Reyhane Shahrokhian 99521361

HomeWork7 of Computer Vision Course

Dr. Mohammadi

Q1:

1-1:

$Dilated\ Kernel\ Size = k + (k - 1) \times (d - 1)$

1-2:

The number of trainable parameters in a dilated convolution remains the same as in a regular convolution for dilation rates of 1.

1-3:

Formula of receptive field in each cnn layer:

$RF_0 = 1$

$RF_i = RF_{i-1} + (k - 1) \times d$

So:

Layer	1	2	3	4	5	6	7	8
Convolution	3*3	3*3	3*3	3*3	3*3	5*5	5*5	7*7
Dilation rate	1	1	4	11	8	3	2	6
Receptive field	3*3	5*5	13*13	35*35	51*51	63*63	71*71	107*107

1-4:

Receptive field in each max-pooling layer considering that pool-size and stride are equal:

$RF_0 = 1$

$RF_i = RF_{i-1} + (stride - 1) \times RF_{i-1} = stride \times RF_{i-1}$

So:

After 2 convolutional layers:

$1 + (5 - 1) \times 1 + (5 - 1) \times 1 = 9$

Then:

$s \times 9 = 9s$

$9s \times s = 9s^2$

$9s^2 \times s = 9s^3 \geq 107 \Rightarrow s = 3$

Q2:

2-1:

Standard convolution layer:

- Number of parameters $= k \times k \times input\ channels \times output\ channels$

  $= 5 \times 5 \times 3 \times 64 = 4800$

- Number of operations

  $= k \times k \times input\ channels \times input\ width \times input\ height \times filters$

  $= 5 \times 5 \times 3 \times 128 \times 128 \times 64 = 78643200$

Depthwise separable convolution layer:

- Number of parameters

  $= k \times k \times input\ channels + 1 \times 1 \times input\ channels \times filters$

  $= 5 \times 5 \times 3 + 1 \times 1 \times 3 \times 64 = 267$

- Number of operations $= k \times k \times input\ channels \times input\ width \times input\ height +$ $1 \times 1 \times input\ channels \times filters \times \ input\ width \times input\ height$

  $= 5 \times 5 \times 3 \times 128 \times 128 + 1 \times 1 \times 3 \times 64 \times 128 \times 128 = 4374528$

Depthwise separable convolutions significantly reduce both the number of parameters and the computational cost compared to standard convolutions.

2-2:

Standard convolution parameters: $3 \times 3 \times 32 \times 32 = 9216$

Depthwise separable convolution parameters: $3 \times 3 \times 32 + 1 \times 1 \times 32 \times 32 = 1312$

$\Rightarrow \frac{1312}{9216} = 0.142$

Q3:

3-1:

The second version could be a better choice as it is a normalized version, so it provides us with a more accurate and robust model. Also, the pixels are in a wide value range(0 to 255) which leads us to choose the second one.

3-2:

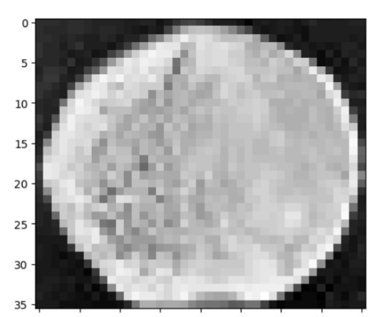
I used the mtm library as it contains a matching template function.

First, the image should be read.

```
coin_image = image.imread('/content/coins.png')
plt.imshow(coin_image, cmap='gray')
```

Then I choose a template.

