Task 5.2: Computer vision and custom vision

Answer 1

The Near Real-time Face detection has been achieved using

Prerequisite – Must have a valid Key and Endpoint for the Face API

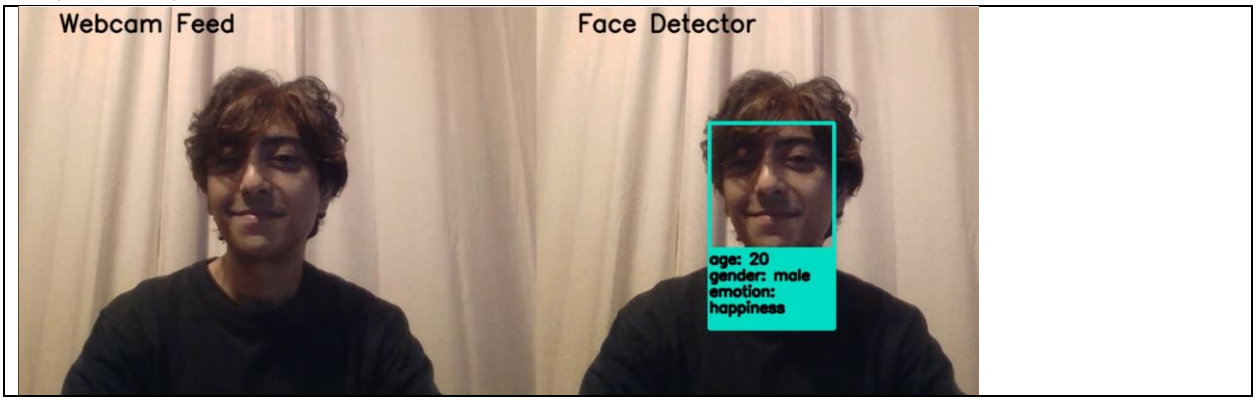
Steps taken to create a Near Real-time Face detection

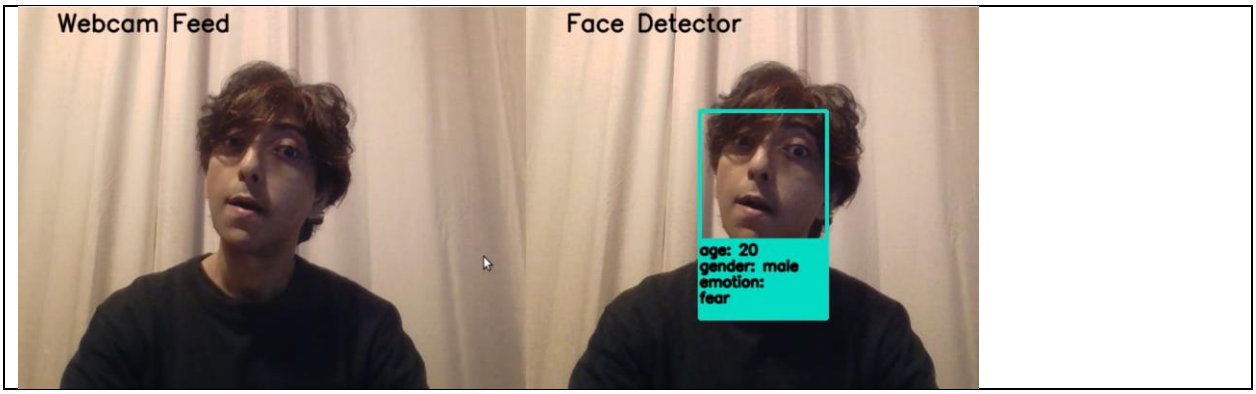
1. Get the Frames from the Webcam and display the frames in the left side window
2. Send the Frames with a delay of 1 second to Face detection API to return the Face rectangle and the Face attributes.
3. Draw the Bounding Box on the Face and the Text attributes on the right side window.

