Version 1

# Install YOLOv8 library

!pip install ultralytics

import cv2

import numpy as np

from ultralytics import YOLO

from google.colab.patches import cv2\_imshow

# Step 1: Load Image

def load\_image(image\_path):

print("[Step 1] Loading the image...")

image = cv2.imread(image\_path)

if image is None:

raise ValueError("Image not found. Please check the file path.")

print(f"Image loaded successfully: {image\_path}")

return image

# Step 2: Segment the Image

def segment\_image(image, grid\_size=4):

print("[Step 2] Segmenting the image...")

h, w, \_ = image.shape

segments = []

step\_h, step\_w = h // grid\_size, w // grid\_size

for i in range(grid\_size):

for j in range(grid\_size):

x\_min, x\_max = j \* step\_w, (j + 1) \* step\_w

y\_min, y\_max = i \* step\_h, (i + 1) \* step\_h

segment = image[y\_min:y\_max, x\_min:x\_max]

segments.append(((x\_min, y\_min, x\_max, y\_max), segment))

print(f"Image divided into {len(segments)} segments.")

return segments

# Step 3: Detect Suspicious Segments

def detect\_suspicious\_segments(image, segments, model, conf\_threshold=0.5):

print("[Step 3] Detecting suspicious segments...")

flagged\_segments = []

for (x\_min, y\_min, x\_max, y\_max), segment in segments:

results = model.predict(source=segment, conf=conf\_threshold, save=False, save\_txt=False)

if results[0].boxes.xyxy.shape[0] > 0: # If any objects are detected

flagged\_segments.append((x\_min, y\_min, x\_max, y\_max))

print(f"{len(flagged\_segments)} segments flagged as suspicious.")

return flagged\_segments

# Step 4: Neutralize Flagged Segments

def neutralize\_segments(image, flagged\_segments):

print("[Step 4] Neutralizing flagged segments...")

h, w, \_ = image.shape

median\_color = np.median(image, axis=(0, 1)) # Calculate median color of the image

for x\_min, y\_min, x\_max, y\_max in flagged\_segments:

image[y\_min:y\_max, x\_min:x\_max] = median\_color # Replace with median color

print("Flagged segments successfully neutralized.")

return image

# Step 5: Evaluate Model's Confidence

def evaluate\_model():

print("[Step 5] Evaluating the model's confidence...")

print("Model evaluation completed.") # Placeholder for actual model evaluation logic

return # No return since evaluation is external to this pipeline

# Step 6: Calculate Metrics

def calculate\_metrics(total\_segments, flagged\_segments):

print("[Step 6] Calculating metrics...")

detection\_rate = (len(flagged\_segments) / total\_segments) \* 100

print(f"Detection Rate: {detection\_rate:.2f}%")

return detection\_rate

# Main Pipeline

def adversarial\_patch\_pipeline(image\_path, conf\_threshold=0.5, grid\_size=4):

# Step 1: Load the image

image = load\_image(image\_path)

# Step 2: Segment the image

segments = segment\_image(image, grid\_size=grid\_size)

# Step 3: Load YOLOv8 model and detect suspicious segments

print("[INFO] Loading YOLOv8 model...")

model = YOLO("yolov8n.pt") # Pre-trained YOLOv8 model

print("[INFO] YOLOv8 model loaded successfully!")

flagged\_segments = detect\_suspicious\_segments(image, segments, model, conf\_threshold)

# Step 4: Neutralize flagged segments

cleaned\_image = neutralize\_segments(image.copy(), flagged\_segments)

# Step 5: Evaluate the model

evaluate\_model()

# Step 6: Calculate metrics

detection\_rate = calculate\_metrics(len(segments), flagged\_segments)

# Show results

print("\nOriginal Image:")

cv2\_imshow(image)

print("\nCleaned Image:")

cv2\_imshow(cleaned\_image)

return flagged\_segments, detection\_rate

# Example Usage

input\_image\_path = "patched\_image.jpg" # Replace with your input image path

flagged\_segments, detection\_rate = adversarial\_patch\_pipeline(input\_image\_path)

Second Version

import cv2

import numpy as np

from tensorflow.keras.applications import VGG16

from tensorflow.keras.applications.vgg16 import preprocess\_input, decode\_predictions

from google.colab.patches import cv2\_imshow

# Load the image

def load\_image(image\_path):

    print("[Step 1] Loading the image...")