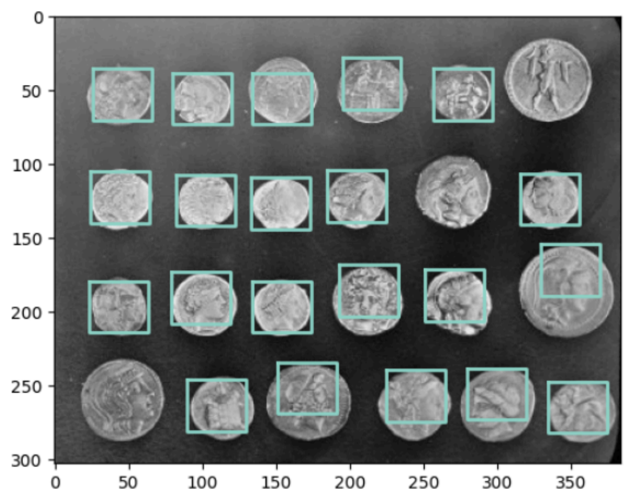
```
# Choose template
template = coin_image[109:145, 133:174]
plt.imshow(template, cmap="gray")
```



Then I called the matchTemplate() function and plotted the detections.

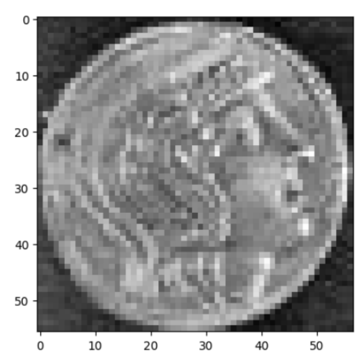
```
# call matchTemplates
detections = mtm.matchTemplates(coin_image, [template], scoreThreshold=0.5, maxOverlap=0)

# Plot the detections
mtm.detection.plotDetections(coin_image, detections)
```

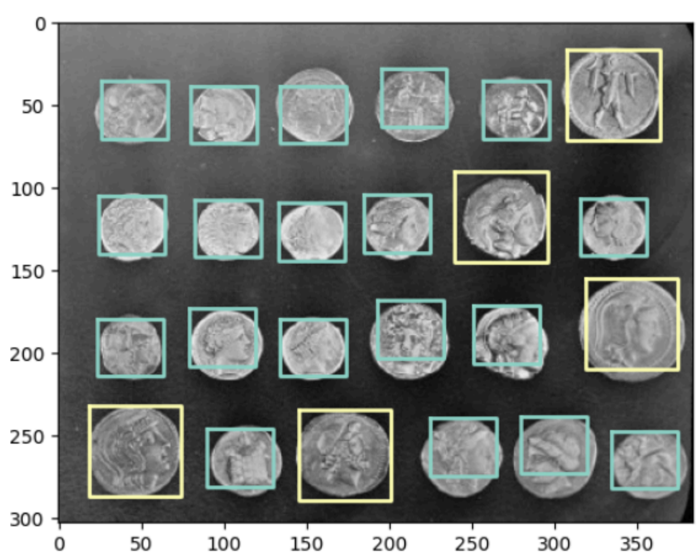


But there are some coins which are bigger than others and they can not be detected accurately. To solve this I added another template.

```
# Another version to detect bigger coins better
big_template = coin_image[232:288, 18:75]
plt.imshow(big_template, cmap="gray")
```



The result after calling matchTemplate on both templates:

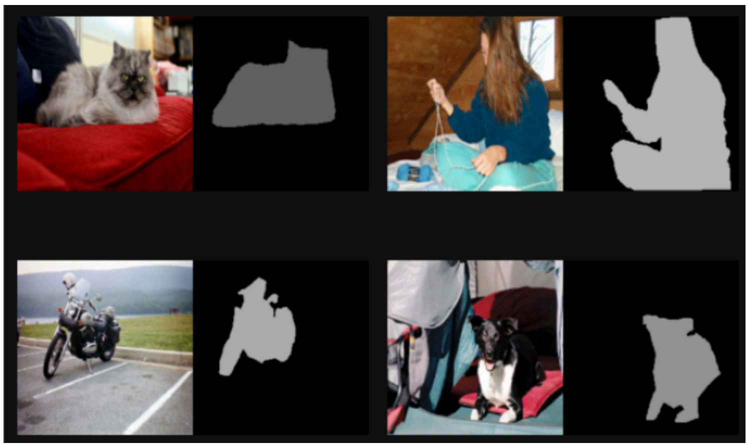


Q4:

The notebook is attached.

