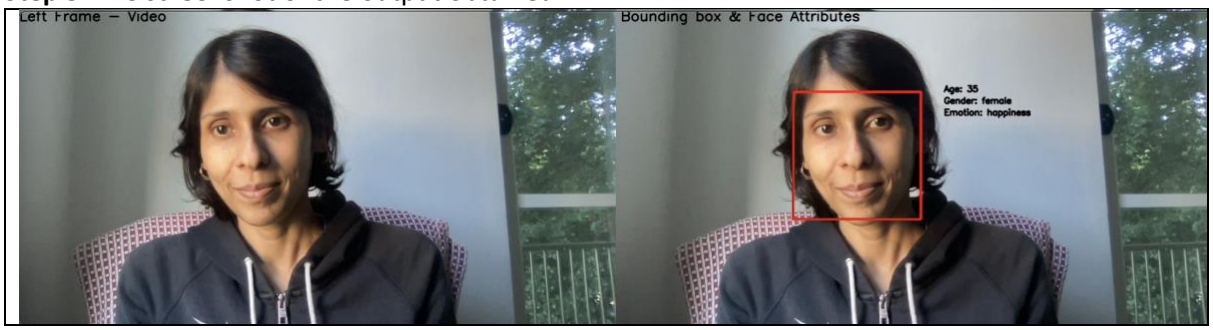
Task 5.2D Custom Vision Classifier

Answer 1

Objective:

The primary objective of this task is to capture face from the webcam in real-time and pass the image to Azure Face Detection API to generate a near real time prediction with 1 second delay. The API will predict the gender, age and emotion of the person and generate the prediction in the near real time. This output is used to draw the bounding box on face region and print the face attribute text in the right-side window.

I have used following steps to build this application.
Step 1: Created an API key and Endpoint for the Face Detection API.
Step 2: Built an application using OpenCV. The application has two side-by-side windows. The left window is showing the direct screen capture from my webcam. The application is capturing the screen image and then sending the frame from the webcam every 1 second to the Face API
Step 3: The face API is sending back the bounding rectangle coordinates and face attributes to the application.
Step 4: The application is then drawing the face rectangle and displaying the face attributes in the right window.
Step 5: The screenshot of the output obtained.



Embedding the Code below:

Importing Required modules and libraries

```
import cv2
import time
import threading
from azure.cognitiveservices.vision.face import FaceClient
from msrest.authentication import CognitiveServicesCredentials
```

The code has been created to create 2 OpenCV frames for showing the Video feeds – 1 in Real-time and other with a delay of 1 second. While the Real-time feed frame runs in Main thread, the Face detector is created in a thread.

```
# Gets the Real-time video feed from the Webcam
def real_time_webcam_feed():
    global left_frame
    global right_frame
    webcam_feed = cv2.VideoCapture(0)
    _, first_frame = webcam_feed.read()
    first_frame = cv2.flip(first_frame, 1)
    # Copy the initial frame to both left and right frames to initialize them
    left_frame = first_frame
    right_frame = first_frame
    # Create the Face Detector thread
    face_detector = threading.Thread(target=face_detector_with_lag, args=())
    face_detector_started = False
    while True:
        _, frame = webcam_feed.read()
        frame = cv2.flip(frame, 1)
        # Copy the frame
        left_frame = frame.copy()
        # Put heading on the frame
        frame = cv2.putText(frame, "Left Frame - Video", (10, 35), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 0), 2, cv2.LINE_AA)
        # Create a split style display
        frame_full = cv2.hconcat([frame, right_frame])
        # Show the feed in a new window
        cv2.imshow("Face detection using Azure Face API", frame_full)
        # Allow a delay of 1 Millisecond for the video to render
        cv2.waitKey(1)
        if not face_detector_started:
            face_detector_started = True
            face_detector.start()
        # Terminating condition based on window state
        if cv2.getWindowProperty("Face detection using Azure Face API", cv2.WND_PROP_VISIBLE) < 1:
            break
    # Destroy all the windows
    cv2.destroyAllWindows()
```

The Face detector function that lags by 1 second

```
# Face Detector running in a separate thread with a lag of 1 Second
def face_detector_with_lag():
    global left_frame
    global right_frame
    attrs = ['emotion', 'age', 'gender']
    while left_frame is not None:
        time.sleep(1)
        frame = left_frame.copy()
        cv2.imwrite('tmp.jpg', frame)
        # Load the saved image to be streamed to face detection API
        img = open('tmp.jpg', "rb")
        # Pass the image to the detector webservice
        faces = face_client.face.detect_with_stream(img, return_face_attributes=attrs, detection_model='detection_01')
        # check if any face has been detected
        if len(faces) > 0:
            frame = draw_result(frame, faces[0])
            right_frame = frame
```

