Nankai University
College of Software Engineer

Final Project
of
Computer Vision

Object Detection: Leveraging Pre-Trained EfficientDet Models for Real-Time Deployment with Gradio

---

Students :
Somoeurn Virakden 2120246050
Uchita Hikaru 2120246057

Professor :
WANG Jing

Contents

1 Abstract

This project develops a real-time object detection system leveraging EfficientDet models from TensorFlow Hub which trained with the COCO 2017 dataset. Our implementation focuses on EfficientDet-D4, balancing performance and efficiency for practical applications. We integrate EfficientDet with TensorFlow-Hub and deploy it using Gradio for interactive object detection. The system demonstrates high accuracy and real-time potential. Our approach showcases the effectiveness of EfficientDet architecture in achieving state-of-the-art performance while maintaining reasonable computational requirements, making it suitable for various real-time object detection scenarios.

2 Introduction

In recent years, object detection has emerged as a crucial technology within the field of computer vision. Its ability to identify, classify, and localize objects within images or videos has made it indispensable for applications ranging from autonomous vehicles to healthcare diagnostics, surveillance systems, and retail automation. Object detection bridges the gap between raw visual data and actionable insights, enabling machines to understand and interact with the physical world effectively.

This project explores the implementation of an object detection system using EfficientDet-D4, a state-of-the-art object detection model, in combination with TensorFlow Hub for pre-trained weights and Gradio for deployment. EfficientDet-D4, part of the EfficientDet model family, is well-known for its high accuracy and computational efficiency. It leverages a compound scaling approach and integrates EfficientNet as its backbone along with a Bidirectional Feature Pyramid Network (BiFPN) for multi-scale feature fusion. These innovations enable EfficientDet to detect objects of varying sizes and complexities with remarkable efficiency.

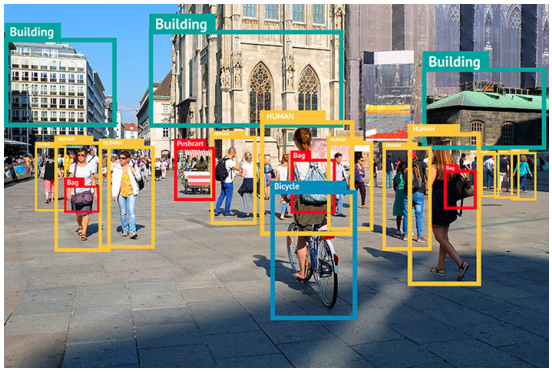

Figure 1: Sample Object Detection

The model uses the COCO dataset (Common Objects in Context), a large-scale dataset containing images of everyday objects in complex scenes. The COCO dataset enables the model to recognize and classify 90 different object categories, including people, animals, vehicles, and household items. By leveraging TensorFlow Hub, the project utilizes pre-trained EfficientDet-D4 weights, eliminating the need for extensive model training while maintaining robust performance. The system is deployed using Gradio, an intuitive Python library that allows users to interact with the object detection system through an easy-to-use web interface.

By the end of this project, we aim to provide an efficient and scalable solution for object detection that can be extended to various domains and use cases.

3 EfficientDet Architecture

EfficientDet is a family of scalable object detection models that are highly efficient in terms of both computation and accuracy. Its architecture builds on three key components:

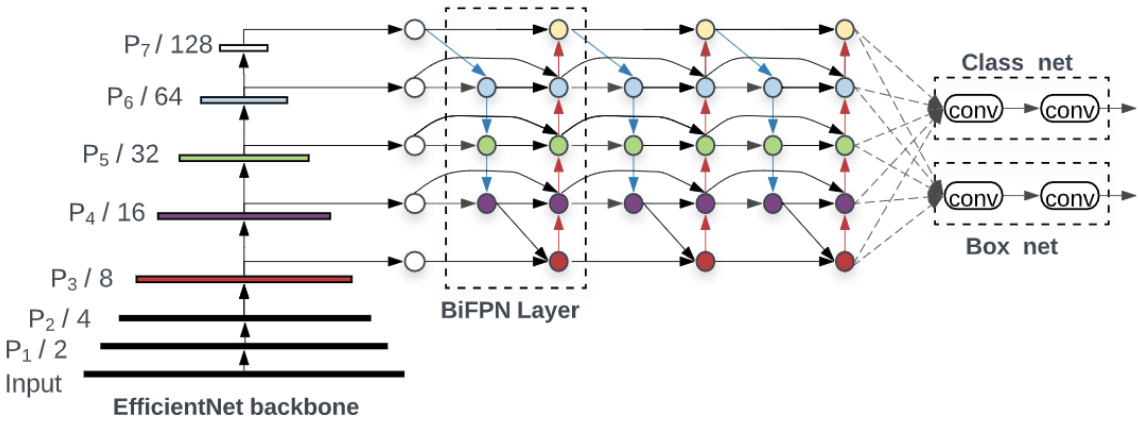

Figure 2: EfficientDet Architecture

3.1 Backbone: EfficientNet

- **EfficientNet** is used as the feature extraction backbone, leveraging its high efficiency for image classification.
- Pretrained on ImageNet, EfficientNet provides multi-scale features that serve as inputs to the subsequent BiFPN.
- Feature levels extracted from EfficientNet are $\{P_3, P_4, P_5, P_6, P_7\}$ corresponding to down sampling rates of $1/8, 1/16, 1/32, 1/64$ and $1/128$ of the input resolution, respectively.

3.2 BiFPN

- **BiFPN** fuses features from different scales using bi-directional connections. It improves upon standard FPNs by:
  - Adding **bi-directional pathways**: A top-down pathway aggregates higher-level semantic features, and a bottom-up pathway captures finer details.
  - Introducing **learnable weights**: Instead of equal contributions, BiFPN learns the relative importance of features from different resolutions.
  - **Efficient design**: Unnecessary nodes and connections are removed, making BiFPN lightweight and computationally efficient.
- Repeated BiFPN layers further refine the multi-scale features, enhancing detection performance while minimizing overhead.

- **Depthwise separable convolutions**: These are used throughout BiFPN to reduce computation without sacrificing accuracy.

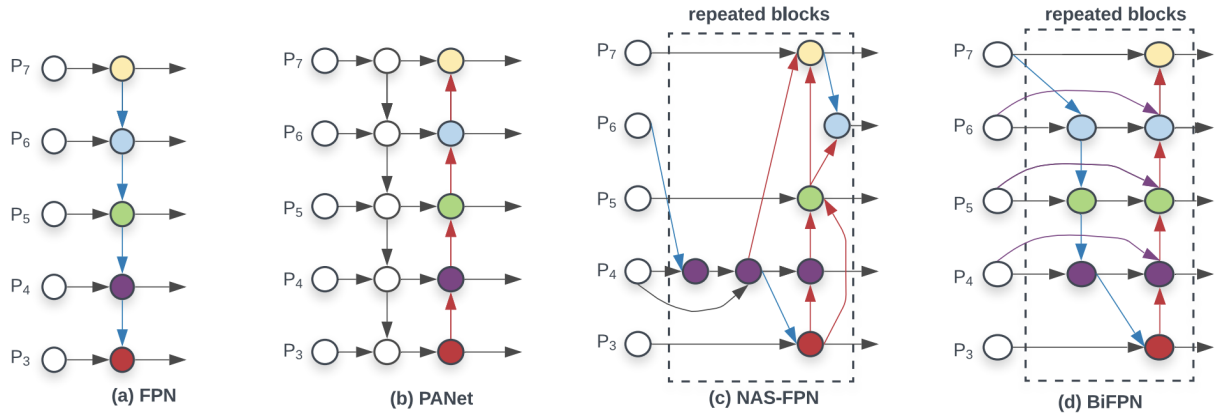

Figure 3: Feature Network Design

3.3 Prediction Network

- The final stage of the architecture consists of two lightweight sub-networks:
  1. **Class Prediction Network**: Predicts object classes for each bounding box.
  2. **Box Prediction Network**: Predicts bounding box coordinates.
- Both networks share parameters across feature levels to reduce the model size.
- These networks are repeated multiple times (depending on the model scale, such as D0-D7) for improved performace.

4 Implementation

The implementation of the object detection system involved the following key steps:

4.1 Model Loading

We utilized TensorFlow Hub to load the pre-trained EfficientDet-D4 model. TensorFlow Hub provides a convenient way to access pre-trained models, allowing us to integrate them seamlessly into our project. By using pre-trained weights, we were able to skip the time-consuming process of training the model from scratch.

```
EfficientDet  = {'EfficientDet D0 512x512'   :
↪  'https://tfhub.dev/tensorflow/efficientdet/d0/1',
                 'EfficientDet D1 640x640'   :
                 ↪  'https://tfhub.dev/tensorflow/efficientdet/d1/1',
                 'EfficientDet D2 768x768'   :
                 ↪  'https://tfhub.dev/tensorflow/efficientdet/d2/1',
                 'EfficientDet D3 896x896'   :
                 ↪  'https://tfhub.dev/tensorflow/efficientdet/d3/1',
                 'EfficientDet D4 1024x1024' :
                 ↪  'https://tfhub.dev/tensorflow/efficientdet/d4/1',
                 'EfficientDet D5 1280x1280' :
                 ↪  'https://tfhub.dev/tensorflow/efficientdet/d5/1',
                 'EfficientDet D6 1280x1280' :
                 ↪  'https://tfhub.dev/tensorflow/efficientdet/d6/1',
                 'EfficientDet D7 1536x1536' :
                 ↪  'https://tfhub.dev/tensorflow/efficientdet/d7/1'
                }
```

```
    model_url = EfficientDet['EfficientDet D4 1024x1024' ]
    od_model = hub.load(model_url)
```

