SVM Algorithm

from google.colab.patches import cv2\_imshow

import cv2

import numpy as np

from sklearn.svm import SVC

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

import os

import tensorflow as tf

import tensorflow\_hub as hub

# Load Faster R-CNN model

object\_detection\_model\_url = "https://tfhub.dev/tensorflow/faster\_rcnn/inception\_resnet\_v2\_640x640/1"

object\_detection\_model = hub.load(object\_detection\_model\_url)

# Load SVM model

svm\_classifier = SVC(kernel='linear', random\_state=42)

# Input and output folders

input\_folder = "/content/drive/MyDrive/Object Measurement (1)"

output\_folder = "/content/drive/MyDrive/Output Folder 3"

os.makedirs(output\_folder, exist\_ok=True)

# Create a text file to store measurements

measurements\_file\_path = "/content/drive/MyDrive/test 3.txt"

measurements\_file = open(measurements\_file\_path, "w")

# Prepare data for image classification

data = []

labels = []

# Process each image in the input folder

for filename in os.listdir(input\_folder):

    # Construct the path to the image

    image\_path = os.path.join(input\_folder, filename)

    # Load the image

    frame = cv2.imread(image\_path)

    # Check if the image is loaded successfully

    if frame is None:

        print(f"Error loading image: {image\_path}")

        continue

    # Convert the frame to grayscale (assuming features are extracted from grayscale images)

    gray\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

    # Extract features (for simplicity, using the average pixel intensity as a feature)

    features = np.mean(gray\_frame)

    # Classify based on some condition (e.g., average pixel intensity threshold)

    label = "Class\_A" if features > 100 else "Class\_B"

    # Display class label and measurement

    measurement = f"Average Pixel Intensity: {features:.2f}"

    print(f"Image: {filename}, {measurement}")

    # Write measurements to the text file

    measurements\_file.write(f"Image: {filename}, {measurement}\n")

    # Append data for classification

    data.append(features)

    labels.append(label)

    # Object Detection using Faster R-CNN

    rgb\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

    input\_tensor = tf.image.resize(rgb\_frame, (640, 640))

    input\_tensor = tf.cast(input\_tensor, dtype=tf.uint8)  # Convert to uint8

    input\_tensor = tf.expand\_dims(input\_tensor, axis=0)

    detection\_results = object\_detection\_model(input\_tensor)

    # Process detections

    boxes = detection\_results['detection\_boxes'][0].numpy()

    scores = detection\_results['detection\_scores'][0].numpy()

    classes = detection\_results['detection\_classes'][0].numpy().astype(int)

    # Display detections on the image

    for i in range(len(boxes)):

        if scores[i] > 0.5:  # Confidence threshold

            box = boxes[i]

            class\_id = classes[i]

            ymin, xmin, ymax, xmax = box

            xmin, xmax, ymin, ymax = int(xmin \* frame.shape[1]), int(xmax \* frame.shape[1]), \

                                     int(ymin \* frame.shape[0]), int(ymax \* frame.shape[0])

            # Draw bounding box

            cv2.rectangle(frame, (xmin, ymin), (xmax, ymax), (0, 255, 0), 2)

            # Display class label and confidence score

            label = f"Class {class\_id} - Score: {scores[i]:.2f}"

            cv2.putText(frame, label, (xmin, ymin - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 0), 2)

            # Write measurements to the text file

            measurements\_file.write(f"Image: {filename}, {label}\n")

    # Save the annotated image to the output folder

    output\_path = os.path.join(output\_folder, f"detection\_output\_{filename}")

    cv2.imwrite(output\_path, cv2.cvtColor(frame, cv2.COLOR\_RGB2BGR))

# Close the measurements file

measurements\_file.close()

# Convert labels to numerical format

label\_encoder = LabelEncoder()

labels\_encoded = label\_encoder.fit\_transform(labels)

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data, labels\_encoded, test\_size=0.2, random\_state=42)

# Train SVM

svm\_classifier.fit(np.array(X\_train).reshape(-1, 1), y\_train)

# Predict on the test set

y\_pred\_svm = svm\_classifier.predict(np.array(X\_test).reshape(-1, 1))

# Evaluate the accuracy for SVM

accuracy\_svm = accuracy\_score(y\_test, y\_pred\_svm)

print(f"SVM Accuracy: {accuracy\_svm:.2f}")

Random Forest Algorithm

from google.colab.patches import cv2\_imshow

import cv2

import numpy as np

from sklearn.ensemble import RandomForestClassifier

from sklearn.preprocessing import LabelEncoder # Add the import

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

import os

import tensorflow as tf

import tensorflow\_hub as hub

# Load Faster R-CNN model

object\_detection\_model\_url = "https://tfhub.dev/tensorflow/faster\_rcnn/inception\_resnet\_v2\_640x640/1"

object\_detection\_model = hub.load(object\_detection\_model\_url)