The utility functions

```python
# draw text for Face Attributes on the screen
def draw_text(frame, left, top, label):
    frame = cv2.putText(frame, label, (left, top), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 0), 2, cv2.LINE_AA)
    return frame


# Age label text to be displayed on the screen
def get_age_label(face_attributes):
    return "Age: " + str(int(face_attributes.age))


# Gender label text to be displayed on the screen
def get_gender_label(face_attributes):
    gender = (face_attributes.gender.split('.'))[0]
    return "Gender: " + str(gender)


# Emotion label text to be displayed on the screen
def get_emotion_label(face_attributes):
    emotion_dict = face_attributes.emotion.__dict__
    # Removing the additional properties from the dict
    # and adding only the attributes in the emotion class returned by the API
    keys = list(emotion_dict.keys())[1:8]
    values = list(emotion_dict.values())[1:8]

    # getting the max value to extract the emotion with highest probability
    max_value = max(values)
    max_index = values.index(max_value)
    emotion = keys[max_index]
    return "Emotion: " + str(emotion)
```

```python
# Coordinates for the Bounding Box
def get_coordinates(rectangle):
    left = rectangle.left
    top = rectangle.top
    right = rectangle.left + rectangle.width
    bottom = rectangle.top + rectangle.height
    return left, top, right, bottom


# Draws the Bounding Box and Face Attribute labels
def draw_result(frame, face):
    frame = cv2.putText(frame, "Bounding box & Face Attributes",
                        (10, 35), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 0), 2, cv2.LINE_AA)
    # Get Age from Face attributes of the face
    age = get_age_label(face.face_attributes)
    # Get Gender from Face attributes of the face
    gender = get_gender_label(face.face_attributes)

    # Get Emotion from Face attributes of the face
    emotion = get_emotion_label(face.face_attributes)
    # Get the coordinates of the rectangular bounding box
    left, top, right, bottom = get_coordinates(face.face_rectangle)
    # Draw the bounding box
    bounding_box_color = (0, 0, 255)
    frame = cv2.rectangle(frame, (left, top), (right, bottom), bounding_box_color, 3)
    # Draw the Face Attributes for Age, Gender, and Emotion
    frame = draw_text(frame, right + 50, top, age)
    frame = draw_text(frame, right + 50, top + 25, gender)
    frame = draw_text(frame, right + 50, top + 50, emotion)
    return frame
```

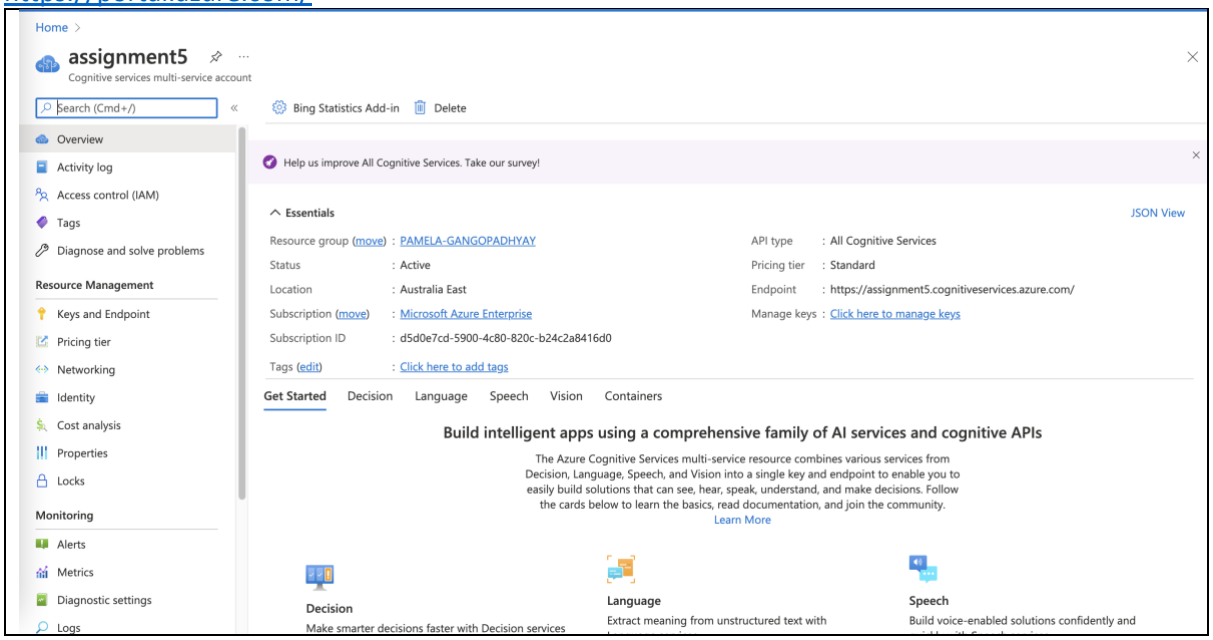
Answer 2

Objective:

The primary objective of this task is to create and train a supervised classification model that will classify dog vs cat from the Kaggle Cat and Dog dataset.