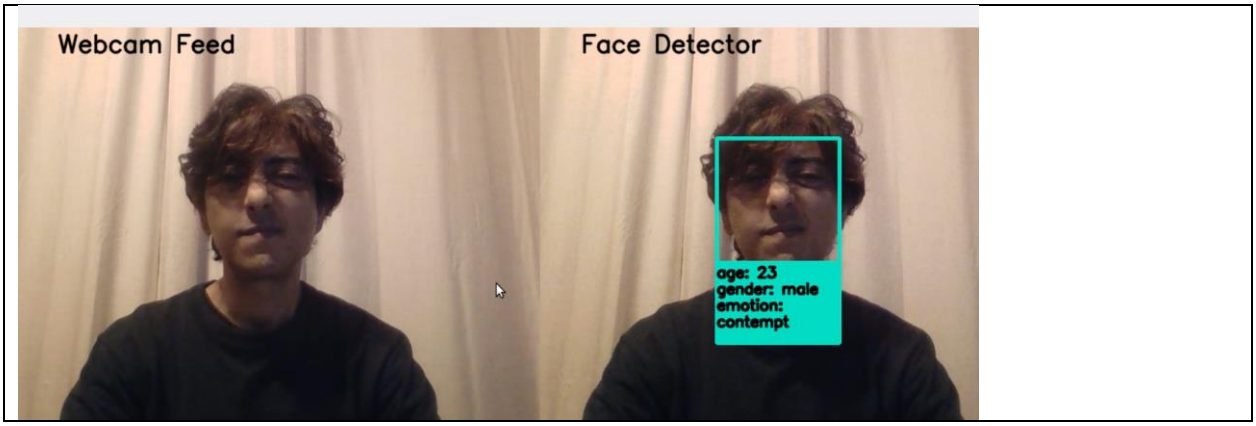
Code Pattern

The code is designed on the lines of a Publisher Consumer model.

Output Samples







Code Explanation

Imported the essential libraries

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

Created a derived class of Thread type, used OpenCV's APIs to capture video stream and show in 2 frames side-by-side

```
class NearRealtimeFaceDetector(threading.Thread):
    """
    NearRealtimeFaceDetector is a derived class from base class Thread.
    It detects Face in near real-time (after 1 second) using a video feed from webcam
    """

    def __init__(self):
        threading.Thread.__init__(self)
        self.video_feed = cv2.VideoCapture(0)
        # left frame to show the webcam feed
        self.frame_1 = None
        # right frame to show the face detector output
        self.frame_2 = None

    def run(self):
        _, self.frame_1 = self.video_feed.read()
        self.frame_2 = cv2.flip(self.frame_1.copy(), 1)

        while True:
            _, frame = self.video_feed.read()
            frame = cv2.flip(frame, 1)
            # create a copy of the flipped frame
            self.frame_1 = frame.copy()
            # Left frame header to indicate that this is the Webcam Feed
            frame = cv2.putText(frame, "Webcam Feed", (50, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 0), 2, cv2.LINE_AA)
            # Create a side-by-side display
            frame_full = cv2.hconcat([frame, self.frame_2])
            # Display the feeds in a window
            cv2.imshow("Near Real-time Face detection", frame_full)
            # 1 millisecond delay to allow the window to load
            cv2.waitKey(1)
            # if the window is closed then break out of the infinite loop
            if cv2.getWindowProperty("Near Real-time Face detection", cv2.WND_PROP_VISIBLE) < 1:
                break
        cv2.destroyAllWindows()
```

Created the Face detector method that works with a lag of 1 second (near real-time)

```python
def detector(self):
    face_attributes = ['emotion', 'age', 'gender']
    while True:
        # Introduce a delay of 1 second
        time.sleep(1)
        # Create a copy of the frame
        frame = self.frame_1.copy()
        # Save the frame as an image
        cv2.imwrite('temp.jpg', frame)
        # Load the saved image to be streamed to face detection API
        local_image = open('temp.jpg', "rb")
        # Pass the image to the detector webservice
        faces = face_client.face.detect_with_stream(local_image, return_face_attributes=face_attributes, detection_model='detection_01')
        face_found = len(faces) > 0
        # check if any face has been detected
        if face_found:
            frame = cv2.putText(frame, "Face Detector", (50, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 0), 2, cv2.LINE_AA)
            single_face = faces[0]

            # Extract face attributes of the first face found
            age_label_value = self.age_label_text(single_face.face_attributes.age)
            gender_label_value = self.gender_label_text(single_face.face_attributes.gender)

            # Retrieve the most prevalent emotion from the face attributes
            prevalent_emotion = self.get_prevalent_emotion(faces[0].face_attributes.emotion)

            # Get the coordinates of the rectangular bounding box
            left, top, right, bottom = self.get_rectangle_coordinates(faces[0])
            frame = self.draw_bounding_box_with_annotation_labels(frame, left, top, right, bottom, age_label_value, gender_label_value, prevalent_emotion)

            # Refresh the frame_2 with the output
            self.frame_2 = frame
```

Created functions for creating labels of age, gender, and emotion

```python
# Returns the age label text
def age_label_text(self, age):
    return "age: " + str(int(age))

# Returns the gender label text
def gender_label_text(self, gender):
    gender = (gender.split('.'))[0]
    return "gender: " + str(gender)

# Recognize the most prevalent emotion and return the name of the emotion
# that the face detector has returned with most confidence
def get_prevalent_emotion(self, emotion_details):
    emotions = emotion_details.__dict__
    emotion_keys = list(emotions.keys())[1:8]
    emotion_values = list(emotions.values())[1:8]
    highest_emotion_value = max(emotion_values)
    highest_emotion_index = emotion_values.index(highest_emotion_value)
    highest_emotion = emotion_keys[highest_emotion_index]
    return highest_emotion
```

