NN Object Detection Project

Nili Alfia 314880873, Omri Drori 207921719

March 2024

GitHub:
`https://github.com/niliya96/Project_NN_object_detection`
A link to the video can be found at this repository, testing.mp4 file.

1 Introduction

The integration of neural networks in object detection is a very important aspect of AI, especially in computer vision tasks. The goal is to simplify the understanding and practical implementation of neural network concepts, ranging from basic classification to complex object detection tasks
In this project we will focus on implementation of object detection system with a classification model as the foundational backbone.

2 Report the Classification Model Backbone

2.1 Backbone Architecture Selection

We need to aggregate the ID's of both of us (aggregate the digits), and use the last digit to decide which backbone architecture to take. For the sum we wrote a script (The script is added to the project GitHub).
The idea of the code is to calculate the sum of the digits of the sum of all numbers in the numbers list, and then reduces this sum to a single-digit number. This process involves using helper functions to handle digit summation and reduction. $(sum_digits, reduce_to_single_digit, aggregate_digits)$
We got an aggregation sum of 8, so we chose MobileNet V3 architecture.

2.2 Current model capabilities

In this section we want to test what is the current capabilities of the model, before the training.
We decide to test the model on the same movies that we will test it after it will be trained. The links to the movies before training can be found in the GitHub repository of the project.

For this part, we used the same model, but we initialize it's weights randomly and call to the $inference_video.py$ file to create the video with the results.
How did we randomize the weights
For running the inference on the not trained model, we need to initialize it's weights randomly. For that we use exactly the same model, but before we used it, we call to an initialization method.
we will explain how this method works and why it's randomize the weights.
The $randomized_model.py$ file contains initializes the weights randomly using specific methods that are suitable for different types of layers within a neural network.There are several reasons why this specific function implements randomized weight initialization rather than using fixed or deterministic values for weights:

1. For Convolutional Layers: The function uses $torch.nn.init.kaiming_uniform$ to initialize the weights of Conv2d layers. This choice is motivated by the Kaiming initialization method, which is designed for rectified linear units (ReLUs) commonly used in deep neural networks. Kaiming initialization initializes weights to random values sampled from a uniform distribution, scaled based on the fan-in (number of input units) to help prevent vanishing or exploding gradients.

2. For BatchNorm2d layers, the function uses $torch.nn.init.uniform$ to initialize the weights randomly sampled from a uniform distribution. This random initialization helps in introducing diversity in scaling factors applied to normalize activations within each batch.

3. For Bias in addition to weight initialization, the function sets biases to zero using 4torch.nn.init.constant. This practice is common in neural network initialization to ensure that biases start with neutral values, preventing any initial bias towards specific outputs and encouraging unbiased learning.

Current Model Capabilities (before train)
The current model has randomized weights. We decided to test it on a video with cars (our dataset). The model didn't work well before training on a video file. On different runs it sometimes recognize a vehicle for a frame or few, but in general the backbone before training is limited.

2.3 Analysis MobileNet V3 SSD

In this section we Will analyze the chosen architecture.
MobileNetV3 SSD is a deep learning model designed for object detection tasks in computer vision, particularly developed for deployment on mobile and embedded devices. It combines the efficiency of MobileNetV3, a lightweight [1] neural network architecture, with the accuracy and versatility of Single Shot Multibox Detector (SSD), a popular object detection framework. MobileNetV3 SSD's architecture typically includes a backbone network based on MobileNetV3 for

feature extraction, followed by additional layers specific to the SSD framework for object detection. These additional layers include multiple convolutional layers responsible for predicting bounding boxes, object classes, and confidence scores. The integration of MobileNetV3 with SSD allows for end-to-end training and inference, enabling seamless deployment of the model on various devices for applications such as object recognition, scene understanding, and augmented reality.