Dataset Source - https://www.kaggle.com/chetankv/dogs-cats-images

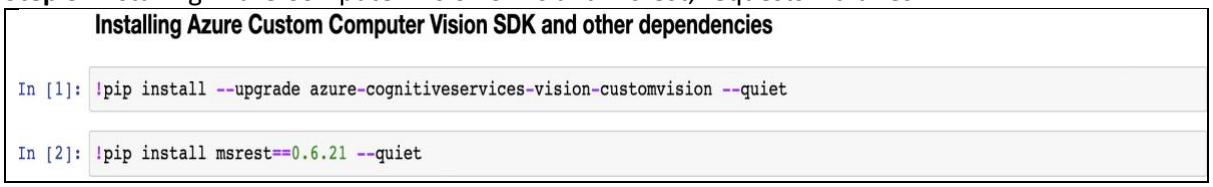
Step 1: Created a new **Azure Custom Vision Resource** for the computer vision service in the https://portal.azure.com/



Step 2: Login to **https://www.customvision.ai/** portal will autodetect the Cognitive Service account I am using and log into that one. The customvision.ai portal is showing the resource details created in the customvision.ai portal

Coding a Custom Classifier

Step 3: Installing Azure Computer Vision SDKs and msrest, requests Libraries



Step 4: Importing required modules

Step 5: I have copied the custom vision API key, the Resource ID from the custom vision portal and created a .env file. I have used python **dotenv** library to load the config data into memory

The Custom Vision API key and the endpoint information are loaded in an .env file in my development envi
Using loadenv from python dotenv module to load the key and endpoint into the memory

```
In [4]: load_dotenv()
Out[4]: True

In [5]: custom_vision_subscription_key = os.environ["CUSTOM_VISION_SUBSCRIPTION_KEY"]
        custom_vision_endpoint = os.environ["CUSTOM_VISION_ENDPOINT"]
        prediction_resource_id = os.environ["CUSTOM_VISION_RESOURCE_ID"]
```

Step 6: Creating the APIKeyCredentials object by passing 'custom_vision_subscription_key' in the header. This will authenticate the account to enable it for creating projects in the custom vision platform.

Creating the APIKeyCredentials object by passing 'custom_vision_subscription_key' in the header ¶

```
In [6]: creds = ApiKeyCredentials(in_headers={"Training-key": custom_vision_subscription_key})
```

Step 7: Creating the **CustomVisionClient** passing the Endpoint and the Credential parameters

Creating the CustomVisionTrainingClient and passing the Enpoint and the APIKeyCredential object

```
In [7]: model_train = CustomVisionTrainingClient(custom_vision_endpoint, creds)
```

Step 8: Creating the Classification project in the General [A2] domain

Getting all the available domains

```
In [8]: domains = model_train.get_domains()
        for domain in domains:
            print(domain.name)

General [A2]
General [A1]
General
Food
Landmarks
Retail
Adult
General (compact) [S1]
General (compact)
Food (compact)
Landmarks (compact)
Retail (compact)
General [A1]
General
Logo
Products on Shelves
General (compact) [S1]
General (compact)
```

Storing the domain id of General [A2] domain

```
In [9]: for domain in domains:
            if domain.name == 'General [A2]':
                domain_id = domain.id
                print(domain.id)
```

Step 10: Creating the classification project for the General [A2] domain by calling **create_project method of the CustomVisionClient** object

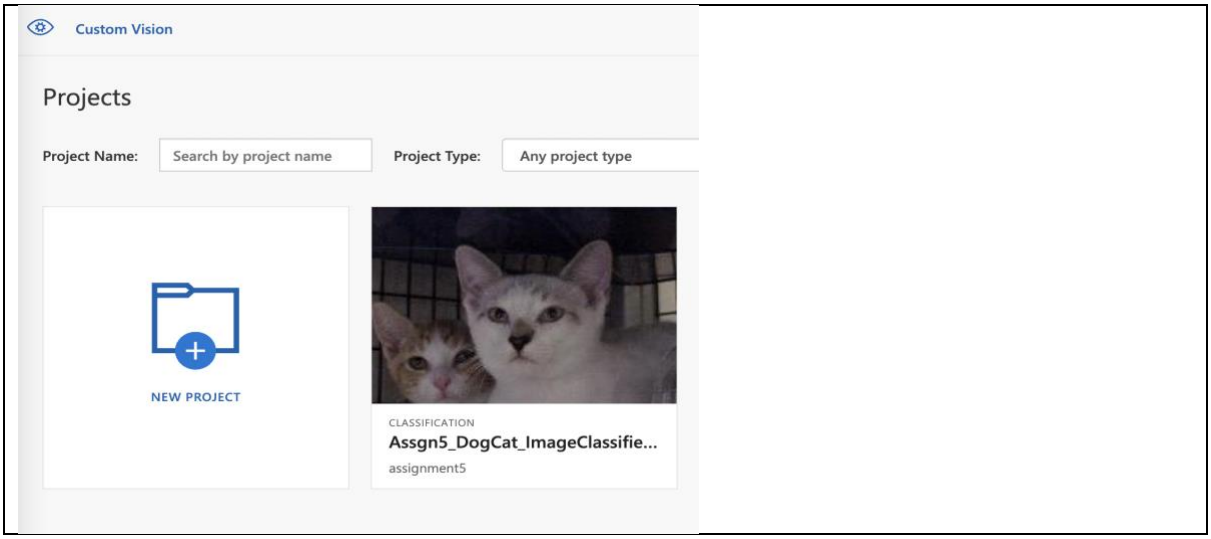
Creating a project in the General [A2] domain

```
In [10]: def create_project(domain_id, project_name):
             return model_train.create_project(name=project_name, domain_id=domain_id, classification_type="Multiclass")
```

