SIT 788: Assignment 5.2

Computer Vision and Custom Vision

Part 1: Real time face detection application

This part has walkthrough for creating a real time face detection application. The resource used (Azure Face detection API [1]) is same from the 5.1 Task.

Step 1: Creating face detection resource

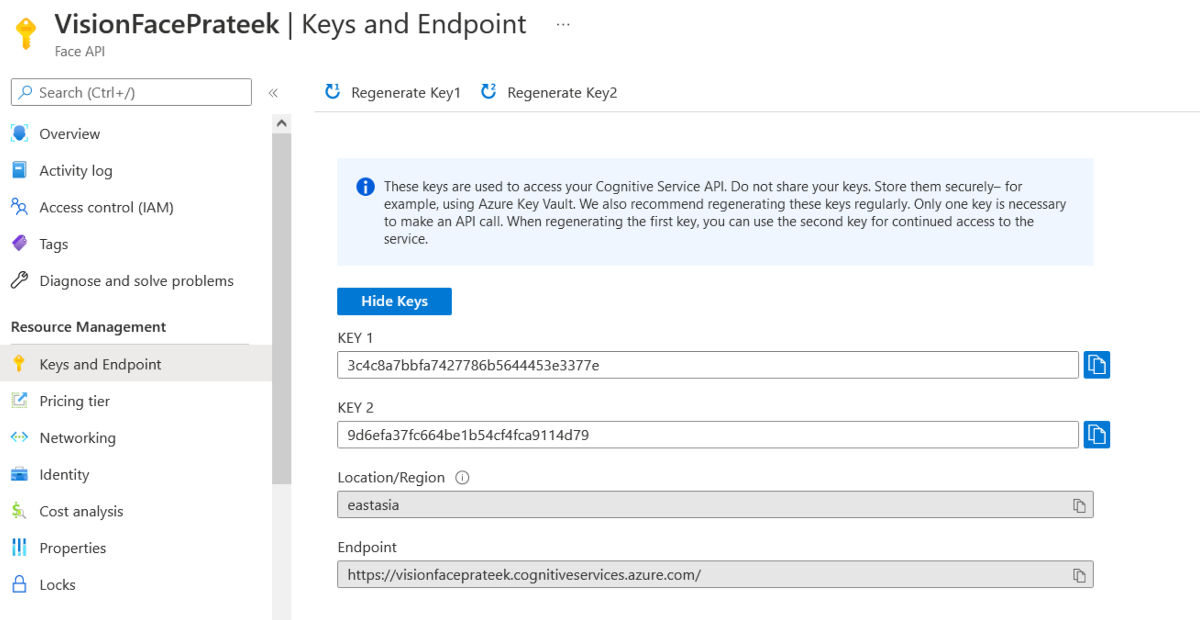


Fig 1.1: Resource used for face detection.

Step 2: Import libraries and connect to the resource

Here we import the libraries and connect to the face detection resource using the generated subscription key and endpoint.

Part 1: Real-time face detection

```
In [1]: from msrest.authentication import CognitiveServicesCredentials
        from azure.cognitiveservices.vision.face import FaceClient

        import cv2
        from IPython.display import Image
        from matplotlib import pyplot as plt
        from time import time
```

```
In [2]: attributes_list = ['age', 'emotion', 'gender']
        fps = 1
        fps = fps * 1000

        subscription_key = "3c4c8a7bbfa7427786b5644453e3377e"
        endpoint = "https://visionfaceprateek.cognitiveservices.azure.com/"
        cv_face_client = FaceClient(
            endpoint, CognitiveServicesCredentials(subscription_key))
        print("Connected to resource", cv_face_client)

        Connected to resource <azure.cognitiveservices.vision.face._face_client.FaceClient object at 0x000001C4FFD48040>
```

Fig 2.1: Import libs and connect to the face detection resource

Step 3: Define function to process passed frame and process results

Using the resource, a function *process_frame* is defined which sends the frame to the azure face detection API. A predefined attribute list is also used to generate additional information (age, emotion, and gender).

```
In [3]: def process_frame(frame2api):
            cv2.imwrite('tmp_image.jpg', frame2api)
            tmp_img = open("tmp_image.jpg", 'rb')
            detected_faces = cv_face_client.face.detect_with_stream(
                tmp_img, return_face_attributes=attributes_list)
            tmp_img.close()
            return detected_faces
```