MobileNet V3
MobileNetV3 is built upon the success of its predecessors, MobileNetV1 and MobileNetV2, by introducing novel architectural elements aimed at improving performance and efficiency. It follows a streamlined architecture composed of depthwise separable convolutions, inverted residuals, and linear bottlenecks.
The backbone of MobileNetV3 relies heavily on depthwise separable convolutions. These convolutions consist of two distinct operations: depthwise convolutions and pointwise convolutions. Depthwise convolutions apply a single filter per input channel, reducing computational cost by factorizing spatial and channel-wise operations. Pointwise convolutions then project the output of depthwise convolutions into a higher-dimensional space, enabling non-linear transformations.
Interesting aspect of this architecture includes the incorporation of linear bottlenecks [2] within inverted residuals [3]. This design choice improves information flow and gradient propagation, leading to more stable training and better feature learning capabilities.
Another interesting aspect is that MobileNetV3 intruduces squeeze-and-excitation (SE). SE blocks are used to recalibrate channel-wise feature responses adaptively, emphasizing informative features and suppressing less relevant ones. This attention mechanism enhances the discriminative power of the network, leading to improved performance, especially in tasks requiring fine-grained feature discrimination.

A little about the architecture model performance. The MobileNetV3 backbone designed to balance efficiency and performance. By leveraging depthwise separable convolutions, inverted residuals with linear bottlenecks, and SE blocks, MobileNetV3 achieves a favorable trade-off between model size, computational complexity, and accuracy. This makes it well-suited for deployment on resource-constrained devices, such as mobile phones, edge devices, and IoT devices, without compromising on detection performance.

3 SSD

In object detection SSD means Single Shot MultiBox Detector. It is a method for detecting objects in images using a single deep neural network. SSD achieves high accuracy and real-time processing by combining predictions from multiple

feature maps with different resolutions. This approach allows SSD to detect objects of various sizes effectively. It uses a backbone, such as MobileNet, for feature extraction, and adds several convolutional layers to predict the presence of objects and their bounding boxes.
The other options for object detections are Faster R-CNN, YOLO (You Only Look Once), and R-FCN (Region-based Fully Convolutional Networks). Each has its unique approach to object detection, balancing between accuracy and speed. We chose to use SSD because it provides high detection accuracy and efficiency, crucial for real-time applications. Unlike other object detection models that may require separate proposals and detection networks, SSD does so in a single pass. This reduces complexity and improves speed. SSD's multi-scale feature map approach for detection enables it to capture objects at various sizes more effectively than some alternatives, providing robustness across different vehicle types and sizes. Its straightforward architecture also simplifies training and implementation, making it a practical choice for our project focused on vehicle detection.

4 Integration ssd with mobilenetv3 backbone

The adaptation of MobileNetV3 for use in the SSD framework involves the removal of its final classification layer. This modification transforms MobileNetV3 into a dedicated feature extractor that outputs a series of feature maps. These maps, with their varied resolutions, are pivotal for the subsequent detection of objects of differing sizes and proportions. The output channels across these feature maps are uniformly configured to ensure a consistent depth, facilitating the uniform application of the SSD detection mechanisms.
To detect vehicles of various dimensions and orientations, a Default Box Generator, also known as an anchor generator, creates multiple bounding boxes of specified scales and aspect ratios. These bounding boxes are then applied across the different feature maps, enabling the model to adaptively detect vehicles by matching these predefined boxes to the detected features within the images.
The SSDHead, comprising additional convolutional layers, is applied to these feature maps for two main purposes: to classify the presence of objects within the anchors and to adjust these anchors for a better fit around the detected objects. This process involves determining the object class and refining the bounding box positions, thereby improving the accuracy of the detections.
Finally, the SSD model is assembled with the backbone, anchor generator, and SSD head. This assembled model is capable of detecting vehicles by effectively classifying and adjusting the predefined anchors based on the extracted features from the images. A Non-Maximum Suppression (NMS) threshold of 0.45 is set to eliminate redundant detections, ensuring that only the most confident detections are retained.

Our SSD (Single Shot MultiBox Detector) model's object detection capability is enhanced through precise localization, achieved by predicting offsets from

predefined default boxes. These offsets adjust the center coordinates (c_x, c_y) and dimensions (width w and height h) relative to each default box d, to closely align with the ground truth box g.

5 Bounding Box Predictions

The formula for calculating these offsets is given by:

$$\Delta x = \frac{(g_j^{c_x} - d_i^{c_x})}{d_i^w}, \quad \Delta y = \frac{(g_j^{c_y} - d_i^{c_y})}{d_i^h},$$
$$\Delta w = \log\left(\frac{g_j^w}{d_i^w}\right), \quad \Delta h = \log\left(\frac{g_j^h}{d_i^h}\right),$$

where Δx and Δy denote center offsets and Δw and Δh represent size offsets. In our PyTorch implementation, several key components contribute to this process:

Default Box Generator: The `DefaultBoxGenerator` function specifies the aspect ratios and scales for the default boxes, crucial for the initial bounding box predictions before offset adjustments.