Creating Project with domain_id and project_name

```
In [11]: project_name = f"Assgn5_DogCat_ImageClassifier_1"
         project = create_project(domain_id, project_name)
         print("Project created", project.id)
```

```
Project created 5a568323-a3da-43d0-95a4-36ff919464c3
```



Step 11: Creating tags for the categories to label the images by calling **create_tag()** method of the **CustomVisionClient** object. Tags are labels/categories the classifier will be using to label a particular image during Training and Inference

Creating tags for the categories to label the images

```
In [12]: cat_tag = model_train.create_tag(project.id, "cats_cls")
         dog_tag = model_train.create_tag(project.id, "dogs_cls")
```

Step 12: Training Dataset Batch upload. I have created 4 batches of data upload under the created tags

Uploading the training dataset into the customvision.ai platform

```
In [13]: def dir_walk(batch_name):
             tag_id = None
             for root, dirs, files in os.walk(batch_name, topdown=False):
                 try:
                     root_dir, class_name = root.split('/')
                     training_image_list = []
                     for image in files:
                         if image != ".DS_Store":
                             with open(os.path.join(root_dir, class_name, image), "rb") as contents:
                                 if class_name == "cats_cls":
                                     tag_id = cat_tag.id
                                 elif class_name == "dogs_cls":
                                     tag_id = dog_tag.id

                                 entry = ImageFileCreateEntry(name=image, contents=contents.read(), tag_ids=[tag_id])
                                 training_image_list.append(entry)

                     upload_status = model_train.create_images_from_files(project.id, ImageFileCreateBatch(images=training_image
                     if not upload_status.is_batch_successful:
                         for image in upload_status.images:
                             if image.status != "OK":
                                 print(image)
                             print("Image status: ", image.status)

                 except ValueError:
                     pass
```

Upload Batch 1

```
In [14]: dir_walk("train_batch_1")
```

Upload Batch 2

```
In [15]: dir_walk("train_batch_2")
```

Upload Batch 3

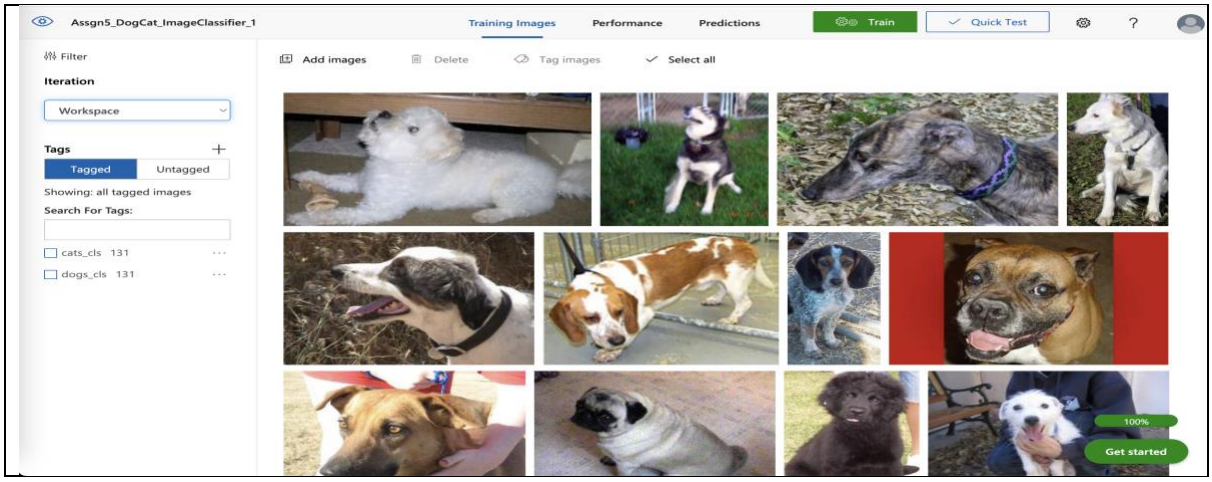
```
In [16]: dir_walk("train_batch_3")
```