Fig 3.1: Function to detect faces

Another function to process the results and overlay annotations on detection frame is defined as *process results.*

```
In [4]: font = cv2.FONT_HERSHEY_SIMPLEX
        fontScale = 1
        color = (0, 0, 255)
        thickness = 2


        def process_result(frame2api, detected_faces):
            if len(detected_faces) > 0:
                print("Number of faces detected:", len(detected_faces))
                for faces_itr in range(len(detected_faces)):
                    org = (detected_faces[faces_itr].face_rectangle.left,
                           detected_faces[faces_itr].face_rectangle.top)
                    gender = detected_faces[faces_itr].face_attributes.gender.value
                    age = detected_faces[faces_itr].face_attributes.age
                    emotions_dict = detected_faces[faces_itr].face_attributes.emotion.as_dict(
                    )
                    emotion_name = max(
                        zip(emotions_dict.values(), emotions_dict.keys()))[1]

                    face_rect = [detected_faces[faces_itr].face_rectangle.left,
                                 detected_faces[faces_itr].face_rectangle.top,
                                 detected_faces[faces_itr].face_rectangle.width,
                                 detected_faces[faces_itr].face_rectangle.height]

                    frame2api = cv2.rectangle(frame2api, face_rect, (0, 0, 255), 5)
                    frame2api = cv2.putText(frame2api, str(
                        gender), org, font, fontScale, color, thickness, cv2.LINE_AA)
                    frame2api = cv2.putText(frame2api, str(
                        age), (org[0]+100, org[1]), font, fontScale, color, thickness, cv2.LINE_AA)
                    frame2api = cv2.putText(frame2api, str(
                        emotion_name), (org[0] + 200, org[1]), font, fontScale, color, thickness, cv2.LINE_AA)
            else:
                print("No Face Detected")
            return len(detected_faces)
```

Fig 3.2: process_result() overlays age, gender and emotion information along with face bbox on the processed framed.

Step 4: Running live camera using OpenCV

OpenCV is used to initialize a live video stream using the webcam. As the purpose of the task is also to sample the frames such that 1 frame per second is passed, time difference between two successive processed frame is calculated. This ensures that there is at least 1 second difference between two consecutive processed frame, rest frames are skipped. This also ensures that the API is called efficiently and, in a cost-effective way.

In [5]:

```python
vid = cv2.VideoCapture(0)
start_time = time() * 1000
init_frame = True

while (True):
    _, frame = vid.read()
    if init_frame:
        frame2api = frame.copy()
        detected_faces = process_frame(frame2api)
        process_result(frame2api, detected_faces)
        init_frame = False

    end_time = time() * 1000
    time_diff = end_time - start_time
    if (int(time_diff) >= fps):
        frame2api = frame.copy()
        detected_faces = process_frame(frame2api)
        process_result(frame2api, detected_faces)
        start_time = time() * 1000

    frame = cv2.putText(frame, "Real Time", (10,10), font,
                        fontScale, color, thickness, cv2.LINE_AA)
    frame2api = cv2.putText(frame2api, "Processed", (50,50),
                            font, fontScale, color, thickness, cv2.LINE_AA)
    frame_merged = cv2.hconcat([frame, frame2api])
    cv2.imshow("Display", frame_merged)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
vid.release()
cv2.destroyAllWindows()
```

```
Number of faces detected: 1
Number of faces detected: 1
Number of faces detected: 1
Number of faces detected: 1
Number of faces detected: 1
Number of faces detected: 1
Number of faces detected: 1
Number of faces detected: 1
```

Fig 4.1: Using OpenCV to run video stream via webcam. 1 FPS is set as the default parameter.