# Load Random Forest model

rf\_classifier = RandomForestClassifier(random\_state=42)

# Input and output folders

input\_folder = "/content/drive/MyDrive/Object Measurement"

output\_folder = "/content/drive/MyDrive/Output Folder 4"

os.makedirs(output\_folder, exist\_ok=True)

# Create a text file to store measurements

measurements\_file\_path = "/content/drive/MyDrive/test 4.txt"

measurements\_file = open(measurements\_file\_path, "w")

# Prepare data for image classification

data = []

labels = []

# Process each image in the input folder

for filename in os.listdir(input\_folder):

# Construct the path to the image

image\_path = os.path.join(input\_folder, filename)

# Load the image

frame = cv2.imread(image\_path)

# Check if the image is loaded successfully

if frame is None:

print(f"Error loading image: {image\_path}")

continue

# Convert the frame to grayscale (assuming features are extracted from grayscale images)

gray\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

# Extract features (for simplicity, using the average pixel intensity as a feature)

features = np.mean(gray\_frame)

# Classify based on some condition (e.g., average pixel intensity threshold)

label = "Class\_A" if features > 100 else "Class\_B"

# Display class label and measurement

measurement = f"Average Pixel Intensity: {features:.2f}"

print(f"Image: {filename}, {measurement}")

# Write measurements to the text file

measurements\_file.write(f"Image: {filename}, {measurement}\n")

# Append data for classification

data.append(features)

labels.append(label)

# Object Detection using Faster R-CNN

rgb\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

input\_tensor = tf.image.resize(rgb\_frame, (640, 640))

input\_tensor = tf.cast(input\_tensor, dtype=tf.uint8) # Convert to uint8

input\_tensor = tf.expand\_dims(input\_tensor, axis=0)

detection\_results = object\_detection\_model(input\_tensor)

# Process detections

boxes = detection\_results['detection\_boxes'][0].numpy()

scores = detection\_results['detection\_scores'][0].numpy()

classes = detection\_results['detection\_classes'][0].numpy().astype(int)

# Display detections on the image

for i in range(len(boxes)):

if scores[i] > 0.5: # Confidence threshold

box = boxes[i]

class\_id = classes[i]

ymin, xmin, ymax, xmax = box

xmin, xmax, ymin, ymax = int(xmin \* frame.shape[1]), int(xmax \* frame.shape[1]), \

int(ymin \* frame.shape[0]), int(ymax \* frame.shape[0])

# Draw bounding box in red color

cv2.rectangle(frame, (xmin, ymin), (xmax, ymax), (0, 0, 255), 2)

# Display class label and confidence score

label = f"Class {class\_id} - Score: {scores[i]:.2f}"

cv2.putText(frame, label, (xmin, ymin - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 255), 2)

# Write measurements to the text file

measurements\_file.write(f"Image: {filename}, {label}\n")

# Save the annotated image to the output folder

output\_path = os.path.join(output\_folder, f"detection\_output\_{filename}")

cv2.imwrite(output\_path, cv2.cvtColor(frame, cv2.COLOR\_RGB2BGR))

# Close the measurements file

measurements\_file.close()

# Convert labels to numerical format

label\_encoder = LabelEncoder()

labels\_encoded = label\_encoder.fit\_transform(labels)

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data, labels\_encoded, test\_size=0.2, random\_state=42)

# Train Random Forest

rf\_classifier.fit(np.array(X\_train).reshape(-1, 1), y\_train)

# Predict on the test set

y\_pred\_rf = rf\_classifier.predict(np.array(X\_test).reshape(-1, 1))

# Evaluate the accuracy for Random Forest

accuracy\_rf = accuracy\_score(y\_test, y\_pred\_rf)

print(f"Random Forest Accuracy: {accuracy\_rf:.2f}")

SSD (Single Short Multi box Detector) Algorithm

from google.colab.patches import cv2\_imshow

import cv2

import numpy as np

import tensorflow as tf

import tensorflow\_hub as hub

import os

# Load MobileNet SSD model

model\_url = "https://tfhub.dev/tensorflow/ssd\_mobilenet\_v2/2"

model = hub.load(model\_url)

# Input and output folders

input\_folder = "/content/drive/MyDrive/Object Measurement"

output\_folder = "/content/drive/MyDrive/Output folder"

os.makedirs(output\_folder, exist\_ok=True)