Created functions for creating bounding box with annotation

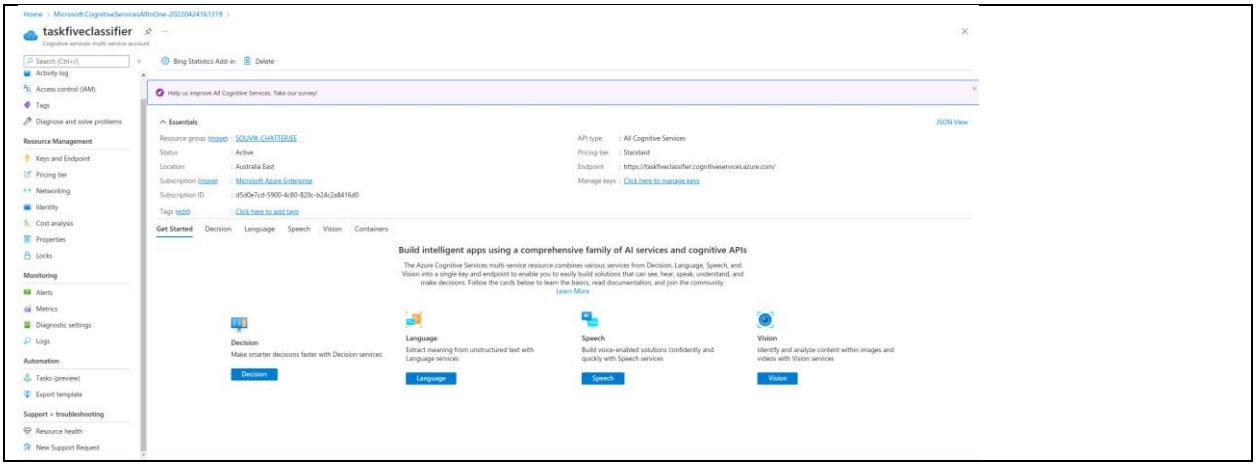
```python
# Draws the Bounding Box and Creates Annotation labels under the Bounding Box to display the Face Attributes
def draw_bounding_box_with_annotation_labels(self, frame, left, top, right, bottom, age, gender, emotion):
    # Draw the bounding box
    border_color = (198, 255, 0)
    frame = cv2.rectangle(frame, (left, top), (right, bottom + 100), border_color, 3)

    # Draw a filled rectangle for showing the output as Annotation labels
    frame = cv2.rectangle(frame, (left, bottom), (right, bottom + 100), border_color, cv2.FILLED)

    label_top = 20
    # Show the results on screen as labels under the bounding box
    frame = cv2.putText(frame, age, (left, bottom + label_top), cv2.FONT_HERSHEY_SIMPLEX, 0.6,
                        (0, 0, 0), 2, cv2.LINE_AA)
    label_top += 20
    frame = cv2.putText(frame, gender, (left, bottom + label_top), cv2.FONT_HERSHEY_SIMPLEX, 0.6,
                        (0, 0, 0), 2, cv2.LINE_AA)
    label_top += 20
    frame = cv2.putText(frame, "emotion: ", (left, bottom + label_top), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 0), 2,
                        cv2.LINE_AA)
    label_top += 20
    # emotion value is moved to the next line to allow enough space
    frame = cv2.putText(frame, emotion, (left, bottom + label_top), cv2.FONT_HERSHEY_SIMPLEX, 0.6,
                        (0, 0, 0), 2, cv2.LINE_AA)
    return frame
```

Answer 2

This task requires developing a custom classifier that can identify Cats and Dogs using the dataset (Source: https://www.kaggle.com/chetankv/dogs-cats-images). The following steps were taken to achieve this.

1. A Cognitive Service Resource needs to be created and it's Key and Endpoint need to be stored and used for authentication purpose.
2. The customvision.ai portal automatically identifies the user and the Resource. It provides the Custom Vision Resource Id.
3. Installed **azure-cognitiveservices-vision-customvision** CustomVision SDK and **msrest** for Authentication.
4. Imported all the required modules.
5. Created a **CustomVisionTrainingClient** using Endpoint and Key details and used it to create a **Project** using **create_project API** using **General [A2] domain** of classification type **Multiclass**.
6. Created **Tags** for Cats and Dogs classes.
7. Loaded data in batches of size less than **64**.
8. Trained the Model using **train_project API** and Published the Iteration using **publish_iteration API**.
9. Created a **CustomVisionPredictionClient** using Endpoint and Key details.
10. Tested the Model using **classify_image API**

Resource screenshot in Azure Portal



CustomVision Portal shows the Resource Id for the following Resource and automatically authenticates the Azure Portal account.