Q5:

After installing packages and importing the libraries, we loaded the PasCal dataset. Then, there is a function to preprocess the data(resizing and batching). Here are some example of dataset:



After that some data augmentation techniques like random flipping and random rotation are done. Then, the Unet network is implemented which contains an encoder(blocks of convolutional layers followed by a max-pooling), bottleneck, and decoder(reverse of encoder in which max-pool is replaced with unsampling).

Defining loss functions(containing diceloss and categoricalfocalloss):

```
dice_loss = sm.losses.DiceLoss()
focal_loss = sm.losses.CategoricalFocalLoss()
total_loss = dice_loss + (1 * focal_loss)
```

Beside the accuracy metrice, we introduce jaccard for image segmentation which is also known as IOU.

```
def jaccard_coef(y_true, y_pred):
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)
    intersection = K.sum(y_true_f * y_pred_f)
    return (intersection + 1.0) / (K.sum(y_true_f) + K.sum(y_pred_f) - intersection + 1.0)
```

Then the model is compiled and the model summary is provided in the code.

In the next part we call one-hot vectors on our dataset.

Before training the model, I write a call back to save the best model.

```
# Callback
callbacks = [
    tf.keras.callbacks.ModelCheckpoint(
        filepath='best_model_unet_scratch.keras',
        save_best_only=True,
        monitor='val_loss',
        mode='min'
    )
]
```

This is the result from training the model:

```
Epoch 1/10
    265/Unknown 828s 2s/step - accuracy: 0.6297 - jaccard_coef: 0.3638 - loss: 0.9873/usr/lib/python3.10/contextlib.py:153: UserWarning: Your input ran out of data; inte
  self.gen.throw(typ, value, traceback)
265/265 ━━━━━━━━━━━━━━━━━━━━ 870s 3s/step - accuracy: 0.6297 - jaccard_coef: 0.3637 - loss: 0.9873 - val_accuracy: 0.6226 - val_jaccard_coef: 0.3295 - val_loss: 0.9481
Epoch 2/10
265/265 ━━━━━━━━━━━━━━━━━━━━ 684s 3s/step - accuracy: 0.6252 - jaccard_coef: 0.3572 - loss: 0.9470 - val_accuracy: 0.5961 - val_jaccard_coef: 0.3597 - val_loss: 0.9385
Epoch 3/10
265/265 ━━━━━━━━━━━━━━━━━━━━ 735s 3s/step - accuracy: 0.6078 - jaccard_coef: 0.3763 - loss: 0.9355 - val_accuracy: 0.5720 - val_jaccard_coef: 0.3590 - val_loss: 0.9321
Epoch 4/10
265/265 ━━━━━━━━━━━━━━━━━━━━ 678s 3s/step - accuracy: 0.6018 - jaccard_coef: 0.3865 - loss: 0.9310 - val_accuracy: 0.5597 - val_jaccard_coef: 0.3571 - val_loss: 0.9322
Epoch 5/10
265/265 ━━━━━━━━━━━━━━━━━━━━ 682s 3s/step - accuracy: 0.5982 - jaccard_coef: 0.3915 - loss: 0.9257 - val_accuracy: 0.5598 - val_jaccard_coef: 0.3530 - val_loss: 0.9216
Epoch 6/10
265/265 ━━━━━━━━━━━━━━━━━━━━ 682s 3s/step - accuracy: 0.5999 - jaccard_coef: 0.3951 - loss: 0.9192 - val_accuracy: 0.5690 - val_jaccard_coef: 0.3719 - val_loss: 0.9219
Epoch 7/10
265/265 ━━━━━━━━━━━━━━━━━━━━ 676s 3s/step - accuracy: 0.6001 - jaccard_coef: 0.4011 - loss: 0.9146 - val_accuracy: 0.6075 - val_jaccard_coef: 0.4208 - val_loss: 0.9217
Epoch 8/10
265/265 ━━━━━━━━━━━━━━━━━━━━ 719s 3s/step - accuracy: 0.6038 - jaccard_coef: 0.4054 - loss: 0.9094 - val_accuracy: 0.6203 - val_jaccard_coef: 0.4281 - val_loss: 0.9078
Epoch 9/10
265/265 ━━━━━━━━━━━━━━━━━━━━ 687s 3s/step - accuracy: 0.6065 - jaccard_coef: 0.4120 - loss: 0.9038 - val_accuracy: 0.5733 - val_jaccard_coef: 0.3853 - val_loss: 0.9157
Epoch 10/10
265/265 ━━━━━━━━━━━━━━━━━━━━ 752s 3s/step - accuracy: 0.6040 - jaccard_coef: 0.4099 - loss: 0.9054 - val_accuracy: 0.5710 - val_jaccard_coef: 0.3725 - val_loss: 0.9054
```