    image = cv2.imread(image\_path)

    if image is None:

        raise ValueError("Image not found. Please check the file path.")

    print(f"Image loaded successfully: {image\_path}")

    return image

# Segment the image

def segment\_image(image, grid\_size=(4, 4)):

    print("[Step 2] Segmenting the image...")

    h, w, c = image.shape

    grid\_h, grid\_w = h // grid\_size[0], w // grid\_size[1]

    segments = []

    for i in range(grid\_size[0]):

        for j in range(grid\_size[1]):

            x\_start, x\_end = j \* grid\_w, (j + 1) \* grid\_w

            y\_start, y\_end = i \* grid\_h, (i + 1) \* grid\_h

            segment = image[y\_start:y\_end, x\_start:x\_end]

            segments.append(((x\_start, y\_start), segment))

    print(f"Image divided into {len(segments)} segments.")

    return segments

# Detect suspicious segments using anomaly scores

def detect\_suspicious\_segments(segments, threshold=50):

    print("[Step 3] Detecting suspicious segments...")

    flagged\_segments = []

    for pos, segment in segments:

        # Calculate anomaly score as standard deviation of pixel values

        anomaly\_score = np.std(segment)

        if anomaly\_score > threshold:  # Flag segments with high variability

            flagged\_segments.append(pos)

    print(f"{len(flagged\_segments)} segments flagged as suspicious.")

    return flagged\_segments

# Neutralize flagged segments

def neutralize\_segments(image, flagged\_segments, grid\_size=(4, 4)):

    print("[Step 4] Neutralizing flagged segments...")

    h, w, c = image.shape

    grid\_h, grid\_w = h // grid\_size[0], w // grid\_size[1]

    for x\_start, y\_start in flagged\_segments:

        x\_end, y\_end = x\_start + grid\_w, y\_start + grid\_h

        # Replace flagged areas with the median color value of the entire image

        image[y\_start:y\_end, x\_start:x\_end] = np.median(image, axis=(0, 1))

    print("Flagged segments successfully neutralized.")

    return image

# Evaluate the model's confidence

def evaluate\_image(model, image):

    print("[Step 5] Evaluating the model's confidence...")

    resized\_image = cv2.resize(image, (224, 224))

    input\_image = preprocess\_input(np.expand\_dims(resized\_image, axis=0))

    predictions = model.predict(input\_image)

    decoded\_predictions = decode\_predictions(predictions, top=3)

    print("Model evaluation completed.")

    return decoded\_predictions

# Metrics

def calculate\_metrics(flagged\_segments, total\_segments):

    print("[Step 6] Calculating metrics...")

    detection\_rate = len(flagged\_segments) / total\_segments \* 100

    print(f"Detection Rate: {detection\_rate:.2f}%")

    return {"detection\_rate": detection\_rate}

# Complete pipeline

def adversarial\_patch\_neutralization(image\_path, grid\_size=(4, 4), threshold=50):

    # Step 1: Load the image

    image = load\_image(image\_path)

    # Step 2: Segment the image

    segments = segment\_image(image, grid\_size)

    total\_segments = len(segments)

    # Step 3: Detect suspicious segments

    flagged\_segments = detect\_suspicious\_segments(segments, threshold)

    # Step 4: Neutralize disturbances

    cleaned\_image = neutralize\_segments(image.copy(), flagged\_segments, grid\_size)

    # Step 5: Evaluate performance

    model = VGG16(weights="imagenet")

    original\_confidence = evaluate\_image(model, image)

    cleaned\_confidence = evaluate\_image(model, cleaned\_image)

    # Step 6: Calculate metrics

    metrics = calculate\_metrics(flagged\_segments, total\_segments)

    return cleaned\_image, original\_confidence, cleaned\_confidence, metrics

# Example usage

input\_image\_path = "Test1.jpg"  # Replace with the path to your image

cleaned\_image, original\_conf, cleaned\_conf, metrics = adversarial\_patch\_neutralization(input\_image\_path)

# Print results

print("\n--- Results ---")

print("Original Confidence:", original\_conf)

print("Cleaned Confidence:", cleaned\_conf)

print("Metrics:", metrics)

# Display images

print("\nOriginal Image:")

cv2\_imshow(cv2.imread(input\_image\_path))

print("\nCleaned Image:")

cv2\_imshow(cleaned\_image)