# Create a text file to store measurements

measurements\_file\_path = "/content/drive/MyDrive/test.txt"

measurements\_file = open(measurements\_file\_path, "w")

# Process each image in the input folder

for filename in os.listdir(input\_folder):

    # Construct the path to the image

    image\_path = os.path.join(input\_folder, filename)

    # Load the image

    frame = cv2.imread(image\_path)

    # Check if the image is loaded successfully

    if frame is None:

        print(f"Error loading image: {image\_path}")

        continue

    # Convert the frame to RGB

    rgb\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

    # Resize the image to fit the model input size

    input\_tensor = tf.image.resize(rgb\_frame, (300, 300))

    input\_tensor = tf.cast(input\_tensor, dtype=tf.uint8)  # Convert to uint8

    input\_tensor = tf.expand\_dims(input\_tensor, axis=0)

    # Perform inference

    detection\_results = model(input\_tensor)

    # Process detections

    boxes = detection\_results['detection\_boxes'][0].numpy()

    scores = detection\_results['detection\_scores'][0].numpy()

    classes = detection\_results['detection\_classes'][0].numpy().astype(int)

    # Display detections on the image

    for i in range(len(boxes)):

        if scores[i] > 0.5:  # Confidence threshold

            box = boxes[i]

            class\_id = classes[i]

            ymin, xmin, ymax, xmax = box

            xmin, xmax, ymin, ymax = int(xmin \* frame.shape[1]), int(xmax \* frame.shape[1]), \

                                     int(ymin \* frame.shape[0]), int(ymax \* frame.shape[0])

            # Draw bounding box

            cv2.rectangle(frame, (xmin, ymin), (xmax, ymax), (0, 255, 0), 2)

            # Display class label, confidence score, and measurement

            label = f"Class {class\_id} - Score: {scores[i]:.2f}"

            measurement = f"Width: {xmax - xmin}, Height: {ymax - ymin}"

            cv2.putText(frame, label, (xmin, ymin - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 0), 2)

            cv2.putText(frame, measurement, (xmin, ymin + 20), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 0), 2)

            # Write measurements to the text file

            measurements\_file.write(f"Image: {filename}, {measurement}\n")

    # Save the annotated image to the output folder

    output\_path = os.path.join(output\_folder, f"detection\_output\_{filename}")

    cv2.imwrite(output\_path, cv2.cvtColor(frame, cv2.COLOR\_RGB2BGR))

# Close the measurements file

measurements\_file.close()

# Display the saved output images

output\_images = [os.path.join(output\_folder, f) for f in os.listdir(output\_folder)]

for output\_image in output\_images:

    img = cv2.imread(output\_image)

    cv2\_imshow(img)

# Print a message indicating that measurements are saved to the text file

print(f"Measurements saved to: {measurements\_file\_path}")

EfficientDet Algorithm

from google.colab.patches import cv2\_imshow

import cv2

import numpy as np

import tensorflow as tf

import tensorflow\_hub as hub

import os

# Load EfficientDet model

model\_url = "https://tfhub.dev/tensorflow/efficientdet/d7/1"

model = hub.load(model\_url)

# Input and output folders

input\_folder = "/content/drive/MyDrive/Object Measurement"

output\_folder = "/content/drive/MyDrive/Output folder 2"

os.makedirs(output\_folder, exist\_ok=True)

# Create a text file to store measurements

measurements\_file\_path = "/content/drive/MyDrive/test 1.txt"

measurements\_file = open(measurements\_file\_path, "w")

# Process each image in the input folder

for filename in os.listdir(input\_folder):

    # Construct the path to the image

    image\_path = os.path.join(input\_folder, filename)

    # Load the image

    frame = cv2.imread(image\_path)

    # Check if the image is loaded successfully

    if frame is None:

        print(f"Error loading image: {image\_path}")

        continue

    # Convert the frame to RGB

    rgb\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

    # Resize the image to fit the model input size

    input\_tensor = tf.image.resize(rgb\_frame, (640, 640))

    input\_tensor = tf.cast(input\_tensor, dtype=tf.uint8)  # Convert to uint8

    input\_tensor = tf.expand\_dims(input\_tensor, axis=0)

    # Perform EfficientDet inference

    detection\_results = model(input\_tensor)

    # Process detections

    boxes = detection\_results['detection\_boxes'][0].numpy()

    scores = detection\_results['detection\_scores'][0].numpy()

    classes = detection\_results['detection\_classes'][0].numpy().astype(int)

    # Display detections on the image

    for i in range(len(boxes)):

        if scores[i] > 0.5:  # Confidence threshold

            box = boxes[i]

            class\_id = classes[i]

            ymin, xmin, ymax, xmax = box

            xmin, xmax, ymin, ymax = int(xmin \* frame.shape[1]), int(xmax \* frame.shape[1]), \