Now we should train the pretrained encoder called mobilenetv2:

```
BACKBONE1 = 'mobilenetv2'

n_classes=21
# define model
model1 = sm.Unet(BACKBONE1, encoder_weights='imagenet', classes=n_classes, activation=activation)
model1.compile(optim, total_loss, metrics=metrics)
print(model1.summary())
```

At first the encoder is freezed and after training all layers are trainable inorder to fine-tune the model.

```
Epoch 1/10
265/265 ━━━━━━━━━━━━━━━━━━━━ 567s 2s/step - accuracy: 0.5399 - jaccard_coef: 0.2919 - loss: 0.9476 - val_accuracy: 0.5146 - val_jaccard_coef: 0.2572 - val_loss: 0.9410
Epoch 2/10
265/265 ━━━━━━━━━━━━━━━━━━━━ 513s 2s/step - accuracy: 0.6613 - jaccard_coef: 0.4250 - loss: 0.8907 - val_accuracy: 0.5910 - val_jaccard_coef: 0.3333 - val_loss: 0.8994
Epoch 3/10
265/265 ━━━━━━━━━━━━━━━━━━━━ 567s 2s/step - accuracy: 0.6402 - jaccard_coef: 0.4303 - loss: 0.8703 - val_accuracy: 0.5612 - val_jaccard_coef: 0.3457 - val_loss: 0.8784
Epoch 4/10
265/265 ━━━━━━━━━━━━━━━━━━━━ 512s 2s/step - accuracy: 0.6356 - jaccard_coef: 0.4401 - loss: 0.8560 - val_accuracy: 0.6282 - val_jaccard_coef: 0.4326 - val_loss: 0.8583
Epoch 5/10
265/265 ━━━━━━━━━━━━━━━━━━━━ 546s 2s/step - accuracy: 0.6478 - jaccard_coef: 0.4574 - loss: 0.8412 - val_accuracy: 0.7033 - val_jaccard_coef: 0.5205 - val_loss: 0.8355
Epoch 6/10
265/265 ━━━━━━━━━━━━━━━━━━━━ 515s 2s/step - accuracy: 0.6625 - jaccard_coef: 0.4750 - loss: 0.8243 - val_accuracy: 0.6352 - val_jaccard_coef: 0.4466 - val_loss: 0.8400
Epoch 7/10
265/265 ━━━━━━━━━━━━━━━━━━━━ 561s 2s/step - accuracy: 0.6682 - jaccard_coef: 0.4811 - loss: 0.8141 - val_accuracy: 0.6651 - val_jaccard_coef: 0.4762 - val_loss: 0.8336
Epoch 8/10
265/265 ━━━━━━━━━━━━━━━━━━━━ 560s 2s/step - accuracy: 0.6712 - jaccard_coef: 0.4860 - loss: 0.8076 - val_accuracy: 0.6451 - val_jaccard_coef: 0.4626 - val_loss: 0.8463
Epoch 9/10
265/265 ━━━━━━━━━━━━━━━━━━━━ 505s 2s/step - accuracy: 0.6779 - jaccard_coef: 0.4956 - loss: 0.7973 - val_accuracy: 0.6903 - val_jaccard_coef: 0.5074 - val_loss: 0.8120
Epoch 10/10
265/265 ━━━━━━━━━━━━━━━━━━━━ 505s 2s/step - accuracy: 0.6819 - jaccard_coef: 0.5002 - loss: 0.7945 - val_accuracy: 0.6695 - val_jaccard_coef: 0.4930 - val_loss: 0.8330
<keras.src.callbacks.history.History at 0x784b3c558340>
```

```python
for l in model1.layers:
    l.trainable = True
```