SSD Head: The `SSDHead` component computes the offsets (Δx, Δy, Δw, Δh) based on the feature maps from the backbone and the parameters from the default boxes.

6 The Loss Function for Single Shot Detection (SSD)

The SSD model utilizes a dual-component loss function during training, integrating both location predictions and confidence scores. This function comprises two main elements: the localization loss and the confidence loss.

6.1 Localization Loss

The localization loss is calculated using the smooth L1 loss, formulated as:

$$L_{\text{loc}}(x, l, g) = \sum_{i \in \text{Pos}}^{N} \sum_{m \in \{cx, cy, w, h\}} x_{ij}^k \text{smooth}_{L1}\left(l_i^m - \hat{g}_j^m\right),$$

where l represents the predicted bounding box, and g denotes the ground truth box. The term x_{ij}^k is an indicator for matching the i-th default box to the j-th ground truth box of category k.

6.2 Confidence Loss

The confidence loss utilizes softmax loss across multiple class confidences c, defined as:

$$L_{\text{conf}}(x, c) = -\sum_{i \in \text{Pos}}^{N} x_{ij}^{p} \log\left(\hat{c}_i^p\right) - \sum_{i \in \text{Neg}} \log\left(\hat{c}_i^0\right),$$

with $\hat{c}_i^p$ being the softmax probability of the i-th prediction belonging to class p. This formula ensures that the model is penalized for incorrect class predictions, encouraging accurate classification.

6.3 Combined Loss Function

The final loss function amalgamates the localization and confidence losses:

$$L(x, c, l, g) = \frac{1}{N}\left(L_{\text{conf}}(x, c) + \alpha L_{\text{loc}}(x, l, g)\right),$$

where N signifies the count of matched default boxes, and α is a balancing coefficient for the localization loss.
This combination enables the SSD model to optimize for both precise object localization and accurate class identification, crucial for effective object detection.

7 metrics

The mAP metric serves as a key evaluation tool, encapsulating the model's accuracy in detecting vehicles across different scenarios. Specifically, mAP quantifies both the precision (correctness of detected vehicles) and recall (model's ability to detect all vehicles) at various threshold levels, providing a holistic view of the model's performance.
In our implementation, the validation process leverages the `torchmetrics.detection.mean_ap.MeanAveragePr` class to compute the mAP scores. This process involves iterating over the validation dataset, generating predictions for each batch, and comparing these predictions against the ground truth annotations. The code snippet below illustrates this workflow:

```
# Example of mAP calculation in our validation process
metric = MeanAveragePrecision()
metric.update(predictions, target)
metrics_list = metric.compute()
```

Two specific mAP metrics are reported:

- **mAP_50:** This is the mAP calculated at the Intersection Over Union (IoU) threshold of 0.5, which indicates the model's performance in detecting vehicles with a moderate overlap with the ground truth.

- **mAP_50_95:** This represents the average mAP calculated over IoU thresholds from 0.5 to 0.95 (inclusive), in steps of 0.05. This metric provides a more comprehensive assessment of the model's detection accuracy at varying levels of precision.

8 Augmentations

In the project, a comprehensive set of image augmentations was applied to the training data to improve the robustness and performance of the vehicle detection model. These augmentations introduce a variety of realistic modifications to the images, enabling the model to learn from a more diverse set of examples and thus generalize better to unseen data.
HorizontalFlip with a probability of 0.5 randomly mirrors images along the vertical axis. This augmentation helps the model become invariant to the orientation of vehicles, enhancing its ability to detect vehicles regardless of their facing direction.
Blur, MotionBlur, and MedianBlur each with a limit of 3 and a probability of 0.1, apply different types of blurring effects to the images. These augmentations mimic real-world scenarios where images may not always be sharp due to factors like camera movement, focus issues, or atmospheric conditions. Training the model with these variations helps improve its performance under less-than-ideal imaging conditions.
ToGray with a probability of 0.3 converts color images to grayscale. This augmentation challenges the model to rely less on color information and more on structural and shape features for vehicle detection. It's particularly useful for scenarios where color may not be a reliable indicator due to lighting conditions or vehicle color variability.
RandomBrightnessContrast and ColorJitter each with a probability of 0.3, and RandomGamma with the same probability, introduce variations in image brightness, contrast, color, and gamma levels. These augmentations simulate different lighting conditions and camera settings, ensuring that the model is less sensitive to such variations and can reliably detect vehicles across a range of illumination scenarios.