Upload Batch 4

```
In [17]: dir_walk("train_batch_4")
```

Upload Batch 5

```
In [18]: dir_walk("train_batch_5")
```

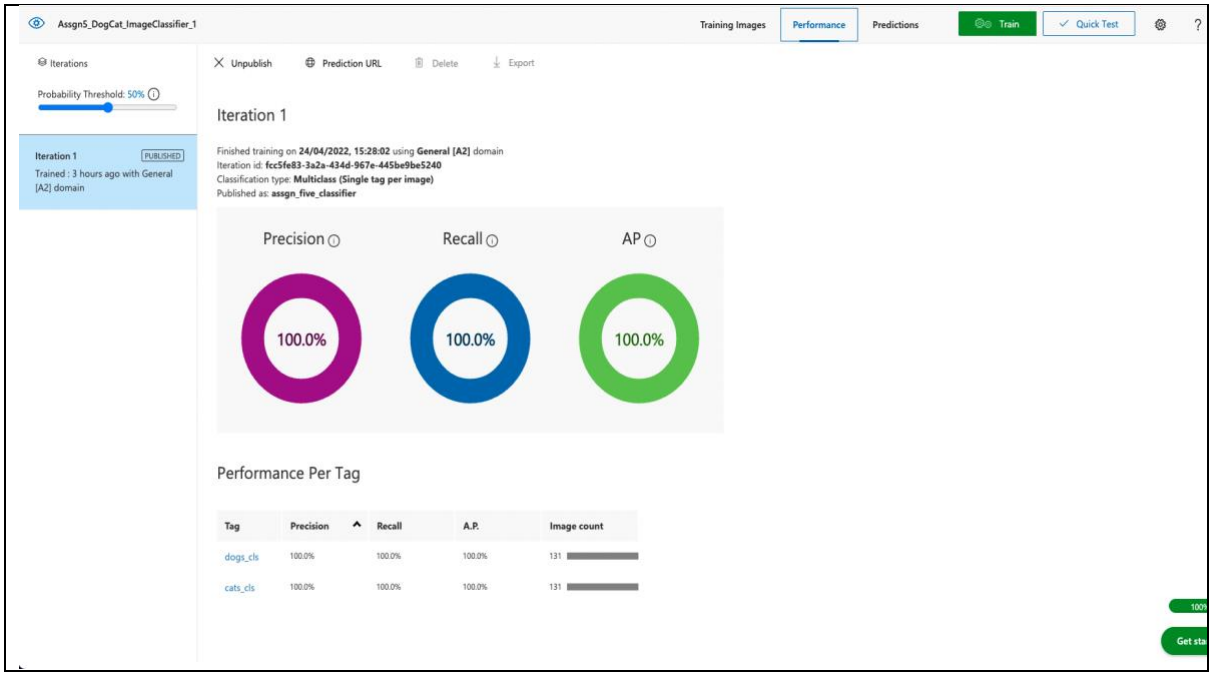


Step 13: Training the model.

Training the model using the uploaded dataset

```
In [19]: print ("Training the model...")
         iteration = model_train.train_project(project.id)
         time.sleep(30)
         while iteration.status != "Completed":
             iteration = model_train.get_iteration(project.id, iteration.id)
             print("Training status: " + iteration.status)
             print("Waiting 10 seconds...")
             time.sleep(10)
```

```
Training the model...
Training status: Training
Waiting 10 seconds...
Training status: Training
Waiting 10 seconds...
Training status: Training
Waiting 10 seconds...
Training status: Training
Waiting 10 seconds...
Training status: Training
Waiting 10 seconds...
Training status: Training
Waiting 10 seconds...
Training status: Training
Waiting 10 seconds...
Training status: Training
Waiting 10 seconds...
Training status: Training
Waiting 10 seconds...
Training status: Training
Waiting 10 seconds...
Training status: Training
Waiting 10 seconds...
Training status: Training
Waiting 10 seconds...
```



Step 14: Publishing the iteration to make the model available for testing and predictions

Publishing the iteration for Prediction

In [20]:
```
publish_iteration_name = "cat_dog_identifier_model"
```

In [21]:
```
projects = model_train.get_projects()
for project in projects:
    if project.name == project_name:
        project_id = project.id

if project_id is None:
    raise ValueError("Project does not exist")

iterations = model_train.get_iterations(project_id=project_id)
for iteration in iterations:
    print(iteration)
model_train.publish_iteration(project_id, iterations[0].id, publish_iteration_name, prediction_resource_id)
print("Successfully completed!")
```

```
{'additional_properties': {}, 'id': 'fcc5fe83-3a2a-434d-967e-445be9be5240', 'name': 'Iteration 1', 'status': 'Complet
ed', 'created': datetime.datetime(2022, 4, 24, 5, 20, 43, 350000, tzinfo=<isodate.tzinfo.Utc object at 0x103c16bb0>),
'last_modified': datetime.datetime(2022, 4, 24, 5, 28, 2, 674000, tzinfo=<isodate.tzinfo.Utc object at 0x103c16bb0>),
'trained_at': datetime.datetime(2022, 4, 24, 5, 28, 2, 637000, tzinfo=<isodate.tzinfo.Utc object at 0x103c16bb0>), 'p
roject_id': '5a568323-a3da-43d0-95a4-36ff919464c3', 'exportable': False, 'exportable_to': None, 'domain_id': '2e37d7f
b-3a54-486a-b4d6-cfc369af0018', 'classification_type': 'Multiclass', 'training_type': 'Regular', 'reserved_budget_in_
hours': 0, 'training_time_in_minutes': 1, 'publish_name': None, 'original_publish_resource_id': None, 'custom_base_mo
del_info': None, 'training_error_details': None}
Successfully completed!
```

Step 15: Generating the predictions on the test dataset.

