Circle_Detection_ML_Challenge

November 29, 2023

[1]:
```
#starter code
from typing import NamedTuple, Optional, Tuple, Generator

import numpy as np
from matplotlib import pyplot as plt
from skimage.draw import circle_perimeter_aa


class CircleParams(NamedTuple):
    row: int
    col: int
    radius: int
def draw_circle(img: np.ndarray, row: int, col: int, radius: int) -> np.ndarray:
    """
    Draw a circle in a numpy array, inplace.
    The center of the circle is at (row, col) and the radius is given by radius.
    The array is assumed to be square.
    Any pixels outside the array are ignored.
    Circle is white (1) on black (0) background, and is anti-aliased.
    """
    rr, cc, val = circle_perimeter_aa(row, col, radius)
    valid = (rr >= 0) & (rr < img.shape[0]) & (cc >= 0) & (cc < img.shape[1])
    img[rr[valid], cc[valid]] = val[valid]
    return img


def noisy_circle(
        img_size: int, min_radius: float, max_radius: float, noise_level: float
) -> Tuple[np.ndarray, CircleParams]:
    """
    Draw a circle in a numpy array, with normal noise.
    """

    # Create an empty image
    img = np.zeros((img_size, img_size))

    radius = np.random.randint(min_radius, max_radius)
```

```
    # x,y coordinates of the center of the circle
    row, col = np.random.randint(img_size, size=2)

    # Draw the circle inplace
    draw_circle(img, row, col, radius)

    added_noise = np.random.normal(0.5, noise_level, img.shape)
    img += added_noise

    return img, CircleParams(row, col, radius)


def show_circle(img: np.ndarray):
    fig, ax = plt.subplots()
    ax.imshow(img, cmap='gray')
    ax.set_title('Circle')
    plt.show()


def generate_examples(
        noise_level: float = 0.5,
        img_size: int = 100,
        min_radius: Optional[int] = None,
        max_radius: Optional[int] = None,
        dataset_path: str = 'ds',
) -> Generator[Tuple[np.ndarray, CircleParams], None, None]:
    if not min_radius:
        min_radius = img_size // 10
    if not max_radius:
        max_radius = img_size // 2
    assert max_radius > min_radius, "max_radius must be greater than min_radius"
    assert img_size > max_radius, "size should be greater than max_radius"
    assert noise_level >= 0, "noise should be non-negative"

    params = f"{noise_level=}, {img_size=}, {min_radius=}, {max_radius=},␣
 ↪{dataset_path=}"
    print(f"Using parameters: {params}")
    while True:
        img, params = noisy_circle(
            img_size=img_size, min_radius=min_radius, max_radius=max_radius,␣
 ↪noise_level=noise_level
        )
        yield img, params


def iou(a: CircleParams, b: CircleParams) -> float:
```

```
    """Calculate the intersection over union of two circles"""
    r1, r2 = a.radius, b.radius
    d = np.linalg.norm(np.array([a.row, a.col]) - np.array([b.row, b.col]))
    if d > r1 + r2:
        # If the distance between the centers is greater than the sum of the␣
 ↪radii, then the circles don't intersect
        return 0.0
    if d <= abs(r1 - r2):
        # If the distance between the centers is less than the absolute␣
 ↪difference of the radii, then one circle is
        # inside the other
        larger_r, smaller_r = max(r1, r2), min(r1, r2)
        return smaller_r ** 2 / larger_r ** 2
    r1_sq, r2_sq = r1**2, r2**2
    d1 = (r1_sq - r2_sq + d**2) / (2 * d)
    d2 = d - d1
    sector_area1 = r1_sq * np.arccos(d1 / r1)
    triangle_area1 = d1 * np.sqrt(r1_sq - d1**2)
    sector_area2 = r2_sq * np.arccos(d2 / r2)
    triangle_area2 = d2 * np.sqrt(r2_sq - d2**2)
    intersection = sector_area1 + sector_area2 - (triangle_area1 +␣
 ↪triangle_area2)
    union = np.pi * (r1_sq + r2_sq) - intersection
    return intersection / union
```

[2]:
```
#data preparation
import numpy as np

def create_dataset(num_samples, img_size=100, noise_level=0.5, min_radius=None,␣
 ↪max_radius=None):
    gen = generate_examples(noise_level, img_size, min_radius, max_radius)
    dataset = []
    labels = []

    for _ in range(num_samples):
        img, params = next(gen)
        dataset.append(img)
        labels.append([params.row, params.col, params.radius])

    return np.array(dataset), np.array(labels)

# Example usage
num_samples = 10000  # for example, 10,000 samples
dataset, labels = create_dataset(num_samples)

# Save dataset
np.save('dataset.npy', dataset)
```

```
np.save('labels.npy', labels)
```

```
Using parameters: noise_level=0.5, img_size=100, min_radius=10, max_radius=50,
dataset_path='ds'
```