4.2 Model Usage

The model is straightforward to operate. Simply feed an image into the loaded model.

```
        result = od_model(image)
```

4.2.1 Model Input

A three-channel image of variable size - the model does NOT support batching.

```
        image = np.asarray(image)
        # Add a batch dimension which is required by the model.
        image = np.expand_dims(image, axis=0)
```

4.2.2 Model Output

The model returns an output dictionary which contains

- num_detections: contains the number of detections [N].
- detection_boxes: contains bounding box coordinates in the following order: [ymin, xmin, ymax, xmax].
- detection_classes: contains detection class index from the label file.
- detection_scores: contains detection scores.
- raw_detection_boxes: contains decoded detection boxes without Non-Max suppression. M is the number of raw detections.

- raw_detection_scores: contains class score logits for raw detection boxes. M is the number of raw detections.
- detection_anchor_indices: contains the anchor indices of the detections after NMS.
- detection_multiclass_scores: contains class score distribution (including background) for detection boxes in the image including background class.

4.2.3 Post Processing

The figure 4 represents the main steps in the process detection function, which processes image detections and annotates them on the original image. The diagram outlines the flow of activities from initial data conversion to final image annotation.

- **Convert Result**: The process begins by converting the results dictionary to numpy arrays.
- **Extract Data**: Detection scores, boxes, and classes are extracted from the converted results.
- **Extract Data**: Detections are filtered based on the threshold, selecting only those above the minimum score
- **Image Annotation Loop**: The loop iterates through the filtered detections:
  - Converts normalized coordinates to pixel coordinates
  - Annotates the image with a bounding box
  - Adds text displaying the class name and confidence score
- **Finalization**: The process ends after all detections have been annotated.

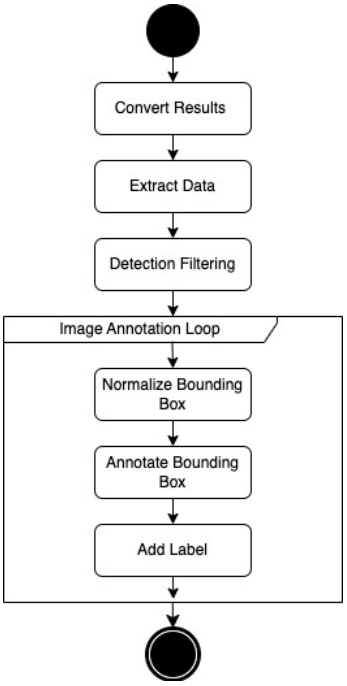

Figure 4: Post-processing Diagram

4.3 Gradio Integration

The system was deployed using Gradio, which allowed users to upload images and interact with the object detection model through a web-based interface. Gradio simplified the deployment process and provided an easy-to-use platform for end-users.

```
    gr.close_all()
    demo = gr.Interface(fn=detect,
            inputs=[gr.Image(label="update image", type="pil"),
                    gr.Slider(1, 100, value=30, label="Minimum Detection
                    ↪  Threshold ")],
            outputs=[gr.Image(label="result", type="pil")], title="Object
            ↪  Detection",
            description="Object Detection with EfficientDets Model",
             allow_flagging="never",)
    demo.launch(share=True)
```

The Gradio interface above enables users to perform object detection on images using the EfficientDet model. The interface takes two inputs: an image, which is uploaded by the user, and a slider for adjusting the "Minimum Detection Threshold," ranging from 1 to 100. This threshold value determines the sensitivity of the detection, with a higher value making the model more selective about what constitutes a detected object. The output of the interface is the processed image, displayed after the detection process is applied. The interface is designed with user-friendliness in mind, featuring clear labels and a description, and allows sharing via a public link. The gr.close_all() function ensures that any previously opened interfaces are closed before launching the new one, preventing conflicts. This setup provides a convenient way for users to interact with the model and adjust the detection parameters dynamically.

References

1. Object Detection https://github.com/virakden/object-detection-with-gradio.git. Accessed: 26/10/2024

2. Original Source https://learnopencv.com/object-detection-tensorflow-hub/

3. **EfficientDet: Scalable and Efficient Object Detection** *author*, Mingxing Tan, Ruoming Pang, Quoc V.Le Google Research, Brain Team.