1. I have instantiated **CustomVisionPredictionClient** by passing the **APIKeyCredentials** and **custom_vision_endpoint** objects.
2. Uploaded the test dataset, this time the dataset is unlabelled
3. Called **classify_image(project.id, publish_iteration_name, contents.read())** and passed project ID, iteration name and file content for generating the prediction

Creating an Prediction Client and passing the Prediction key in the header of the Prediction credentials

In [22]:
```
prediction_credentials = ApiKeyCredentials(in_headers={"Prediction-key": custom_vision_subscription_key})
predictor = CustomVisionPredictionClient(custom_vision_endpoint, prediction_credentials)
```

In [24]:
```
def get_prediction(predictions):
    return max(predictions.items(), key=operator.itemgetter(1))[0]
```

In [25]:
```
def get_confidence(predictions):
    return max(predictions.items(), key=operator.itemgetter(1))[1] * 100
```

In [32]:
```
def predict_class(root_path, image_name):
    predictions = {}
    with open(os.path.join(root_path, image_name), "rb") as contents:
        results = predictor.classify_image(project.id, publish_iteration_name, contents.read())

        for prediction in results.predictions:
            predictions[prediction.tag_name] = prediction.probability
    return predictions
```

Uploading test files and generating Predictions

In [33]:
```
for root_path, _, image_files in os.walk("test_set", topdown=False):
    for image_name in image_files:
        if image_name != ".DS_Store":
            predictions = predict_class(root_path, image_name)
            prediction = get_prediction(predictions)
            confidence = get_confidence(predictions)
            print("Prediction:", prediction, "Truth:", image_name, "Confidence", confidence, "%")
```

```
Prediction: dogs_cls Truth: dog.4005.jpg Confidence 98.24822 %
Prediction: dogs_cls Truth: dog.4010.jpg Confidence 99.998283 %
Prediction: dogs_cls Truth: dog.4004.jpg Confidence 99.94599 %
Prediction: cats_cls Truth: cat.4008.jpg Confidence 99.52466 %
Prediction: dogs_cls Truth: dog.4006.jpg Confidence 99.2004 %
Prediction: dogs_cls Truth: dog.4007.jpg Confidence 99.96558399999999 %
Prediction: cats_cls Truth: cat.4009.jpg Confidence 99.75140999999999 %
Prediction: dogs_cls Truth: dog.4003.jpg Confidence 99.98579 %
Prediction: dogs_cls Truth: dog.4002.jpg Confidence 99.84226000000001 %
Prediction: dogs_cls Truth: dog.4001.jpg Confidence 99.72695999999999 %
Prediction: cats_cls Truth: cat.4002.jpg Confidence 99.89398 %
Prediction: cats_cls Truth: cat.4003.jpg Confidence 99.996614 %
Prediction: cats_cls Truth: cat.4001.jpg Confidence 99.99307400000001 %
Prediction: cats_cls Truth: cat.4010.jpg Confidence 99.98053999999999 %
Prediction: cats_cls Truth: cat.4004.jpg Confidence 99.990785 %
Prediction: cats_cls Truth: cat.4005.jpg Confidence 99.99943999999999 %
Prediction: cats_cls Truth: cat.4007.jpg Confidence 99.999464 %
Prediction: dogs_cls Truth: dog.4009.jpg Confidence 99.95492 %
Prediction: dogs_cls Truth: dog.4008.jpg Confidence 99.999607 %
Prediction: cats_cls Truth: cat.4006.jpg Confidence 99.886644 %
```

Assignment_5_2_D

April 24, 2022

0.0.1 SIT788 - Engineering AI Solutions -Task5.2D - Custom Classifier

0.0.2 Installing Azure Custom Computer Vision SDK and other dependencies

[1]:
```
!pip install --upgrade azure-cognitiveservices-vision-customvision --quiet
```

[2]:
```
!pip install msrest==0.6.21 --quiet
```

0.0.3 Importing all the required Modules

[3]:
```
import os
from azure.cognitiveservices.vision.customvision.training import␣
 ↪CustomVisionTrainingClient
from azure.cognitiveservices.vision.customvision.training.models import␣
 ↪ImageFileCreateBatch, ImageFileCreateEntry
from msrest.authentication import ApiKeyCredentials
from msrest.authentication import CognitiveServicesCredentials
from azure.cognitiveservices.vision.customvision.prediction import␣
 ↪CustomVisionPredictionClient
import numpy as np
from dotenv import load_dotenv
import operator
import os
import time
%matplotlib inline
```

0.0.4 The Custom Vision API key and the endpoint information are loaded in an .env file in my development environment. Using loadenv from python dotenv module to load the key and endpoint into the memory

[4]:
```
load_dotenv()
```

[4]: True

[5]:
```
custom_vision_subscription_key = os.environ["CUSTOM_VISION_SUBSCRIPTION_KEY"]
custom_vision_endpoint = os.environ["CUSTOM_VISION_ENDPOINT"]
prediction_resource_id = os.environ["CUSTOM_VISION_RESOURCE_ID"]
```

0.0.5 Creating the APIKeyCredentials object by passing 'custom_vision_subscription_key' in the header

[6]:
```
creds = ApiKeyCredentials(in_headers={"Training-key":␣
↪custom_vision_subscription_key})
```

0.0.6 Creating the CustomVisionTrainingClient and passing the Enpoint and the APIKeyCredential object

[7]:
```
model_train = CustomVisionTrainingClient(custom_vision_endpoint, creds)
```