[3]:
```
# Load the dataset
dataset = np.load('dataset.npy')
labels = np.load('labels.npy')
```

[4]:
```
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import normalize

# Normalize the image data
dataset_normalized = normalize(dataset, axis=1)

# Reshape the dataset for CNN (adding a channel dimension)
# Assuming the images are grayscale, hence the channel dimension is 1
dataset_reshaped = dataset_normalized.reshape((-1, 100, 100, 1))

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(dataset_reshaped, labels,␣
 ↪test_size=0.2, random_state=42)

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

[4]:
```
((8000, 100, 100, 1), (2000, 100, 100, 1), (8000, 3), (2000, 3))
```

[5]:
```
import tensorflow as tf

def custom_loss_function(y_true, y_pred):
    # Assuming y_true and y_pred are both in the format [x, y, radius]
    loss_x = tf.reduce_mean(tf.abs(y_true[:, 0] - y_pred[:, 0]))
    loss_y = tf.reduce_mean(tf.abs(y_true[:, 1] - y_pred[:, 1]))
    loss_radius = tf.reduce_mean(tf.square(y_true[:, 2] - y_pred[:, 2]))

    # Weighted sum of the losses
    return loss_x + loss_y + 0.5 * loss_radius
```

[6]:
```
import tensorflow as tf

def iou_metric(y_true, y_pred):
    # Assuming y_true and y_pred are tensors with shape [batch_size, 3]
    # where the three columns are row, col, and radius of the circles
    r1 = y_true[:, 2]
    r2 = y_pred[:, 2]
    d = tf.norm(y_true[:, :2] - y_pred[:, :2], axis=1)

    condition1 = tf.less_equal(d, tf.abs(r1 - r2))
```

```
    condition2 = tf.greater(d, r1 + r2)

    larger_r = tf.where(r1 > r2, r1, r2)
    smaller_r = tf.where(r1 > r2, r2, r1)

    iou_inside = tf.where(condition1, tf.square(smaller_r) / tf.
 ↪square(larger_r), tf.zeros_like(r1))
    iou_no_overlap = tf.where(condition2, tf.zeros_like(r1), tf.zeros_like(r1))

    r1_sq = tf.square(r1)
    r2_sq = tf.square(r2)
    d1 = (r1_sq - r2_sq + tf.square(d)) / (2 * d)
    d2 = d - d1

    sector_area1 = r1_sq * tf.acos(d1 / r1)
    triangle_area1 = d1 * tf.sqrt(r1_sq - tf.square(d1))
    sector_area2 = r2_sq * tf.acos(d2 / r2)
    triangle_area2 = d2 * tf.sqrt(r2_sq - tf.square(d2))

    intersection = sector_area1 + sector_area2 - (triangle_area1 +␣
 ↪triangle_area2)
    union = np.pi * (r1_sq + r2_sq) - intersection

    iou_overlap = tf.where(tf.logical_not(tf.logical_or(condition1,␣
 ↪condition2)), intersection / union, tf.zeros_like(r1))

    return tf.reduce_mean(tf.where(condition1, iou_inside, tf.where(condition2,␣
 ↪iou_no_overlap, iou_overlap)))
```

[ ]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Define the model
model = Sequential([
    # Convolutional layer 1
    Conv2D(32, (3, 3), activation='relu', input_shape=(100, 100, 1)),
    MaxPooling2D((2, 2)),

    # Convolutional layer 2
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),

    # Convolutional layer 3
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
```

```
    # Flattening layer
    Flatten(),

    # Dense layer
    Dense(128, activation='relu'),

    # Output layer
    Dense(3, activation='linear')  # 3 outputs for x, y, and radius
])

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error', metrics=[iou_metric])

# Model summary
model.summary()

# Train the model
history = model.fit(X_train, y_train, epochs=20, batch_size=32,␣
 ↪validation_data=(X_test, y_test))

# The 'history' object will contain training and validation loss and metrics␣
 ↪records.
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 98, 98, 32)        320

 max_pooling2d (MaxPooling2  (None, 49, 49, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 47, 47, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 23, 23, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 21, 21, 128)       73856

 max_pooling2d_2 (MaxPoolin  (None, 10, 10, 128)       0
 g2D)

 flatten (Flatten)           (None, 12800)             0

 dense (Dense)               (None, 128)               1638528

 dense_1 (Dense)             (None, 3)                 387
```

```
=================================================================
Total params: 1731587 (6.61 MB)
Trainable params: 1731587 (6.61 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/20
250/250 [==============================] - 160s 634ms/step - loss: 664.3457 -
iou_metric: 0.1487 - val_loss: 613.5710 - val_iou_metric: 0.1721
Epoch 2/20
250/250 [==============================] - 161s 642ms/step - loss: 615.1802 -
iou_metric: 0.1543 - val_loss: 606.9290 - val_iou_metric: 0.1744
Epoch 3/20
250/250 [==============================] - 156s 625ms/step - loss: 593.5839 -
iou_metric: 0.1590 - val_loss: 591.4317 - val_iou_metric: 0.1708
Epoch 4/20
250/250 [==============================] - 157s 626ms/step - loss: 565.4077 -
iou_metric: 0.1715 - val_loss: 506.4536 - val_iou_metric: 0.1771
Epoch 5/20
250/250 [==============================] - 155s 619ms/step - loss: 409.4862 -
iou_metric: 0.2451 - val_loss: 309.5942 - val_iou_metric: 0.3067
Epoch 6/20
250/250 [==============================] - 157s 629ms/step - loss: 278.3953 -
iou_metric: 0.3198 - val_loss: 249.5817 - val_iou_metric: 0.3485
Epoch 7/20
250/250 [==============================] - 146s 585ms/step - loss: 233.5523 -
iou_metric: 0.3488 - val_loss: 210.8982 - val_iou_metric: 0.3665
Epoch 8/20
250/250 [==============================] - 147s 589ms/step - loss: 201.3549 -
iou_metric: 0.3761 - val_loss: 192.0311 - val_iou_metric: 0.4026
Epoch 9/20
250/250 [==============================] - 148s 592ms/step - loss: 171.5144 -
iou_metric: 0.4060 - val_loss: 156.7925 - val_iou_metric: 0.4297
Epoch 10/20
250/250 [==============================] - 148s 591ms/step - loss: 147.9548 -
iou_metric: 0.4320 - val_loss: 143.0120 - val_iou_metric: 0.4576
Epoch 11/20
250/250 [==============================] - 148s 591ms/step - loss: 129.8520 -
iou_metric: 0.4566 - val_loss: 124.4865 - val_iou_metric: 0.4707
Epoch 12/20
250/250 [==============================] - 147s 588ms/step - loss: 115.4060 -
iou_metric: 0.4805 - val_loss: 127.8476 - val_iou_metric: 0.4724
Epoch 13/20
250/250 [==============================] - 148s 594ms/step - loss: 101.5594 -
iou_metric: 0.5042 - val_loss: 100.9936 - val_iou_metric: 0.5111
Epoch 14/20
250/250 [==============================] - 149s 595ms/step - loss: 86.4125 -
iou_metric: 0.5327 - val_loss: 101.0887 - val_iou_metric: 0.5222
Epoch 15/20
```

```
250/250 [==============================] - 147s 590ms/step - loss: 79.1424 -
iou_metric: 0.5490 - val_loss: 80.7074 - val_iou_metric: 0.5591
Epoch 16/20
250/250 [==============================] - 147s 590ms/step - loss: 69.6874 -
iou_metric: 0.5711 - val_loss: 75.3551 - val_iou_metric: 0.5656
Epoch 17/20
248/250 [============================>.] - ETA: 1s - loss: 63.6853 - iou_metric:
0.5836
```

[9]:
```python
def evaluate_model(model, X_test, y_test, iou_thresholds: Tuple[float, ...] =␣
↪(0.5, 0.75, 0.9, 0.95)) -> dict:
    """Evaluate the model on the test set using various IOU thresholds."""
    predictions = model.predict(X_test)
    accuracies = {threshold: 0 for threshold in iou_thresholds}

    for pred, true in zip(predictions, y_test):
        pred_circle = CircleParams(*pred)
        true_circle = CircleParams(*true)

        iou_score = iou(pred_circle, true_circle)

        for threshold in iou_thresholds:
            if iou_score >= threshold:
                accuracies[threshold] += 1

    total_samples = len(y_test)
    accuracies = {threshold: acc / total_samples for threshold, acc in␣
↪accuracies.items()}
    return accuracies

# Example of how to use evaluate_model
accuracies = evaluate_model(model, X_test, y_test)
print(accuracies)
```

```
63/63 [==============================] - 19s 298ms/step
{0.5: 0.7135, 0.75: 0.274, 0.9: 0.029, 0.95: 0.0015}
```

[ ]:
```python

```