Resource in CustomVision Portal

Installed the customvision and msrest libraries and imported all the necessary modules

Install azure-cognitiveservices-vision-customvision and msrest

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

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

Import necessary libraries

```
In [3]: import os
        import operator
        import time
        from azure.cognitiveservices.vision.customvision.training import CustomVisionTrainingClient
        from azure.cognitiveservices.vision.customvision.training.models import ImageFileCreateBatch, ImageFileCreateEntry
        from msrest.authentication import ApiKeyCredentials, CognitiveServicesCredentials
        from azure.cognitiveservices.vision.customvision.prediction import CustomVisionPredictionClient
        from configparser import ConfigParser
```

Loaded Key, Endpoint, and Resource Id from config file

```
In [6]: config = ConfigParser()
        config.read('classifier_service_details.cfg')

Out[6]: ['classifier_service_details.cfg']

In [7]: key = config['infrastructure']['key']
        endpoint = config['infrastructure']['endpoint']
        resource_id = config['infrastructure']['resource_id']
```

Constructed CustomVisionTrainingClient with Enpoint and the APIKeyCredential and Fetched all domains to use the domain's id while creating project

Constructing CustomVisionTrainingClient with Enpoint and the APIKeyCredential

```
In [8]: cv_client = CustomVisionTrainingClient(endpoint, ApiKeyCredentials(in_headers={"Training-key": key}))
```

Fetch all domains to use the domain's id while creating project

```
In [10]: domains = cv_client.get_domains()
```

Since there is no dedicated domain that is appropriate, hence using General [A2]. So fetched all the domains and used the domain id for General [A2]

Since there is no dedicated domain that is appropriate, hence using General [A2]

In [11]:
```
domain_id = None
for domain in domains:
    if domain.name == 'General [A2]':
        domain_id = domain.id
```

In [12]:
```
print(domain_id)
```

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

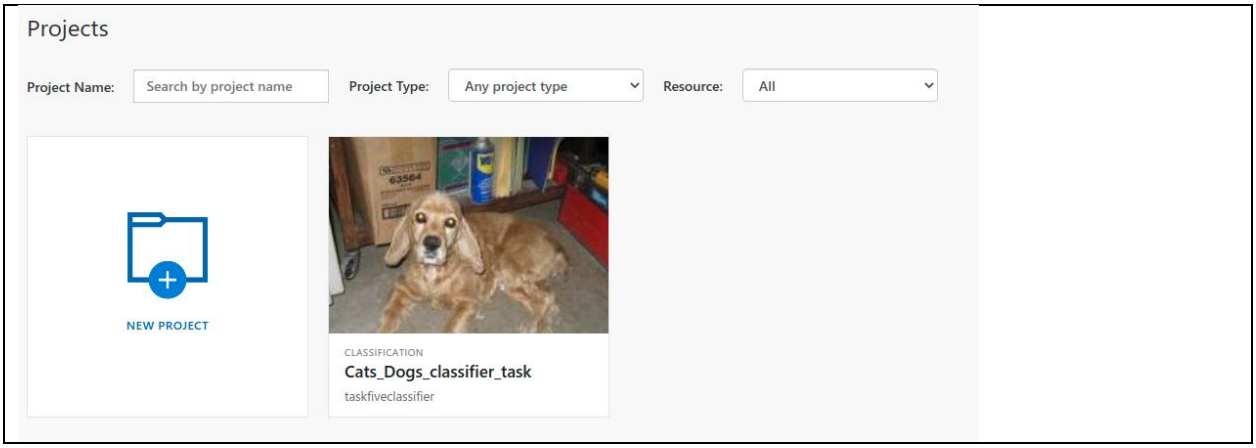
Created the Project of Classification Type “Multiclass” using create_project API

Creating Project

In [13]:
```
project_name = "Cats_Dogs_classifier_task"
project = cv_client.create_project(name=project_name, domain_id=domain_id, classification_type="Multiclass")
print("Id of the Project created", project.id)
```

```
Id of the Project created 8cbb8844-d84e-4447-932a-d094f8a84932
```

Screenshot of Project created in CustomVision Portal



Created Tags for Cat and Dog classes and loaded files in 4 batches

Creating tags for Cat and Dog classes for labeling the images

In [15]:
```
tag_c = cv_client.create_tag(project.id, "cats_tag")
tag_d = cv_client.create_tag(project.id, "dogs_tag")
```

In [16]:
```
def upload_images(img_list):
    upload_status = cv_client.create_images_from_files(project.id, ImageFileCreateBatch(images=img_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)
```

Loading training batch 1

In [17]:
```
for root, dirs, files in os.walk("trn_1", topdown=False):
    try:
        root_dir, category = root.split('/')
        trn_images = []
        for image in files:
            with open(os.path.join(root_dir, category, image), "rb") as img_content:
                if category == "cats_tag":
                    tag_id = tag_c.id
                elif category == "dogs_tag":
                    tag_id = tag_d.id
                img_entry = ImageFileCreateEntry(name=image, contents=img_content.read(), tag_ids=[tag_id])
                trn_images.append(img_entry)
        upload_images(trn_images)
    except ValueError:
        pass
```

Loading training batch 2

In [18]:
```
for root, dirs, files in os.walk("trn_2", topdown=False):
    try:
        root_dir, category = root.split('/')
        trn_images = []
        for image in files:
            with open(os.path.join(root_dir, category, image), "rb") as img_content:
                if category == "cats_tag":
                    tag_id = tag_c.id
                elif category == "dogs_tag":
                    tag_id = tag_d.id
                img_entry = ImageFileCreateEntry(name=image, contents=img_content.read(), tag_ids=[tag_id])
                trn_images.append(img_entry)
        upload_images(trn_images)
    except ValueError:
        pass
```