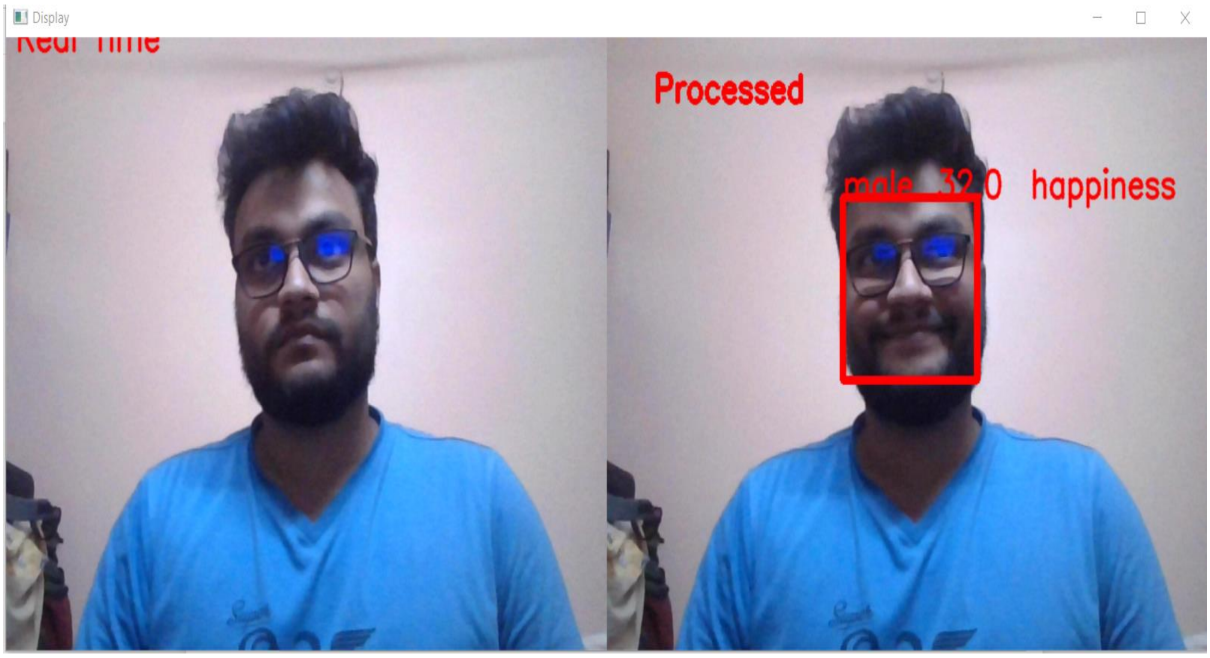


Fig 4.2: Output showing real time frame [Left] vs Last processed frame with face bbox and facial attributes overlayed [Right]

