = "https://sdai.cognitiveservices.azure.com/"
PredictionENDPOINT = "https://sdai-prediction.cognitiveservices.azure.com/"
training_key = "56e05f26d6d84d42a99c877e392c6170"
prediction_key = "e9eb4c1a158c4bfc8da8397bb161ef2c"
prediction_resource_id = "/subscriptions/5cf92398-d8a0-400a-9d32-05fcb62b046b/resourceGroups/sdai3/providers/Microsoft.Cognitive"
```

We are validating the API key of both training and prediction and using the key and endpoint to create Custom Vision client objects for both training and prediction.

```
In [29]: credentials = ApiKeyCredentials(in_headers={"Training-key": training_key})
trainer = CustomVisionTrainingClient(ENDPOINT, credentials)
prediction_credentials = ApiKeyCredentials(in_headers={"Prediction-key": prediction_key})
predictor = CustomVisionPredictionClient(PredictionENDPOINT, prediction_credentials)
```

After creating the custom vision project, we now create a custom vision project using the “create\_project” function.

```
In [30]: publish_iteration_name = "classifyModel"

credentials = ApiKeyCredentials(in_headers={"Training-key": training_key})
trainer = CustomVisionTrainingClient(ENDPOINT, credentials)

# Create a new project
print ("Creating project...")
project_name = uuid.uuid4()
project = trainer.create_project(project_name)
print('d')

Creating project...
d
```

After creating the project, we created two tags namely mango and apple.

```
In [31]: # Make two tags in the new project
mango_tag = trainer.create_tag(project.id, "Mango")
apple_tag = trainer.create_tag(project.id, "Apple")
```

Now, we upload the images from the local storage in the system by giving the directory path of the folder that contains the Dataset. Each image is loaded one after the other into the respective tags. If the images are not uploaded successfully, an error with status is printed.

#### Upload and tag images

```
In [32]: #_file_ : Put Location where your "images" folder is Located in your system

base_image_location = os.path.join (os.path.dirname("C:/Users/badda/Downloads/Data1/"), "Images")

print("Adding images...")

image_list = []

for image_num in range(1, 11):
    file_name = "mango_{}.jpg".format(image_num)
    with open(os.path.join (base_image_location, "Mango", file_name), "rb") as image_contents:
        image_list.append(ImageFileCreateEntry(name=file_name, contents=image_contents.read(), tag_ids=[mango_tag.id]))

for image_num in range(1, 11):
    file_name = "apple_{}.jpg".format(image_num)
    with open(os.path.join (base_image_location, "Apple", file_name), "rb") as image_contents:
        image_list.append(ImageFileCreateEntry(name=file_name, contents=image_contents.read(), tag_ids=[apple_tag.id]))

upload_result = trainer.create_images_from_files(project.id, ImageFileCreateBatch(images=image_list))
if not upload_result.is_batch_successful:
    print("Image batch upload failed.")
    for image in upload_result.images:
        print("Image status: ", image.status)
    exit(-1)

Adding images...
```

Now we train the project with the above-loaded images using the train\_project function. Once the training is completed, the training status is printed as completed.

## Train the project

```
In [33]: print ("Training...")
iteration = trainer.train_project(project.id)
while (iteration.status != "Completed"):
    iteration = trainer.get_iteration(project.id, iteration.id)
    print ("Training status: " + iteration.status)
    print ("Waiting 10 seconds...")
    time.sleep(10)
```

[illegible]

Once the training is completed, we must publish the current completed iteration to the project endpoint using the “publish iteration” function.

### Publish the current iteration

```
In [34]: # The iteration is now trained. Publish it to the project endpoint
trainer.publish_iteration(project.id, iteration.id, publish_iteration_name, prediction_resource_id)
print("Done!")
```

Done!

Once the training is published, we now use the test images to predict and classify the image. In the below code, I have used the image of a mango and the prediction shows that it's 100% mango.

## Test the prediction endpoint

```
In [35]: # Now there is a trained endpoint that can be used to make a prediction
prediction_credentials = ApiKeyCredentials(in_headers={"Prediction-key": prediction_key})
predictor = CustomVisionPredictionClient(PredictionENDPOINT, prediction_credentials)

with open(os.path.join(base_image_location, "Test/test_image.jpg"), "rb") as image_contents:
    results = predictor.classify_image(
        project.id, publish_iteration_name, image_contents.read())

# Display the results.
for prediction in results.predictions:
    print("\t" + prediction.tag_name +
          ": {:.2f}%".format(prediction.probability * 100))

Mango: 100.00%
Apple: 0.00%
```

In the below code, I have used the image of an apple and the prediction shows that it's 100% apple.