Loading training batch 3

In [19]:
```
for root, dirs, files in os.walk("trn_3", topdown=False):
    try:
        root_dir, category = root.split('/')
        trn_images = []
        for image in files:
            with open(os.path.join(root_dir, category, image), "rb") as img_content:
                if category == "cats_tag":
                    tag_id = tag_c.id
                elif category == "dogs_tag":
                    tag_id = tag_d.id
                img_entry = ImageFileCreateEntry(name=image, contents=img_content.read(), tag_ids=[tag_id])
                trn_images.append(img_entry)
        upload_images(trn_images)
    except ValueError:
        pass
```

Loading training batch 4

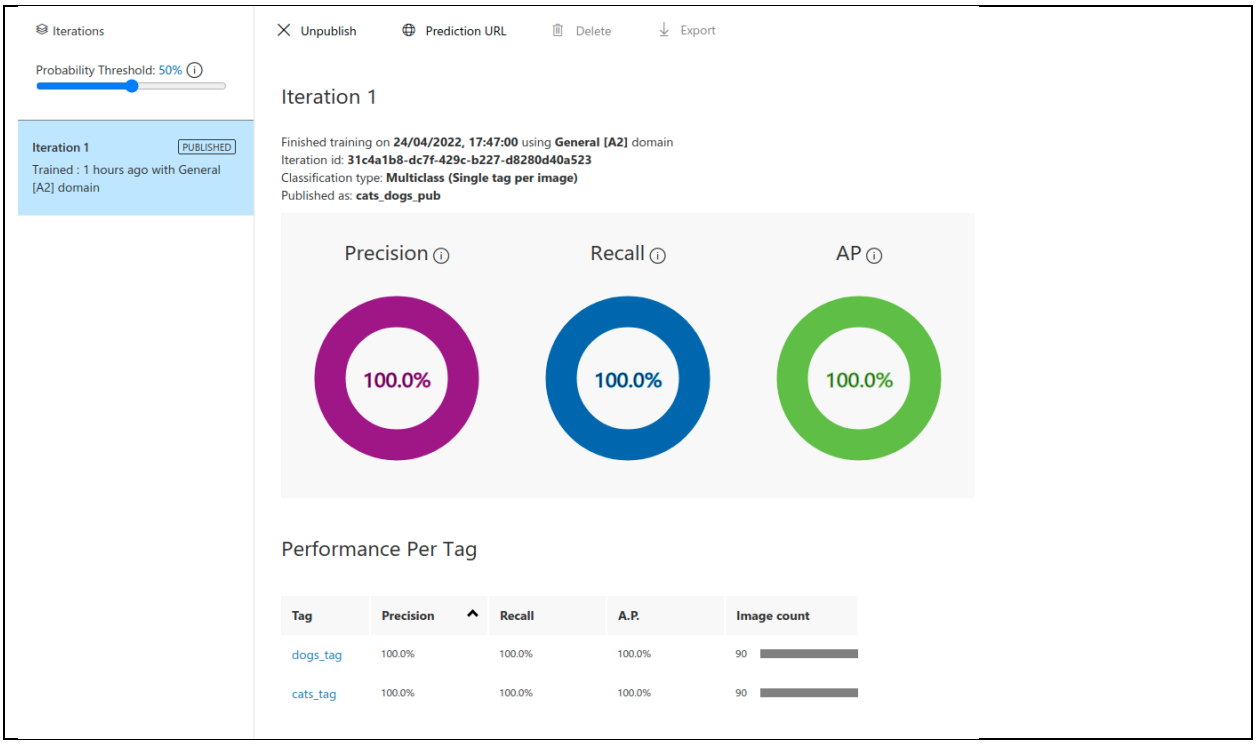
In [20]:
```
for root, dirs, files in os.walk("trn_4", topdown=False):
    try:
        root_dir, category = root.split('/')
        trn_images = []
        for image in files:
            with open(os.path.join(root_dir, category, image), "rb") as img_content:
                if category == "cats_tag":
                    tag_id = tag_c.id
                elif category == "dogs_tag":
                    tag_id = tag_d.id
                img_entry = ImageFileCreateEntry(name=image, contents=img_content.read(), tag_ids=[tag_id])
                trn_images.append(img_entry)
        upload_images(trn_images)
    except ValueError:
        pass
```

Trained the Model using train_project API

Training the model

In [21]:
```
iteration = cv_client.train_project(project.id)
# wait for the iteration
time.sleep(42)
while iteration.status != "Completed":
    iteration = cv_client.get_iteration(project.id, iteration.id)
    print("Status: " + iteration.status)
    time.sleep(10)
```

```
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Completed
```



Publish the Iteration using publish_iteration API

Before we start Testing the Model, the Iteration needs to be Published in CustomVision

```
In [22]: publish_iter = "cats_dogs_pub"

In [23]: iterations = cv_client.get_iterations(project_id=project.id)

In [31]: cv_client.publish_iteration(project.id, iterations[0].id, publish_iter, resource_id)
Out[31]: True

In [32]: iterations = cv_client.get_iterations(project_id=project.id)
         for iteration in iterations:
             print(iteration)
```

```
{'additional_properties': {}, 'id': '31c4a1b8-dc7f-429c-b227-d8280d40a523', 'name': 'Iteration 1', 'status': 'Comp
leted', 'created': datetime.datetime(2022, 4, 24, 6, 55, 48, 166000, tzinfo=<isodate.tzinfo.Utc object at 0x7f750a
e6ec40>), 'last_modified': datetime.datetime(2022, 4, 24, 7, 54, 5, 154000, tzinfo=<isodate.tzinfo.Utc object at 0
x7f750ae6ec40>), 'trained_at': datetime.datetime(2022, 4, 24, 7, 47, 0, 302000, tzinfo=<isodate.tzinfo.Utc object
at 0x7f750ae6ec40>), 'project_id': '8cbb8844-d84e-4447-932a-d094f8a84932', 'exportable': False, 'exportable_to': N
one, 'domain_id': '2e37d7fb-3a54-486a-b4d6-cfc369af0018', 'classification_type': 'Multiclass', 'training_type': 'R
egular', 'reserved_budget_in_hours': 0, 'training_time_in_minutes': 1, 'publish_name': 'cats_dogs_pub', 'original_
publish_resource_id': '/subscriptions/d5d0e7cd-5900-4c80-820c-b24c2a8416d0/resourceGroups/SOUVIK-CHATTERJEE/provid
ers/Microsoft.CognitiveServices/accounts/taskfiveclassifier', 'custom_base_model_info': None, 'training_error_deta
ils': None}
```

Created the Prediction Client using CustomVisionPredictionClient API

Create Prediction Client

```
In [33]: predictor = CustomVisionPredictionClient(endpoint, ApiKeyCredentials(in_headers={"Prediction-key": key}))
```

Tested the Model with data from testing_set using classify_image API

Test the Model

In [36]:
```
def get_predicted_class(root_path, img):
    preds = {}
    with open(os.path.join(root_path, img), "rb") as img_contents:
        pred_results = predictor.classify_image(project.id, publish_iter, img_contents.read())

        for pred in pred_results.predictions:
            preds[pred.tag_name] = pred.probability
    return preds
```

In [38]:
```
for root_path, _, img_files in os.walk("tst_files", topdown=False):
    for img_name in img_files:
        preds = get_predicted_class(root_path, img_name)
        pred_items = preds.items()
        pred = max(pred_items, key=operator.itemgetter(1))[0]
        conf = max(pred_items, key=operator.itemgetter(1))[1]
        conf_percentage = (conf * 100)
        percent_str = str(conf_percentage) + "%"
        print("Ground Truth:", img_name, "Prediction:", pred, "Confidence", percent_str, "%")
```

```
Ground Truth: cat.94.jpg Prediction: cats_tag Confidence 99.95722%
Ground Truth: cat.95.jpg Prediction: cats_tag Confidence 99.933004%
Ground Truth: dog.95.jpg Prediction: dogs_tag Confidence 99.57066%
Ground Truth: dog.98.jpg Prediction: dogs_tag Confidence 99.897003%
Ground Truth: dog.97.jpg Prediction: dogs_tag Confidence 99.996746%
Ground Truth: dog.91.jpg Prediction: dogs_tag Confidence 99.3909%
Ground Truth: cat.91.jpg Prediction: cats_tag Confidence 99.82772%
Ground Truth: dog.92.jpg Prediction: dogs_tag Confidence 99.316627%
Ground Truth: dog.99.jpg Prediction: dogs_tag Confidence 99.98796%
Ground Truth: dog.94.jpg Prediction: dogs_tag Confidence 99.65952999999999%
Ground Truth: cat.98.jpg Prediction: cats_tag Confidence 99.9577%
Ground Truth: dog.96.jpg Prediction: dogs_tag Confidence 99.60526%
Ground Truth: dog.93.jpg Prediction: dogs_tag Confidence 99.98519399999999%
Ground Truth: cat.99.jpg Prediction: cats_tag Confidence 99.988747%
Ground Truth: dog.100.jpg Prediction: dogs_tag Confidence 99.987555%
Ground Truth: cat.100.jpg Prediction: cats_tag Confidence 99.957865%
Ground Truth: cat.96.jpg Prediction: cats_tag Confidence 99.95153%
Ground Truth: cat.97.jpg Prediction: cats_tag Confidence 99.95941499999999%
Ground Truth: cat.93.jpg Prediction: cats_tag Confidence 99.988055%
Ground Truth: cat.92.jpg Prediction: cats_tag Confidence 99.05807399999999%
```