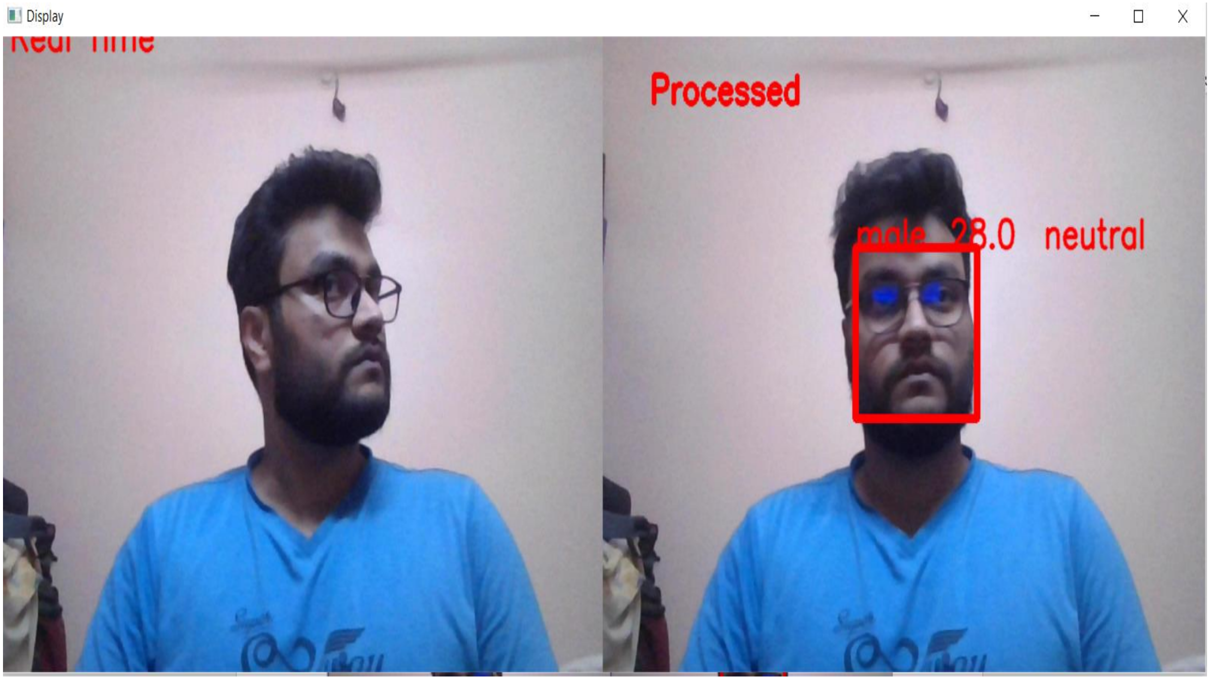


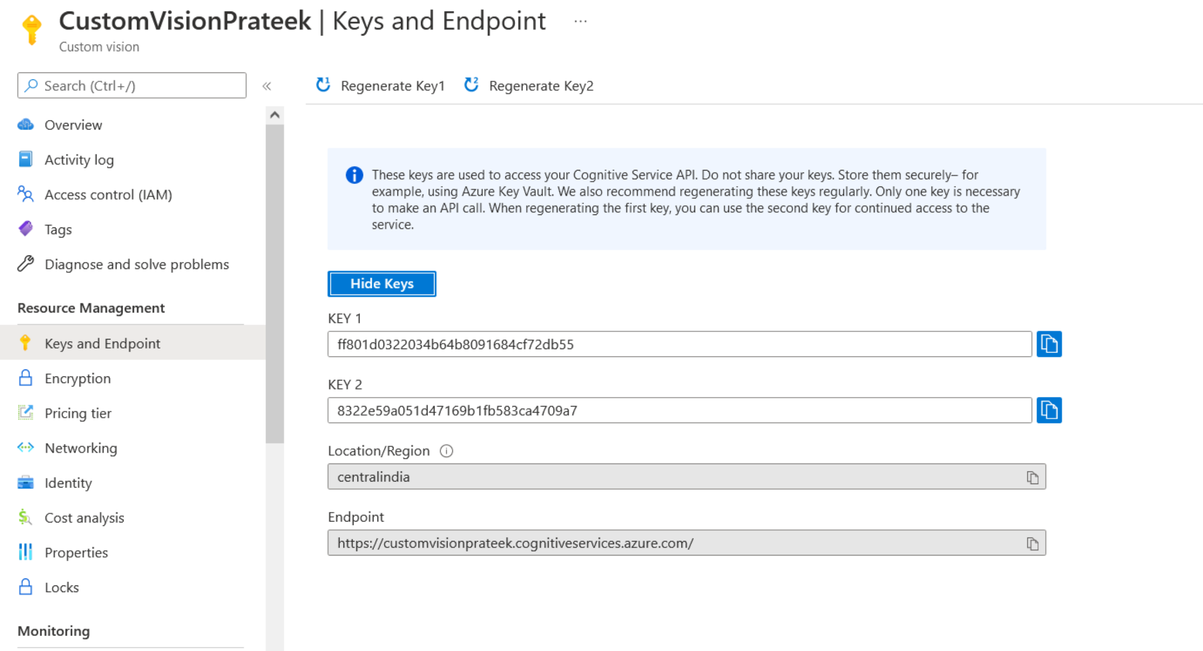
Fig 4.3: Output showing real time frame [Left] vs Last processed frame with face bbox and facial attributes overlayed [Right]

Part 2: Custom Vision

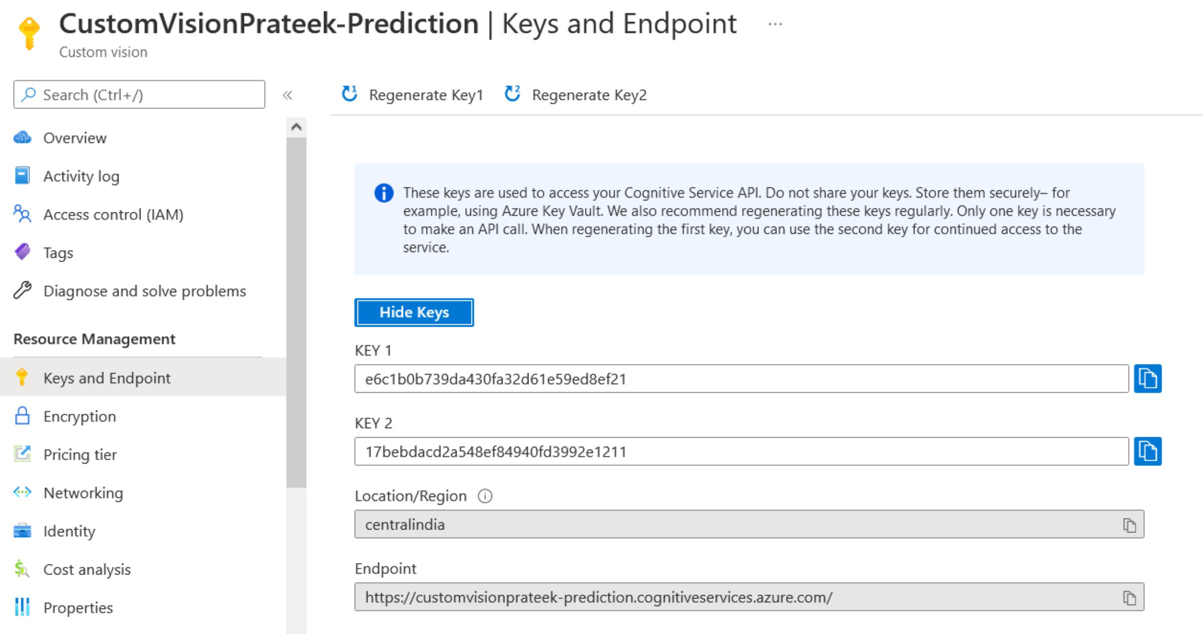
Custom Vision [3] is offered by Azure for the scenario when we need to use our own data and train a model from scratch. Currently, object detection and object classification are supported in this service. In this experiment, we use Dogs and Cats dataset [2] to train a classifier using Azure Custom Vision. This model is then deployed and tested using Azure Custom Vision.

Step 1: Creating a custom vision resource

Once the test resource is created via azure portal, we can see two resources: one for training and other for prediction.



(a)



(b)

Fig 1.1: (a) Keys and endpoint for training resource (b) Keys and endpoint for prediction resource.

Step 2: Installing and importing dependencies

After resource creation, python dependencies for custom vision are installed.

```
In [6]: !pip install azure-cognitiveservices-vision-customvision

Requirement already satisfied: azure-cognitiveservices-vision-customvision in c:\users\singh\anaconda3\lib\site-packages
(3.1.0)
Requirement already satisfied: azure-common~=1.1 in c:\users\singh\anaconda3\lib\site-packages (from azure-cognitiveservic
es-vision-customvision) (1.1.28)
Requirement already satisfied: msrest>=0.5.0 in c:\users\singh\anaconda3\lib\site-packages (from azure-cognitiveservices-v
ision-customvision) (0.6.21)
Requirement already satisfied: isodate>=0.6.0 in c:\users\singh\anaconda3\lib\site-packages (from msrest>=0.5.0->azure-cog
nitiveservices-vision-customvision) (0.6.1)
Requirement already satisfied: requests-oauthlib>=0.5.0 in c:\users\singh\anaconda3\lib\site-packages (from msrest>=0.5.0-
>azure-cognitiveservices-vision-customvision) (1.3.1)
Requirement already satisfied: requests~=2.16 in c:\users\singh\anaconda3\lib\site-packages (from msrest>=0.5.0->azure-cog
nitiveservices-vision-customvision) (2.25.1)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\singh\anaconda3\lib\site-packages (from msrest>=0.5.0->azure
-cognitiveservices-vision-customvision) (2020.12.5)
Requirement already satisfied: six in c:\users\singh\anaconda3\lib\site-packages (from isodate>=0.6.0->msrest>=0.5.0->azur
e-cognitiveservices-vision-customvision) (1.15.0)
Requirement already satisfied: chardet<5,>=3.0.2 in c:\users\singh\anaconda3\lib\site-packages (from requests~=2.16->msres
t>=0.5.0->azure-cognitiveservices-vision-customvision) (4.0.0)
Requirement already satisfied: idna<3,>=2.5 in c:\users\singh\anaconda3\lib\site-packages (from requests~=2.16->msrest>=0.
5.0->azure-cognitiveservices-vision-customvision) (2.10)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\singh\anaconda3\lib\site-packages (from requests~=2.16->m
srest>=0.5.0->azure-cognitiveservices-vision-customvision) (1.26.4)
Requirement already satisfied: oauthlib>=3.0.0 in c:\users\singh\anaconda3\lib\site-packages (from requests-oauthlib>=0.5.
0->msrest>=0.5.0->azure-cognitiveservices-vision-customvision) (3.2.0)
```

```
In [153]: 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
```

Fig 2.1: Installing custom vision and importing libs

Step 3: Connecting to the resource

The resource created in step 1 is connected via subscription key and endpoints. Two clients, one for training and one for prediction are created.

Note: The subscription keys and endpoints for prediction and training are different, hence they are connected separately.

```
In [154]: endpoint = "https://customvisionprateek.cognitiveservices.azure.com/"
          training_key = "ff801d0322034b64b8091684cf72db55"
          prediction_key = "e6c1b0b739da430fa32d61e59ed8ef21"
          prediction_resource_id = "/subscriptions/d5d0e7cd-5900-4c80-820c-b24c2a8416d0/resourceGroups/PRATEEK-SINGH/providers/Microsc
```

```
In [155]: credentials = ApiKeyCredentials(in_headers={"Training-key": training_key})
          trainer = CustomVisionTrainingClient(endpoint, credentials)
          prediction_credentials = ApiKeyCredentials(in_headers={"Prediction-key": prediction_key})
          predictor = CustomVisionPredictionClient(endpoint, prediction_credentials)
```

```
In [156]: publish_iteration_name = "classifyDogCat-v3"

          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)

          Creating project...
```

Fig 3.1: Clients for training and prediction. Project created using generated credentials.

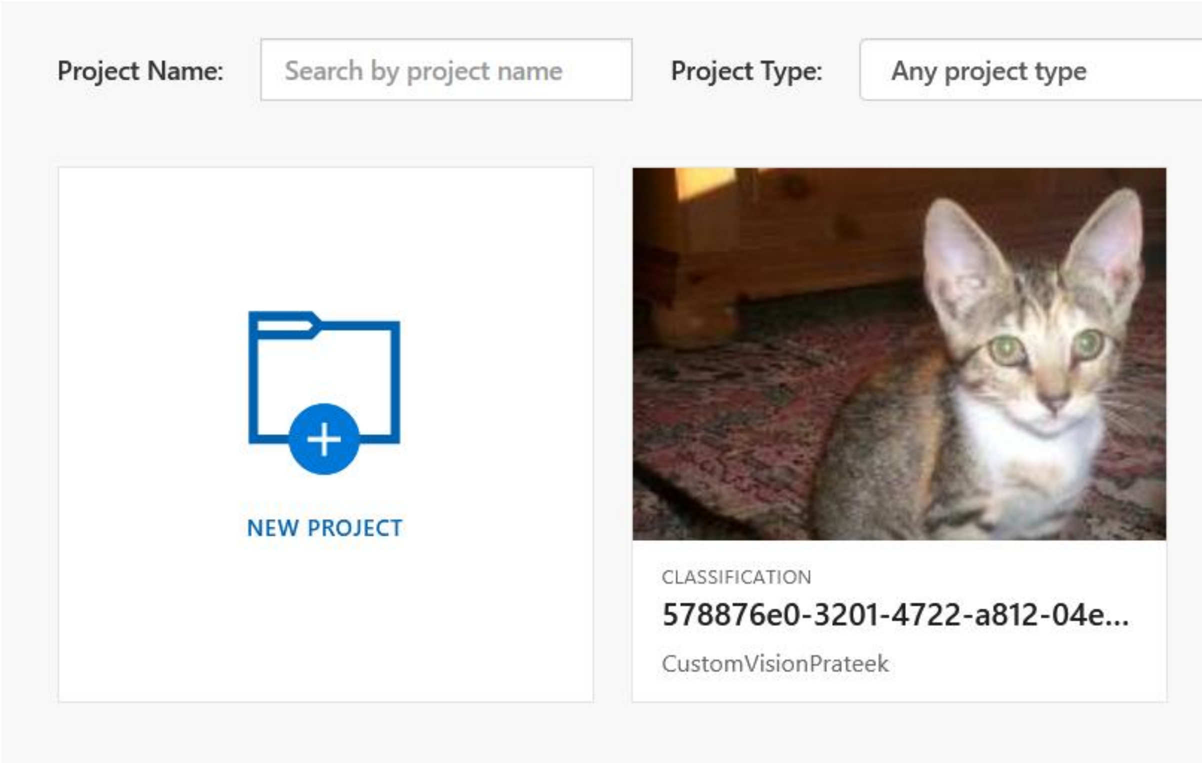


Fig 3.2: Once the project is created using the SDK, it is reflected in the custom vision ai portal.