                                     int(ymin \* frame.shape[0]), int(ymax \* frame.shape[0])

            # Draw bounding box

            cv2.rectangle(frame, (xmin, ymin), (xmax, ymax), (0, 255, 0), 2)

            # Display class label, confidence score, and measurement

            label = f"Class {class\_id} - Score: {scores[i]:.2f}"

            measurement = f"Width: {xmax - xmin}, Height: {ymax - ymin}"

            cv2.putText(frame, label, (xmin, ymin - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 0), 2)

            cv2.putText(frame, measurement, (xmin, ymin + 20), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 0), 2)

            # Write measurements to the text file

            measurements\_file.write(f"Image: {filename}, {measurement}\n")

    # Save the annotated image to the output folder

    output\_path = os.path.join(output\_folder, f"detection\_output\_{filename}")

    cv2.imwrite(output\_path, cv2.cvtColor(frame, cv2.COLOR\_RGB2BGR))

# Close the measurements file

measurements\_file.close()

RCNN Algorithm

from google.colab.patches import cv2\_imshow

import cv2

import numpy as np

import tensorflow as tf

import tensorflow\_hub as hub

import os

# Load Faster R-CNN model

model\_url = "https://tfhub.dev/tensorflow/faster\_rcnn/inception\_resnet\_v2\_640x640/1"

model = hub.load(model\_url)

# Input and output folders

input\_folder = "/content/drive/MyDrive/Object Measurement"

output\_folder = "/content/drive/MyDrive/Output Folder 1"

os.makedirs(output\_folder, exist\_ok=True)

# Create a text file to store measurements

measurements\_file\_path = "/content/drive/MyDrive/test.txt"

measurements\_file = open(measurements\_file\_path, "w")

# Process each image in the input folder

for filename in os.listdir(input\_folder):

    # Construct the path to the image

    image\_path = os.path.join(input\_folder, filename)

    # Load the image

    frame = cv2.imread(image\_path)

    # Check if the image is loaded successfully

    if frame is None:

        print(f"Error loading image: {image\_path}")

        continue

    # Convert the frame to RGB

    rgb\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

    # Resize the image to fit the model input size

    resized\_frame = tf.image.resize(rgb\_frame, (640, 640))

    # Convert to tf.uint8 and normalize pixel values to [0, 255]

    input\_tensor = tf.cast(resized\_frame, dtype=tf.uint8)

    input\_tensor = tf.expand\_dims(input\_tensor, axis=0)

    # Perform Faster R-CNN inference

    detection\_results = model(input\_tensor)

    # Rest of your code remains unchanged...

    # (processing detections, drawing bounding boxes, saving images, etc.)

    # Process detections

    boxes = detection\_results['detection\_boxes'][0].numpy()

    scores = detection\_results['detection\_scores'][0].numpy()

    classes = detection\_results['detection\_classes'][0].numpy().astype(int)

    # Display detections on the image

    for i in range(len(boxes)):

        if scores[i] > 0.5:  # Confidence threshold

            box = boxes[i]

            class\_id = classes[i]

            ymin, xmin, ymax, xmax = box

            xmin, xmax, ymin, ymax = int(xmin \* frame.shape[1]), int(xmax \* frame.shape[1]), \

                                     int(ymin \* frame.shape[0]), int(ymax \* frame.shape[0])

            # Draw bounding box

            cv2.rectangle(frame, (xmin, ymin), (xmax, ymax), (0, 255, 0), 2)

            # Display class label, confidence score, and measurement

            label = f"Class {class\_id} - Score: {scores[i]:.2f}"

            measurement = f"Width: {xmax - xmin}, Height: {ymax - ymin}"

            cv2.putText(frame, label, (xmin, ymin - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 0), 2)

            cv2.putText(frame, measurement, (xmin, ymin + 20), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 0), 2)

            # Write measurements to the text file

            measurements\_file.write(f"Image: {filename}, {measurement}\n")

    # Save the annotated image to the output folder

    output\_path = os.path.join(output\_folder, f"detection\_output\_{filename}")

    cv2.imwrite(output\_path, cv2.cvtColor(frame, cv2.COLOR\_RGB2BGR))

# Close the measurements file

measurements\_file.close()

# Display the saved output images

output\_images = [os.path.join(output\_folder, f) for f in os.listdir(output\_folder)]

for output\_image in output\_images:

    img = cv2.imread(output\_image)

    cv2\_imshow(img)

# Print a message indicating that measurements are saved to the text file

print(f"Measurements saved to: {measurements\_file\_path}")

R-CNN ALGORITHM