Task_5_2_D_221382131

April 24, 2022

0.0.1 Install azure-cognitiveservices-vision-customvision and msrest

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

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

0.0.2 Import necessary libraries

[3]:
```
import os
import operator
import time
from azure.cognitiveservices.vision.customvision.training import␣
 ↪CustomVisionTrainingClient
from azure.cognitiveservices.vision.customvision.training.models import␣
 ↪ImageFileCreateBatch, ImageFileCreateEntry
from msrest.authentication import ApiKeyCredentials,␣
 ↪CognitiveServicesCredentials
from azure.cognitiveservices.vision.customvision.prediction import␣
 ↪CustomVisionPredictionClient
from configparser import ConfigParser
```

[6]:
```
config = ConfigParser()
config.read('classifier_service_details.cfg')
```

[6]:
```
['classifier_service_details.cfg']
```

[7]:
```
key = config['infrastructure']['key']
endpoint = config['infrastructure']['endpoint']
resource_id = config['infrastructure']['resource_id']
```

0.0.3 Constructing CustomVisionTrainingClient with Enpoint and the APIKeyCredential

[8]:
```
cv_client = CustomVisionTrainingClient(endpoint,␣
↪ApiKeyCredentials(in_headers={"Training-key": key}))
```

0.0.4 Fetch all domains to use the domain's id while creating project

[10]:
```
domains = cv_client.get_domains()
```

0.0.5 Since there is no dedicated domain that is appropriate, hence using General [A2]

[11]:
```
domain_id = None
for domain in domains:
    if domain.name == 'General [A2]':
        domain_id = domain.id
```

[12]:
```
print(domain_id)
```

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

0.0.6 Creating Project

[13]:
```
project_name = "Cats_Dogs_classifier_task"
project = cv_client.create_project(name=project_name, domain_id=domain_id,␣
↪classification_type="Multiclass")
print("Id of the Project created", project.id)
```

```
Id of the Project created 8cbb8844-d84e-4447-932a-d094f8a84932
```

0.0.7 Creating tags for Cat and Dog classes for labeling the images

[15]:
```
tag_c = cv_client.create_tag(project.id, "cats_tag")
tag_d = cv_client.create_tag(project.id, "dogs_tag")
```

[16]:
```
def upload_images(img_list):
    upload_status = cv_client.create_images_from_files(project.id,␣
↪ImageFileCreateBatch(images=img_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)
```

0.0.8 Loading training batch 1

[17]:
```
for root, dirs, files in os.walk("trn_1", topdown=False):
    try:
        root_dir, category = root.split('/')
        trn_images = []
        for image in files:
            with open(os.path.join(root_dir, category, image), "rb") as␣
↪img_content:
                if category == "cats_tag":
                    tag_id = tag_c.id
                elif category == "dogs_tag":
                    tag_id = tag_d.id
                img_entry = ImageFileCreateEntry(name=image,␣
↪contents=img_content.read(), tag_ids=[tag_id])
                trn_images.append(img_entry)
        upload_images(trn_images)
    except ValueError:
        pass
```

0.0.9 Loading training batch 2

[18]:
```
for root, dirs, files in os.walk("trn_2", topdown=False):
    try:
        root_dir, category = root.split('/')
        trn_images = []
        for image in files:
            with open(os.path.join(root_dir, category, image), "rb") as␣
↪img_content:
                if category == "cats_tag":
                    tag_id = tag_c.id
                elif category == "dogs_tag":
                    tag_id = tag_d.id
                img_entry = ImageFileCreateEntry(name=image,␣
↪contents=img_content.read(), tag_ids=[tag_id])
                trn_images.append(img_entry)
        upload_images(trn_images)
    except ValueError:
        pass
```

0.0.10 Loading training batch 3

[19]:
```
for root, dirs, files in os.walk("trn_3", topdown=False):
    try:
        root_dir, category = root.split('/')
        trn_images = []
        for image in files:
            with open(os.path.join(root_dir, category, image), "rb") as␣
↪img_content:
                if category == "cats_tag":
                    tag_id = tag_c.id
                elif category == "dogs_tag":
                    tag_id = tag_d.id
                img_entry = ImageFileCreateEntry(name=image,␣
↪contents=img_content.read(), tag_ids=[tag_id])
                trn_images.append(img_entry)
        upload_images(trn_images)
    except ValueError:
        pass
```

0.0.11 Loading training batch 4

[20]:
```
for root, dirs, files in os.walk("trn_4", topdown=False):
    try:
        root_dir, category = root.split('/')
        trn_images = []
        for image in files:
            with open(os.path.join(root_dir, category, image), "rb") as␣
↪img_content:
                if category == "cats_tag":
                    tag_id = tag_c.id
                elif category == "dogs_tag":
                    tag_id = tag_d.id
                img_entry = ImageFileCreateEntry(name=image,␣
↪contents=img_content.read(), tag_ids=[tag_id])
                trn_images.append(img_entry)
        upload_images(trn_images)
    except ValueError:
        pass
```

0.0.12 Training the model

[21]:
```
iteration = cv_client.train_project(project.id)
# wait for the iteration
time.sleep(42)
while iteration.status != "Completed":
    iteration = cv_client.get_iteration(project.id, iteration.id)
    print("Status: " + iteration.status)
    time.sleep(10)
```

```
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Training
Status: Completed
```

0.0.13 Before we start Testing the Model, the Iteration needs to be Published in CustomVision

[22]:
```
publish_iter = "cats_dogs_pub"
```

[23]:
```
iterations = cv_client.get_iterations(project_id=project.id)
```

[31]:
```
cv_client.publish_iteration(project.id, iterations[0].id, publish_iter,␣
↪resource_id)
```

[31]:
```
True
```

[32]:
```
iterations = cv_client.get_iterations(project_id=project.id)
for iteration in iterations:
    print(iteration)
```

```
{'additional_properties': {}, 'id': '31c4a1b8-dc7f-429c-b227-d8280d40a523',
'name': 'Iteration 1', 'status': 'Completed', 'created': datetime.datetime(2022,
4, 24, 6, 55, 48, 166000, tzinfo=<isodate.tzinfo.Utc object at 0x7f750ae6ec40>),
'last_modified': datetime.datetime(2022, 4, 24, 7, 54, 5, 154000,
tzinfo=<isodate.tzinfo.Utc object at 0x7f750ae6ec40>), 'trained_at':
datetime.datetime(2022, 4, 24, 7, 47, 0, 302000, tzinfo=<isodate.tzinfo.Utc
object at 0x7f750ae6ec40>), 'project_id':
'8cbb8844-d84e-4447-932a-d094f8a84932', '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':
'cats_dogs_pub', 'original_publish_resource_id':
'/subscriptions/d5d0e7cd-5900-4c80-820c-b24c2a8416d0/resourceGroups/SOUVIK-
CHATTERJEE/providers/Microsoft.CognitiveServices/accounts/taskfiveclassifier',
'custom_base_model_info': None, 'training_error_details': None}
```

0.0.14 Create Prediction Client

[33]:
```
predictor = CustomVisionPredictionClient(endpoint,␣
↪ApiKeyCredentials(in_headers={"Prediction-key": key}))
```

0.0.15 Test the Model

[36]:
```
def get_predicted_class(root_path, img):
    preds = {}
    with open(os.path.join(root_path, img), "rb") as img_contents:
        pred_results = predictor.classify_image(project.id, publish_iter,␣
↪img_contents.read())

        for pred in pred_results.predictions:
            preds[pred.tag_name] = pred.probability
    return preds
```

[38]:
```
for root_path, _, img_files in os.walk("tst_files", topdown=False):
    for img_name in img_files:
        preds = get_predicted_class(root_path, img_name)
        pred_items = preds.items()
        pred = max(pred_items, key=operator.itemgetter(1))[0]
        conf = max(pred_items, key=operator.itemgetter(1))[1]
        conf_percentage = (conf * 100)
        percent_str = str(conf_percentage) + "%"
```

```
        print("Ground Truth:", img_name, "Prediction:", pred, "Confidence",␣
↪percent_str, "%")
```

```
Ground Truth: cat.94.jpg Prediction: cats_tag Confidence 99.95722%
Ground Truth: cat.95.jpg Prediction: cats_tag Confidence 99.933004%
Ground Truth: dog.95.jpg Prediction: dogs_tag Confidence 99.57066%
Ground Truth: dog.98.jpg Prediction: dogs_tag Confidence 99.897003%
Ground Truth: dog.97.jpg Prediction: dogs_tag Confidence 99.996746%
Ground Truth: dog.91.jpg Prediction: dogs_tag Confidence 99.3909%
Ground Truth: cat.91.jpg Prediction: cats_tag Confidence 99.82772%
Ground Truth: dog.92.jpg Prediction: dogs_tag Confidence 99.316627%
Ground Truth: dog.99.jpg Prediction: dogs_tag Confidence 99.98796%
Ground Truth: dog.94.jpg Prediction: dogs_tag Confidence 99.65952999999999%
Ground Truth: cat.98.jpg Prediction: cats_tag Confidence 99.9577%
Ground Truth: dog.96.jpg Prediction: dogs_tag Confidence 99.60526%
Ground Truth: dog.93.jpg Prediction: dogs_tag Confidence 99.98519399999999%
Ground Truth: cat.99.jpg Prediction: cats_tag Confidence 99.988747%
Ground Truth: dog.100.jpg Prediction: dogs_tag Confidence 99.987555%
Ground Truth: cat.100.jpg Prediction: cats_tag Confidence 99.957865%
Ground Truth: cat.96.jpg Prediction: cats_tag Confidence 99.95153%
Ground Truth: cat.97.jpg Prediction: cats_tag Confidence 99.95941499999999%
Ground Truth: cat.93.jpg Prediction: cats_tag Confidence 99.988055%
Ground Truth: cat.92.jpg Prediction: cats_tag Confidence 99.05807399999999%
```

```
[ ]:
```