Step 4: Uploading data and training the classifier

After the project is created, the images are uploaded using the SDK *[the SDK only allows 64 images to be uploaded, once the limit is reached then the images can be uploaded again but not in the same go; no such limitation is on the custom vision webpage].*

Once the images are uploaded the training is done and model is published on the custom vison portal.

Train using the dwonloaded dataset

```
In [159]: base_image_location = os.path.dirname("C:\\Users\\singh\\Documents\\MastersAppliedAIDeakin\\"
                                               "SIT788_Engineering_AI_Solutions\\Assignments\\SIT788_5_2_Data\\dataset\\training_set\

          print("Adding images...")

          image_list = []
          step = 32
          itr = 32
          #for itr in range(0, 100, step):
          for image_num in range(itr, itr+step):
              file_name = "dog.{}.jpg".format(image_num+1)
              with open(os.path.join (base_image_location, "dogs", file_name), "rb") as image_contents:
                  image_list.append(ImageFileCreateEntry(name=file_name, contents=image_contents.read(), tag_ids=[dog_tag.id]))

          for image_num in range(itr, itr+step):
              file_name = "cat.{}.jpg".format(image_num+1)
              with open(os.path.join (base_image_location, "cats", file_name), "rb") as image_contents:
                  image_list.append(ImageFileCreateEntry(name=file_name, contents=image_contents.read(), tag_ids=[cat_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)
          else:
              print("Adding images successful")

          Adding images...
          Adding images successful
```

Fig 4.1: Uploading data

```
In [160]: iteration = trainer.train_project(project.id)
          while (iteration.status != "Completed"):
              iteration = trainer.get_iteration(project.id, iteration.id)
              print ("Training status: " + iteration.status)
              time.sleep(10)
          print ("Training Done", iteration.status)
          trainer.publish_iteration(project.id, iteration.id, publish_iteration_name, prediction_resource_id)
          print ("Done!")

          Training status: Training
          Training status: Training
          Training status: Training
          Training status: Training
          Training status: Training
          Training status: Training
          Training status: Training
          Training status: Training
          Training status: Training
          Training status: Training
          Training status: Training
          Training status: Training
          Training status: Training
          Training status: Training
          Training status: Training
          Training status: Training
          Training status: Training
          Training status: Training
          Training status: Training
          Training status: Training
```

Fig 4.2: Training and publishing model

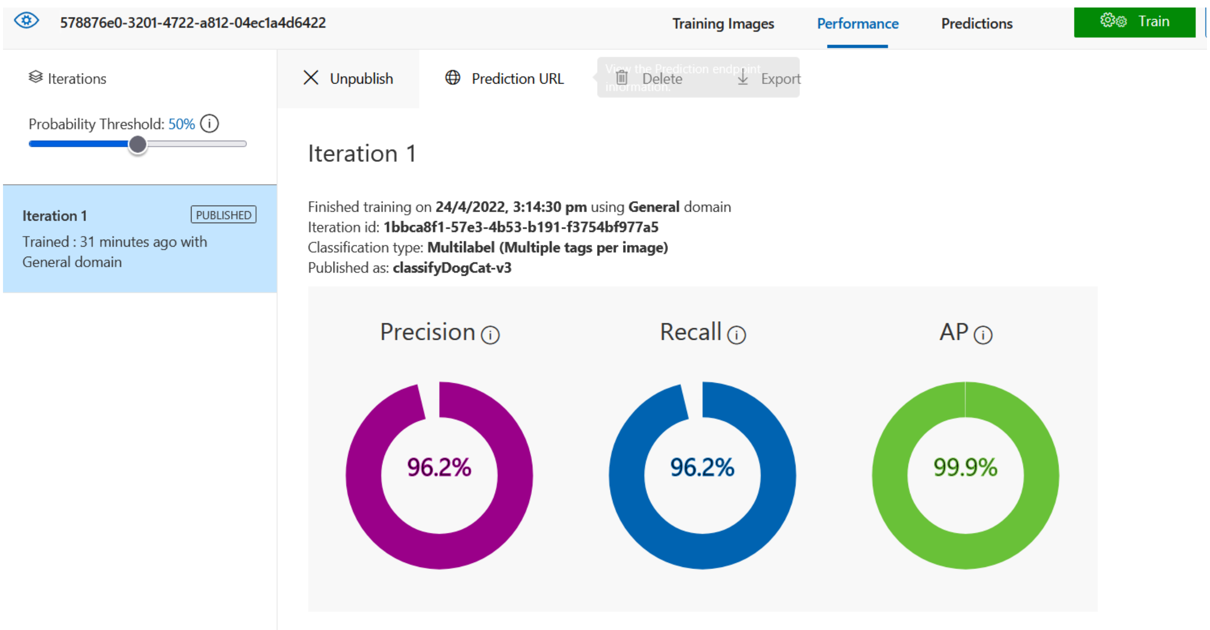


Fig 4.3: Published model stats for training on custom vision webpage.

Step 5: Testing using test data

As the data is two class, it can be treated as a binary classification. The class with higher probability is the predicted class *[The sum of probabilities is always 100, so the dominant class will have probability greater than 50%*]. The testing set from [2] has 1000 samples for dogs and cats each. All these images are sent to the prediction resource and evaluation metrics is calculated for binary classification.

```
In [172]: import glob
          all_dogs_images = glob.glob(test_image_location + "/dogs/*")
          all_cats_images = glob.glob(test_image_location + "/cats/*")
```

```
In [186]: from tqdm import tqdm
          TP = 0
          FN = 0
          TN = 0
          FP = 0


          for image_dog in tqdm(all_dogs_images):
              with open(os.path.join (test_image_location, image_dog), "rb") as image_contents:
                  results = predictor.classify_image(
                  project.id, publish_iteration_name, image_contents.read())

                  for prediction in results.predictions:
                      if prediction.probability > 0.5:
                          prediction_name = prediction.tag_name
                  if (prediction_name == "Dog"):
                      TP += 1
                  else:
                      FN += 1

          for image_cat in tqdm(all_cats_images):
              with open(os.path.join (test_image_location, image_cat), "rb") as image_contents:
                  results = predictor.classify_image(
                  project.id, publish_iteration_name, image_contents.read())

                  for prediction in results.predictions:
                      if prediction.probability > 0.5:
                          prediction_name = prediction.tag_name
                  if (prediction_name == "Cat"):
                      TN += 1
                  else:
                      FP += 1

          100%|██████████████████████████████████████████████| 1000/1000 [07:12<00:00,  2.31it/s]
          100%|██████████████████████████████████████████████| 1000/1000 [07:06<00:00,  2.34it/s]
```

Fig 5.1: Testing cats and dogs' folder with the trained classifier.

```
In [178]: Recall = TP / (TP + FN)
          Precision = TP / (TP + FP)
          F_Measure = (2 * Precision * Recall) / (Precision + Recall)
          Accuracy = (TP + TN)/(TP + TN + FP + FN)
```

```
In [185]: total = TP + TN + FP + FN
          print ("Total classified samples: ", total)
          print ("Evaluation metrics: Recall: ", Recall, "Precision: ", Precision,
                 "F-Measure: ", F_Measure, "Accuracy: ", Accuracy)

          Total classified samples:  2000
          Evaluation metrics: Recall:  0.981 Precision:  0.9969512195121951 F-Measure:  0.9889112903225806 Accuracy:  0.989
```

Fig 5.2: Evaluation metrics for the trained classifier.

The classifier is trained well considering only 64 images per class were used to train the classifier. The evaluation metrics for the classifier are:

- Precision: 99.6%
- Recall: 98.1%
- f-Measure: 98.9%
- Accuracy: 98.9%

References:

[1] https://azure.microsoft.com/en-us/services/cognitive-services/face/

[2] https://www.kaggle.com/datasets/chetankv/dogs-cats-images

[3] https://azure.microsoft.com/en-us/services/cognitive-services/custom-vision-service/