```
In [36]: # Now there is a trained endpoint that can be used to make a prediction
prediction_credentials = ApiKeyCredentials(in_headers={"Prediction-key": prediction_key})
predictor = CustomVisionPredictionClient(PredictionENDPOINT, prediction_credentials)

with open(os.path.join(base_image_location, "Test/test.jpg"), "rb") as image_contents:
    results = predictor.classify_image(
        project.id, publish_iteration_name, image_contents.read())

# Display the results.
for prediction in results.predictions:
    print("\t" + prediction.tag_name +
          ": {:.2f}%".format(prediction.probability * 100))

Apple: 100.00%
Mango: 0.00%
```

Now, I have written a code to upload the image from the local drive in runtime. Here, we must copy and paste the directory of the actual test image folder and the test image filename. Once the input is given, the image is classified, and prediction is displayed.

#Task 4: Make a small code toolkit where you upload the image in runtime and it performs classification. #You have to use same ipynb file to perform the task. (40%)

```
In [48]: # below is a way to dynamically upload the image by giving image directory and image file name as input to perform classification
prediction_credentials = ApiKeyCredentials(in_headers={"Prediction-key": prediction_key})
predictor = CustomVisionPredictionClient(PredictionENDPOINT, prediction_credentials)

import os
base_image_location1 = input("Enter test image directory path ")
base_image_location1 = str(base_image_location1).replace(os.path.sep, '/') + '/'
image = input("Enter jpg image filename without extension ") + '.jpg'

with open(os.path.join(base_image_location1, image), "rb") as image_contents:
    results = predictor.classify_image(
        project.id, publish_iteration_name, image_contents.read())

# Display the results.
for prediction in results.predictions:
    print("\t" + prediction.tag_name +
          ": {:.2f}%".format(prediction.probability * 100))

Enter test image directory path C:/Users/badda/Downloads/Data1/Images/Test/
Enter jpg image filename without extension apple
Apple: 100.00%
Mango: 0.00%
```

Below is one more way to dynamically upload images in the runtime. Here we can directly select the images from the dropdown as shown below from the file directory that has been passed to the “filedir” variable in the below code.

In [51]: *#Below is one more way to upload images at runtime to perform classification*

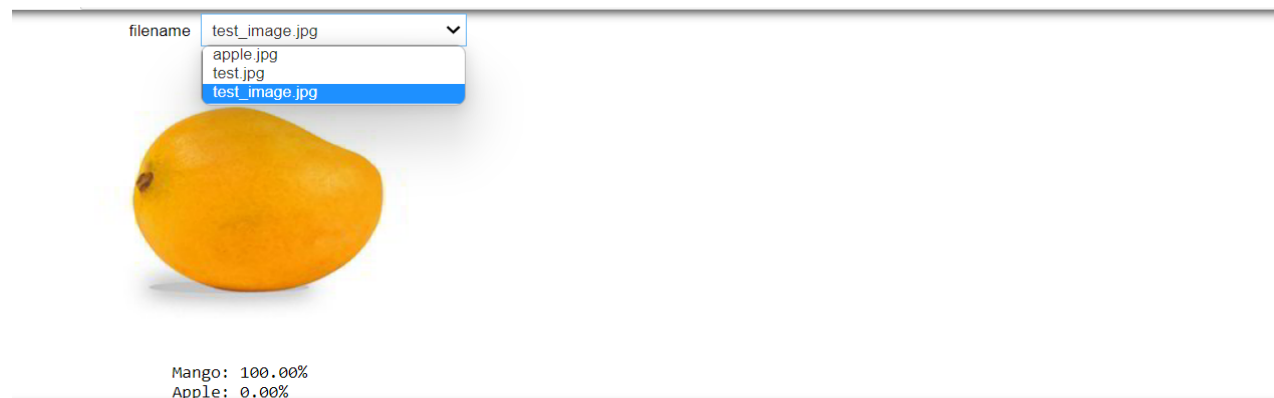
```
import os
from IPython.display import Image
import ipywidgets as widgets
from ipywidgets import interact, interact_manual

#download anu image to the below directory
filedir = "C:/Users/badda/Downloads/Data1/Images/Test/"

#below function is used to select the image at runtime
@interact
def show_images(filename=os.listdir(filedir)):
    display(Image(filedir+filename))
    with open(os.path.join (filedir, filename), "rb") as image_contents:
        results = predictor.classify_image(
            project.id, publish_iteration_name, image_contents.read())

    # Display the results.
    for prediction in results.predictions:
        print("\t" + prediction.tag_name +
              ": {:.2f}%".format(prediction.probability * 100))
```

We can download and save any number of images into the file directory mentioned in the code to display in the drop-down. We can select any image from the drop-down. When we dynamically change the images, that image is classified and a prediction is printed for that image that has been selected.



filename apple.jpg



Apple: 100.00%  
Mango: 0.00%



## ICE-4