0.0.7 Getting all the available domains

[8]:
```
domains = model_train.get_domains()
for domain in domains:
    print(domain.name)
```

```
General [A2]
General [A1]
General
Food
Landmarks
Retail
Adult
General (compact) [S1]
General (compact)
Food (compact)
Landmarks (compact)
Retail (compact)
General [A1]
General
Logo
Products on Shelves
General (compact) [S1]
General (compact)
```

0.0.8 Storing the domain id of General [A2] domain

[9]:
```
for domain in domains:
    if domain.name == 'General [A2]':
        domain_id = domain.id
        print(domain.id)
```

```
2e37d7fb-3a54-486a-b4d6-cfc369af0018
```

0.0.9 Creating a project in the General [A2] domain

[10]:
```
def create_project(domain_id, project_name):
    return model_train.create_project(name=project_name, domain_id=domain_id,␣
↪classification_type="Multiclass")
```

0.0.10 Creating Project with domain_id and project_name

[11]:
```
project_name = f"Assgn5_DogCat_ImageClassifier_1"
project = create_project(domain_id, project_name)
print("Project created", project.id)
```

```
Project created 5a568323-a3da-43d0-95a4-36ff919464c3
```

0.0.11 Creating tags for the categories to label the images

[12]:
```
cat_tag = model_train.create_tag(project.id, "cats_cls")
dog_tag = model_train.create_tag(project.id, "dogs_cls")
```

0.0.12 Uploading the training dataset into the customvision.ai platform

[13]:
```
def dir_walk(batch_name):
    tag_id = None
    for root, dirs, files in os.walk(batch_name, topdown=False):
        try:
            root_dir, class_name = root.split('/')
            training_image_list = []
            for image in files:
                if image != ".DS_Store":
                    with open(os.path.join(root_dir, class_name, image), "rb")␣
↪as contents:
                        if class_name == "cats_cls":
                            tag_id = cat_tag.id
                        elif class_name == "dogs_cls":
                            tag_id = dog_tag.id

                        entry = ImageFileCreateEntry(name=image,␣
↪contents=contents.read(), tag_ids=[tag_id])
                        training_image_list.append(entry)

            upload_status = model_train.create_images_from_files(project.id,␣
↪ImageFileCreateBatch(images=training_image_list))
            if not upload_status.is_batch_successful:
                for image in upload_status.images:
                    if image.status != "OK":
                        print(image)
                    print("Image status: ", image.status)
```

```
        except ValueError:
            pass
```

0.0.13 Upload Batch 1

[14]:
```
dir_walk("train_batch_1")
```

0.0.14 Upload Batch 2

[15]:
```
dir_walk("train_batch_2")
```

0.0.15 Upload Batch 3

[16]:
```
dir_walk("train_batch_3")
```

0.0.16 Upload Batch 4

[17]:
```
dir_walk("train_batch_4")
```

0.0.17 Upload Batch 5

[18]:
```
dir_walk("train_batch_5")
```

0.0.18 Training the model using the uploaded dataset

[19]:
```
print ("Training the model...")
iteration = model_train.train_project(project.id)
time.sleep(30)
while iteration.status != "Completed":
    iteration = model_train.get_iteration(project.id, iteration.id)
    print("Training status: " + iteration.status)
    print("Waiting 10 seconds...")
    time.sleep(10)
```

```
Training the model…
Training status: Training
Waiting 10 seconds…
Training status: Training
Waiting 10 seconds…
Training status: Training
Waiting 10 seconds…
Training status: Training
Waiting 10 seconds…
Training status: Training
```

```
Waiting 10 seconds…
Training status: Training
Waiting 10 seconds…
Training status: Training
Waiting 10 seconds…
Training status: Training
Waiting 10 seconds…
Training status: Training
Waiting 10 seconds…
Training status: Training
Waiting 10 seconds…
Training status: Training
Waiting 10 seconds…
Training status: Training
Waiting 10 seconds…
Training status: Training
Waiting 10 seconds…
Training status: Training
Waiting 10 seconds…
Training status: Training
Waiting 10 seconds…
Training status: Training
Waiting 10 seconds…
Training status: Training
Waiting 10 seconds…
Training status: Training
Waiting 10 seconds…
Training status: Training
Waiting 10 seconds…
Training status: Training
Waiting 10 seconds…
Training status: Training
Waiting 10 seconds…
Training status: Training
Waiting 10 seconds…
Training status: Training
Waiting 10 seconds…
Training status: Training
Waiting 10 seconds…
Training status: Training
Waiting 10 seconds…
Training status: Training
Waiting 10 seconds…
Training status: Training
Waiting 10 seconds…
Training status: Training
Waiting 10 seconds…
Training status: Training
```

```
Waiting 10 seconds…
Training status: Training
Waiting 10 seconds…
Training status: Training
Waiting 10 seconds…
Training status: Training
Waiting 10 seconds…
Training status: Training
Waiting 10 seconds…
Training status: Training
Waiting 10 seconds…
Training status: Training
Waiting 10 seconds…
Training status: Training
Waiting 10 seconds…
Training status: Training
Waiting 10 seconds…
Training status: Training
Waiting 10 seconds…
Training status: Completed
Waiting 10 seconds…
```