And the final result:

```
Epoch 1/5
265/265 ━━━━━━━━━━━━━━━━━━━━ 586s 2s/step - accuracy: 0.6522 - jaccard_coef: 0.4659 - loss: 0.8416 - val_accuracy: 0.5897 - val_jaccard_coef: 0.3996 - val_loss: 0.8129
Epoch 2/5
265/265 ━━━━━━━━━━━━━━━━━━━━ 519s 2s/step - accuracy: 0.7013 - jaccard_coef: 0.5197 - loss: 0.7628 - val_accuracy: 0.6630 - val_jaccard_coef: 0.4763 - val_loss: 0.7479
Epoch 3/5
265/265 ━━━━━━━━━━━━━━━━━━━━ 521s 2s/step - accuracy: 0.7279 - jaccard_coef: 0.5506 - loss: 0.7206 - val_accuracy: 0.7127 - val_jaccard_coef: 0.5328 - val_loss: 0.7016
Epoch 4/5
265/265 ━━━━━━━━━━━━━━━━━━━━ 560s 2s/step - accuracy: 0.7430 - jaccard_coef: 0.5693 - loss: 0.6935 - val_accuracy: 0.7411 - val_jaccard_coef: 0.5675 - val_loss: 0.6658
Epoch 5/5
265/265 ━━━━━━━━━━━━━━━━━━━━ 519s 2s/step - accuracy: 0.7498 - jaccard_coef: 0.5785 - loss: 0.6853 - val_accuracy: 0.7574 - val_jaccard_coef: 0.5888 - val_loss: 0.6537
<keras.src.callbacks.history.History at 0x784b23d42d70>
```

Q6:

Due to the internet and gpu problems, I couldn't load the dataset and run the cells. But here are the parts which I have completed and the explanations are commented in the code:

```python
# مرحله 4: آماده‌سازی مدل
# از مدل زو TensorFlow (Fast R-CNN می‌توانند به روشی مشابه استفاده شود).

# TODO: نوشتن تابع برای بارگذاری یک مدل Faster R-CNN پیش‌آموزش دیده
import tensorflow_hub as hub
def load_pretrained_model():
    """
    بارگذاری یک مدل Faster R-CNN پیش‌آموزش دیده از مدل زو TensorFlow.

    خروجی‌ها:
    - model: مدل Faster R-CNN پیش‌آموزش دیده.
    """
    # دانشجویان باید این قسمت را تکمیل کنند
    # URL to a pre-trained Faster R-CNN model in TensorFlow Hub
    model_url = "https://tfhub.dev/tensorflow/faster_rcnn/resnet50_v1_640x640/1"

    # Load the pre-trained model
    model = hub.load(model_url)
    return model

model = load_pretrained_model()
```

```python
# TODO: نوشتن تابع برای پیش‌پردازش تصویر برای استنتاج مدل
def preprocess_image(image_path):
    """
    پیش‌پردازش یک تصویر برای استنتاج مدل.

    ورودی‌ها:
    - image_path (str): مسیر فایل تصویر.

    خروجی‌ها:
    - input_image (آرایه numpy): تصویر پیش‌پردازش شده آماده برای ورودی مدل.
    - original_image (آرایه numpy): تصویر اصلی برای نمایش.
    """
    # دانشجویان باید این قسمت را تکمیل کنند
    # Read the image
    original_image = cv2.imread(image_path)
    if original_image is None:
        raise FileNotFoundError(f"Image not found at {image_path}")

    # Convert image to RGB (from BGR, which is default in OpenCV)
    original_image = cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB)

    # Resize the image to the size expected by the model (640x640 for the pre-trained Faster R-CNN model)
    input_image = cv2.resize(original_image, (640, 640))

    # Normalize the image: the pre-trained Faster R-CNN model expects pixel values in the range [0, 1]
    input_image = input_image.astype(np.float32) / 255.0

    return input_image, original_image
```

```python
# TODO: نوشتن تابع برای شناسایی اشیا
def detect_objects(model, image_path, threshold=0.5):
    """
    شناسایی اشیا در یک تصویر با استفاده از یک مدل پیش‌آموزش دیده.

    ورودی‌ها:
    - model: مدل پیش‌آموزش دیده.
    - image_path (str): مسیر فایل تصویر.
    - threshold (شناور): آستانه شناسایی.

    خروجی‌ها:
    - هیچکدام: تصویر با جعبه‌های مرزی شناسایی شده نمایش داده می‌شود.
    """
    # دانشجویان باید این قسمت را تکمیل کنند
    # Preprocess the image
    input_image, original_image = preprocess_image(image_path)

    # Convert the preprocessed image to a tensor and add a batch dimension
    input_tensor = tf.convert_to_tensor(input_image)
    input_tensor = input_tensor[tf.newaxis, ...]

    # Perform inference
    detections = model(input_tensor)

    # Extract detection results
    detection_boxes = detections['detection_boxes'][0].numpy()
    detection_scores = detections['detection_scores'][0].numpy()
    detection_classes = detections['detection_classes'][0].numpy().astype(int)

    # Get the height and width of the original image
    height, width, _ = original_image.shape

    # Draw bounding boxes and labels on the original image
    for i in range(len(detection_boxes)):
        if detection_scores[i] >= threshold:
            box = detection_boxes[i] * [height, width, height, width]
            box = box.astype(int)
            class_id = detection_classes[i]
            score = detection_scores[i]

            # Draw bounding box
            cv2.rectangle(original_image, (box[1], box[0]), (box[3], box[2]), (0, 255, 0), 2)

            # Draw label and score
            label = f"Class {class_id}: {score:.2f}"
            cv2.putText(original_image, label, (box[1], box[0] - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

    # Display the image with bounding boxes
    plt.imshow(original_image)
    plt.axis('off')
    plt.show()
```

```python
def compute_iou(box1, box2):
    ymin1, xmin1, ymax1, xmax1 = box1
    ymin2, xmin2, ymax2, xmax2 = box2

    inter_ymin = max(ymin1, ymin2)
    inter_xmin = max(xmin1, xmin2)
    inter_ymax = min(ymax1, ymax2)
    inter_xmax = min(xmax1, xmax2)

    inter_area = max(0, inter_ymax - inter_ymin) * max(0, inter_xmax - inter_xmin)
    box1_area = (ymax1 - ymin1) * (xmax1 - xmin1)
    box2_area = (ymax2 - ymin2) * (xmax2 - xmin2)
    union_area = box1_area + box2_area - inter_area

    iou = inter_area / union_area if union_area != 0 else 0
    return iou
```

```python
# TODO:  نوشتن تابع ارزیابی مدل با استفاده از معیار های خواسته شده
def evaluate_model(model, dataset, iou_threshold=0.5):
    """
    ارزیابی عملکرد مدل بر روی یک مجموعه اعتبارسنجی.

    ورودی‌ها:
    - model: مدل آموزش دیده.
    - dataset: مجموعه داده اعتبارسنجی.

    خروجی‌ها:
    - metrics: دیکشنری حاوی معیارهای ارزیابی مانند میانگین دقت متوسط (mAP).
    """
    # دانشجویان باید این قسمت را تکمیل کنند
    all_detections = []
    all_annotations = []
    for image_path, ground_truth_boxes, ground_truth_labels in tqdm(dataset):
        input_image, _ = preprocess_image(image_path)
        input_tensor = tf.convert_to_tensor(input_image)
        input_tensor = input_tensor[tf.newaxis, ...]

        detections = model(input_tensor)

        detection_boxes = detections['detection_boxes'][0].numpy()
        detection_scores = detections['detection_scores'][0].numpy()
        detection_classes = detections['detection_classes'][0].numpy().astype(int)

        filtered_boxes = []
        filtered_scores = []
        filtered_classes = []
        for i in range(len(detection_scores)):
            if detection_scores[i] >= iou_threshold:
                filtered_boxes.append(detection_boxes[i])
                filtered_scores.append(detection_scores[i])
                filtered_classes.append(detection_classes[i])

        all_detections.append((filtered_boxes, filtered_scores, filtered_classes))
        all_annotations.append((ground_truth_boxes, ground_truth_labels))

    mAP = compute_map(all_detections, all_annotations, iou_threshold)
    return {'mAP': mAP}
```

```python
def compute_map(detections, annotations, iou_threshold):
    average_precisions = []
    for class_id in set([cls for det in detections for cls in det[2]]):
        true_positives = []
        scores = []
        num_annotations = 0
        for i in range(len(detections)):
            detected_boxes, detected_scores, detected_classes = detections[i]
            gt_boxes, gt_labels = annotations[i]
            gt_boxes = [box for j, box in enumerate(gt_boxes) if gt_labels[j] == class_id]
            detected_boxes = [box for j, box in enumerate(detected_boxes) if detected_classes[j] == class_id]
            detected_scores = [score for j, score in enumerate(detected_scores) if detected_classes[j] == class_id]
            num_annotations += len(gt_boxes)

            if len(gt_boxes) == 0:
                true_positives.extend([0] * len(detected_boxes))
                scores.extend(detected_scores)
                continue

            detected_boxes = np.array(detected_boxes)
            gt_boxes = np.array(gt_boxes)
            scores.extend(detected_scores)

            for d, detected_box in enumerate(detected_boxes):
                ious = [compute_iou(detected_box, gt_box) for gt_box in gt_boxes]
                max_iou = max(ious) if ious else 0
                if max_iou >= iou_threshold:
                    true_positives.append(1)
                else:
                    true_positives.append(0)

        if num_annotations == 0:
            average_precisions.append(0)
            continue

        sorted_indices = np.argsort(-np.array(scores))
        true_positives = np.array(true_positives)[sorted_indices]
        false_positives = 1 - true_positives

        cum_true_positives = np.cumsum(true_positives)
        cum_false_positives = np.cumsum(false_positives)
        precision = cum_true_positives / (cum_true_positives + cum_false_positives)
        recall = cum_true_positives / num_annotations

        precision = np.concatenate([[0], precision, [0]])
        recall = np.concatenate([[0], recall, [1]])

        for i in range(len(precision) - 1, 0, -1):
            precision[i - 1] = np.maximum(precision[i - 1], precision[i])

        indices = np.where(recall[1:] != recall[:-1])[0]
        average_precision = np.sum((recall[indices + 1] - recall[indices]) * precision[indices + 1])
        average_precisions.append(average_precision)

    mAP = np.mean(average_precisions)
    return mAP
```