9 experiments

9.1 Optimizations parameters

The provided graphs in 1 and 2 depict the performance metrics for two different optimization algorithms—Adam and Stochastic Gradient Descent (SGD) with Nesterov momentum. The performance is evaluated in terms of the loss and the mean Average Precision (mAP) across epochs.
For the loss, as shown in the first image, the Adam optimizer demonstrates a consistently lower loss value compared to SGD with Nesterov momentum, indicating that Adam is leading to a more optimal solution.

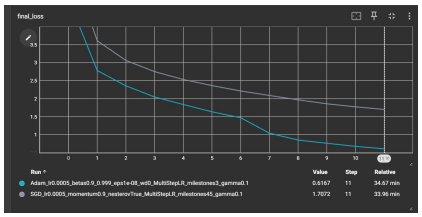

Figure 1: First Image

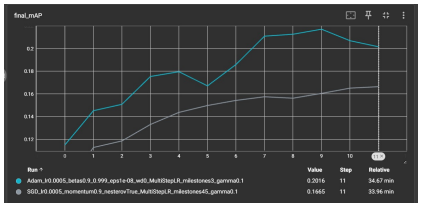

Figure 2: Second Image

The second image shows the mAP, a common metric for object detection models that measures the accuracy considering both precision and recall across all classes.Also Here, the Adam optimizer outperforms SGD with Nesterov momentum for most of the training duration, although there is a noticeable dip in performance around step 5. Despite this, Adam recovers and achieves a higher mAP by step 11.
In conclusion, the Adam optimizer yields better results in both final loss and mAP compared to SGD with Nesterov momentum. These findings suggest that Adam may be more suitable for this problem, possibly due to its adaptive learning rate capabilities that account for the sparse gradients.

9.2 Anchor Boxes configurations

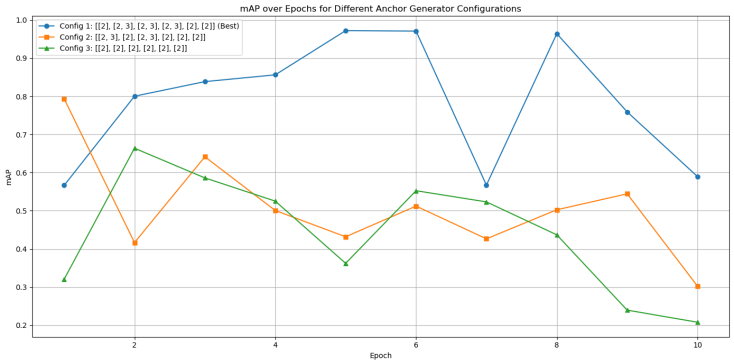

Figure 3: Anchor Boxes Configuration Experiment

Anchor boxes are predefined bounding boxes of various ratios and scales that serve as reference points for the object detection model to predict the location of objects. During training, the model learns to adjust these anchor boxes to match the actual objects in the input images.
The graph in 3 displays the mAP over training epochs for three distinct anchor configurations.

Config 1 shows a general upward trend in mAP, with minor fluctuations, achieving the highest value at epoch 8. This indicates that the anchor box sizes and aspect ratios in Config 1 are likely well-tuned to the dimensions of the target objects, leading to effective detection performance.
Config 2 and Config 3 Finished much less then config 1. This suggest they might be less effective configurations.
In summary, Config 1 offers the most reliable and highest mAP throughout training, indicating it's the optimal anchor generator configuration for this object detection task. Configurations 2 and 3 seem to be less aligned with the object characteristics in the dataset and might require adjustments for better performance.

9.3 Note on the experiments

Because the data was very large, and we didn't have GPU, we couldn't make a lot of experiments. Whats more, we reached reasonable results after those experiments so we stopped there.

10 Dataset

The dataset we chose for this object detection system can be found at:
`https://universe.roboflow.com/tarslab-z5tlh/vehicle-detection-cyp5j`
This dataset includes 19,267 images of vehicles, bounded by boxes with fit configuration. Classes that can be found at this datasets are:

```
CLASSES = ['shared-taxi', 'truck-5-axle', 'auto', 'mav', 'private-bus',
'taxi', 'cycle', 'truck-4-axle', 'mini-lcv', 'lcv', 'goods-auto', 'heavy-vehicle',
'govt-bus', 'BIKE', 'truck-3-axle', 'school-bus', 'tractor', 'two-wheeler',
'mini-bus', 'pedestrian', 'car', 'truck-2-axle']
```

The dataset is split to train valid and test in the following amounts:
train - 9763 images (0.5 of the data)
valid - 7577 images (0.4 of the data)
test - 1927 images (0.1 of the data)

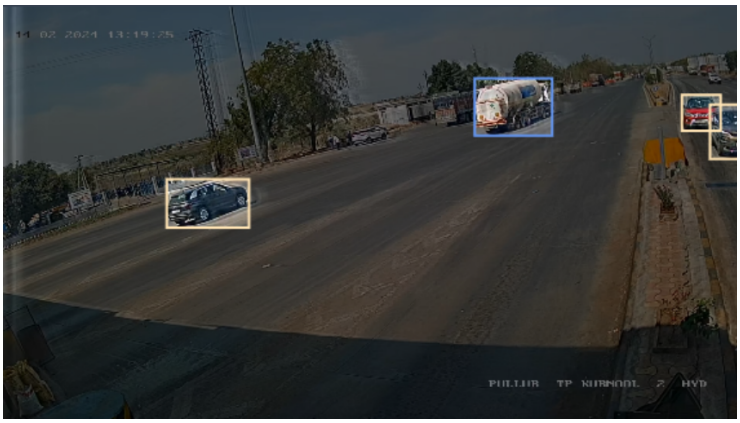

Figure 4: Example from dataset (train)

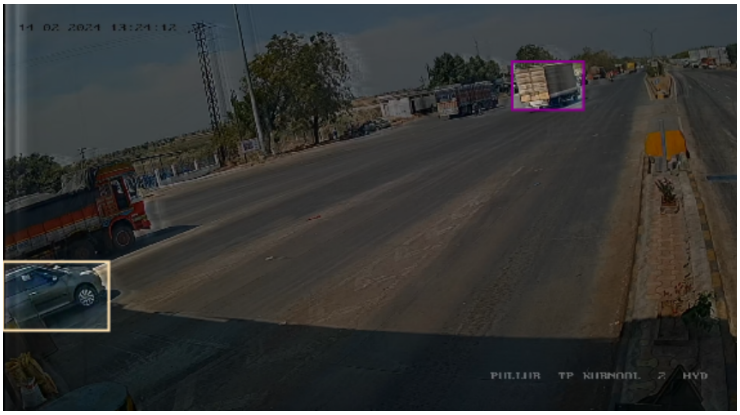

Figure 5: Example from dataset (valid)

Figure 6: Example from dataset (test)

The amount of the train set helps the model to learn diverse object labels and bbox configurations. A sufficient amount of valid set helps to check the model's generalization on unseen data and detect multiple labels and AABB correctly.

Comments

[1] A lightweight model refers to a type of machine learning or deep learning model that is designed to be efficient in terms of computational resources, memory usage, and model size while still maintaining acceptable levels of performance for a given task. The term "lightweight" typically implies that the model is optimized to run on resource-constrained devices such as mobile phones, IoT devices, and edge computing platforms.

[2] Linear bottlenecks replace traditional non-linear activation functions (e.g., ReLU) with linear activations.

[3] Inverted residual blocks, is an architecture that enhances feature representation while minimizing computational overhead. Inverted residuals consist of a lightweight bottleneck layer followed by expansion and projection layers. The bottleneck layer reduces the number of channels to a lower-dimensional space, while the expansion and projection layers adjust the feature map dimensions to facilitate richer feature extraction.

References

[1] `https://paperswithcode.com/method/mobilenetv3`
[2] `https://arxiv.org/abs/1905.02244`
[3] `https://openaccess.thecvf.com/content_ICCV_2019/papers/Howard_Searching_for_MobileNetV3_ICCV_2019_paper.pdf`

[4] https://www.youtube.com/watch
?v=O7mQpJnB-cw&t=67s&ab_channel=MaziarRaissi
[5] https://www.researchgate.net/ publication/339561485_Searching_for_MobileNetV3