0.0.19 Publishing the iteration for Prediction

[20]:
```
publish_iteration_name = "cat_dog_identifier_model"
```

[21]:
```
projects = model_train.get_projects()
for project in projects:
    if project.name == project_name:
        project_id = project.id

if project_id is None:
    raise ValueError("Project does not exist")

iterations = model_train.get_iterations(project_id=project_id)
for iteration in iterations:
    print(iteration)
model_train.publish_iteration(project_id, iterations[0].id,␣
 ↪publish_iteration_name, prediction_resource_id)
print("Successfully completed!")
```

```
{'additional_properties': {}, 'id': 'fcc5fe83-3a2a-434d-967e-445be9be5240',
'name': 'Iteration 1', 'status': 'Completed', 'created': datetime.datetime(2022,
4, 24, 5, 20, 43, 350000, tzinfo=<isodate.tzinfo.Utc object at 0x103c16bb0>),
'last_modified': datetime.datetime(2022, 4, 24, 5, 28, 2, 674000,
tzinfo=<isodate.tzinfo.Utc object at 0x103c16bb0>), 'trained_at':
datetime.datetime(2022, 4, 24, 5, 28, 2, 637000, tzinfo=<isodate.tzinfo.Utc
object at 0x103c16bb0>), 'project_id': '5a568323-a3da-43d0-95a4-36ff919464c3',
'exportable': False, 'exportable_to': None, 'domain_id':
'2e37d7fb-3a54-486a-b4d6-cfc369af0018', 'classification_type': 'Multiclass',
```

```
'training_type': 'Regular', 'reserved_budget_in_hours': 0,
'training_time_in_minutes': 1, 'publish_name': None,
'original_publish_resource_id': None, 'custom_base_model_info': None,
'training_error_details': None}
Successfully completed!
```

0.0.20 Creating an Prediction Client and passing the Prediction key in the header of the Prediction credentials

[22]:
```
prediction_credentials = ApiKeyCredentials(in_headers={"Prediction-key":␣
↪custom_vision_subscription_key})
predictor = CustomVisionPredictionClient(custom_vision_endpoint,␣
↪prediction_credentials)
```

[24]:
```
def get_prediction(predictions):
    return max(predictions.items(), key=operator.itemgetter(1))[0]
```

[25]:
```
def get_confidence(predictions):
    return max(predictions.items(), key=operator.itemgetter(1))[1] * 100
```

[32]:
```
def predict_class(root_path, image_name):
    predictions = {}
    with open(os.path.join(root_path, image_name), "rb") as contents:
        results = predictor.classify_image(project.id, publish_iteration_name,␣
↪contents.read())

        for prediction in results.predictions:
            predictions[prediction.tag_name] = prediction.probability
    return predictions
```

0.0.21 Uploading test files and generating Predictions

[33]:
```
for root_path, _, image_files in os.walk("test_set", topdown=False):
    for image_name in image_files:
        if image_name != ".DS_Store":
            predictions = predict_class(root_path, image_name)
            prediction = get_prediction(predictions)
            confidence = get_confidence(predictions)
            print("Prediction:", prediction, "Truth:", image_name,␣
↪"Confidence", confidence, "%")
```

```
Prediction: dogs_cls Truth: dog.4005.jpg Confidence 98.24822 %
Prediction: dogs_cls Truth: dog.4010.jpg Confidence 99.998283 %
Prediction: dogs_cls Truth: dog.4004.jpg Confidence 99.94599 %
Prediction: cats_cls Truth: cat.4008.jpg Confidence 99.52466 %
Prediction: dogs_cls Truth: dog.4006.jpg Confidence 99.2004 %
Prediction: dogs_cls Truth: dog.4007.jpg Confidence 99.96558399999999 %
```

```
Prediction: cats_cls Truth: cat.4009.jpg Confidence 99.75140999999999 %
Prediction: dogs_cls Truth: dog.4003.jpg Confidence 99.98579 %
Prediction: dogs_cls Truth: dog.4002.jpg Confidence 99.84226000000001 %
Prediction: dogs_cls Truth: dog.4001.jpg Confidence 99.72695999999999 %
Prediction: cats_cls Truth: cat.4002.jpg Confidence 99.89398 %
Prediction: cats_cls Truth: cat.4003.jpg Confidence 99.996614 %
Prediction: cats_cls Truth: cat.4001.jpg Confidence 99.99307400000001 %
Prediction: cats_cls Truth: cat.4010.jpg Confidence 99.98053999999999 %
Prediction: cats_cls Truth: cat.4004.jpg Confidence 99.990785 %
Prediction: cats_cls Truth: cat.4005.jpg Confidence 99.99943999999999 %
Prediction: cats_cls Truth: cat.4007.jpg Confidence 99.999464 %
Prediction: dogs_cls Truth: dog.4009.jpg Confidence 99.95492 %
Prediction: dogs_cls Truth: dog.4008.jpg Confidence 99.999607 %
Prediction: cats_cls Truth: cat.4006.jpg Confidence 99.886644 %
```

[ ]: