МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования
«**Московский технический университет связи и информатики**»
(**МТУСИ**)

Кафедра «Информационные технологии и программирование»

Отчет по проектному практикуму

Выполнил: студенты 2 курса ИТ
направления 09.03.01
«Информатика и вычислительная техника»
студент гр. БВТ2203
Кудрявцев В.Д.
Недопекин М.А.
Мухин Р.А.

Москва, 2023 г.

Содержание

1 Введение

1.1 Что такое YOLO

YOLO, что расшифровывается как ”You Only Look Once”(Ты смотришь только один раз), представляет собой популярный алгоритм для обнаружения объектов в изображениях и видео. Этот метод был предложен в статье под названием ”You Only Look Once: Unified, Real-Time Object Detection”(2015) Йозефом Редмоном, Сантом Диввали, Россом Гиршиком, Али Фархади. YOLO отличается от других методов обнаружения объектов тем, что он осуществляет прогнозирование всех объектов в одном проходе, что делает его быстрым и эффективным для реального времени.

1.2 Как работает YOLOv1

Изображение разбивается на сетку размером $S \cdot S$. Каждая ячейка сетки предсказывает B баудинг боксов. Каждый баудинг бокс содержит 5 предсказаний: x (центр по x),y (центр по y),w (ширина),h (высота), вероятность того, что в данном баудинг боксе содержится объект заданного класса. Также для каждой ячейки сетки алгоритм предсказывает условные вероятности C классов, $Pr(Class_i|Object)$.

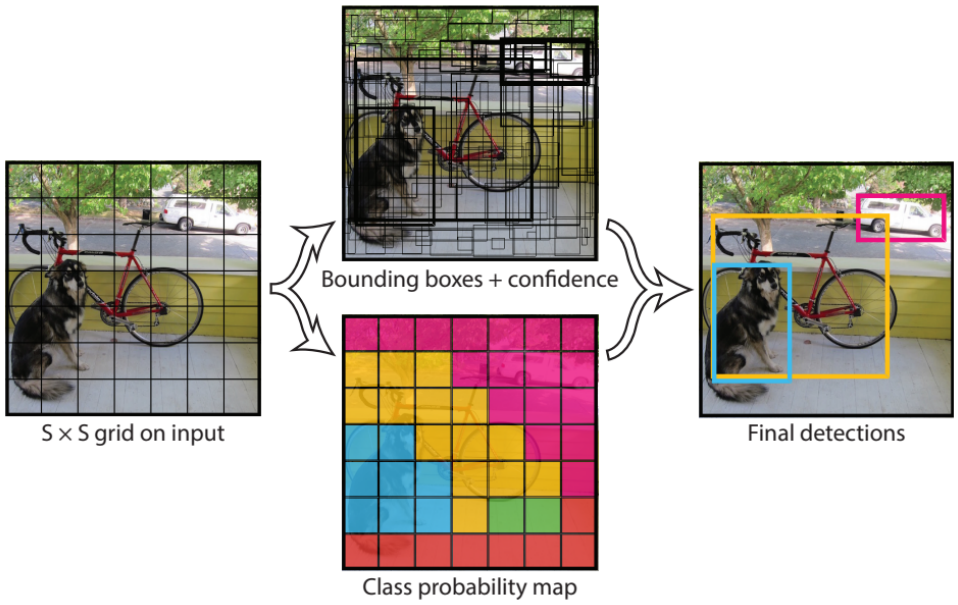

Рис. 1: Модель

Наша система моделирует обнаружение как задачу регрессии. Она разделяет изображение на сетку размером $S \times S$ и для каждой ячейки сетки предсказывает B ограничивающих рамок, уверенность в этих рамках и вероятности C классов. Эти предсказания кодируются в виде тензора размером $S \times S \times (B \cdot 5 + C)$.

1.3 Архитектура YOLOv1

В оригинальной статье представлена архитектура, изображенная на рис. 2

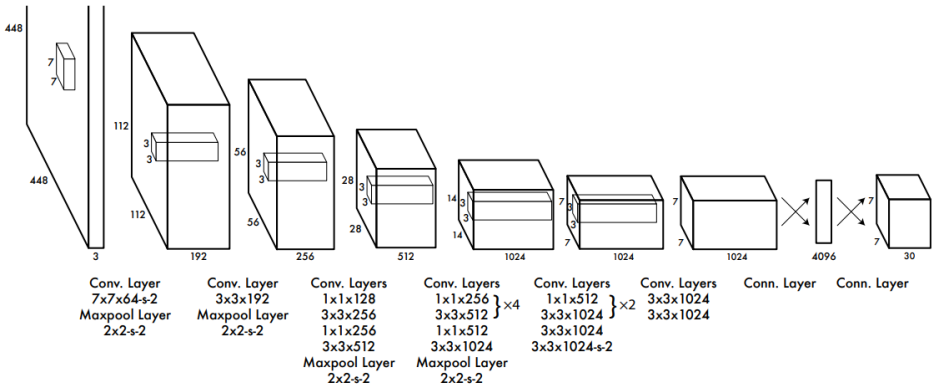

Рис. 2: Архитектура

В нашем случае мы будем использовать архитектуру ResNet50, представленная на рис. 3. Основной особенностью архитектуры ResNet является использование блоков с остаточным (residual) соединением, что позволяет обучать глубокие сети с большим количеством слоев, минимизируя проблему затухания градиентов.

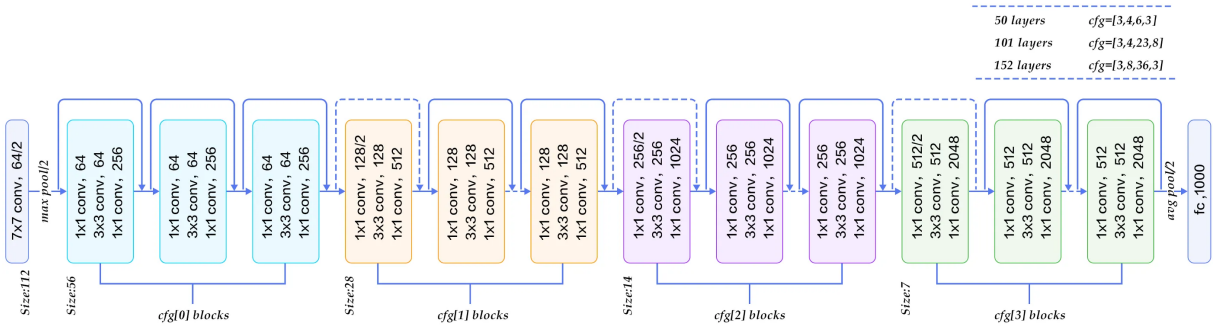

Рис. 3: Архитектура ResNet50

1.4 Функция потерь

Функция потерь представлена на рис. 4

$$
\begin{aligned}
&\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2
\end{aligned}
$$

Рис. 4: Функция потерь

Первые два слагаемых оценивают ошибку в предсказании координат объекта. Он включает в себя среднеквадратичную ошибку (MSE) между предсказанными и истинными значениями координат центра объекта и размеров объекта. Третье и четвертое слагаемое оценивает ошибку в предсказании уверенности в наличии объекта в bounding box’e. Последнее слагаемое оценивает ошибку в предсказании класса объекта.

2 Ход работы

2.1 Выбор датасет

Первоначально был самостоятельно размечен датасет, состоящий из 3000 изображений, предназначенный для детекции касок. Датасет был размечен с помощью Roboflow. Позже мы решили изменить датасет на этот. Данный датасет предназначен для детекции объектов строительной техники, а именно: эскаваторы, самосвалы и колесные погрузчики. Этот датасет содержит около 3000 изображений и уже размечен.

2.2 Стэк технологий

Модель была реализована с помощью фреймворка глубокого обучения PyTorch. Для аугментаций была использована библиотека albumentations. Также была использована библиотека PyTorch Lightning, которая упращает написание этапа обучения модели.

3 Реализация класса с архитектурой модели

```
class YOLO(nn.Module):
    def __init__(self, S=7, B=2, C=3, input_channels=3):
        super(YOLO, self).__init__()
        self.S = S
        self.C = C
        self.B = B
        resnet = models.resnet50(pretrained=True,
        ↪  weights=torchvision.models.resnet.ResNet50_Weights)
        resnet.fc = nn.Linear(resnet.fc.in_features, S*S*(C + B*5))
        self.darknet = resnet

    def forward(self, x):
        x = self.darknet(x)
        x = x.view(-1, self.S, self.S, 5*self.B + self.C)
        return x
```

Listing 1: Архитектура модели

4 Реализация класса с функцией потерь

```python
class YoloLoss(nn.Module):
    def __init__(self, S=7, B=2, C=3):
        super(YoloLoss, self).__init__()
        self.mse = nn.MSELoss(reduction="sum")
        self.S = S
        self.B = B
        self.C = C

        self.lambda_noobj = 0.5
        self.lambda_coord = 5

    def forward(self, predictions, target):
        predictions = predictions.reshape(-1, self.S, self.S, self.C + self.B * 5)

        iou_b1 = intersection_over_union(predictions[..., self.C + 1:self.C + 5], target[..., self.C + 1:self.C + 5])
        iou_b2 = intersection_over_union(predictions[..., self.C + 6:self.C + 10], target[..., self.C + 1:self.C + 5])
        ious = torch.cat([iou_b1.unsqueeze(0), iou_b2.unsqueeze(0)], dim=0)

        iou_maxes, bestbox = torch.max(ious, dim=0)
        exists_box = target[..., self.C].unsqueeze(3)

        box_predictions = exists_box * (
            (
                bestbox * predictions[..., self.C + 6:self.C + 10]
                + (1 - bestbox) * predictions[..., self.C + 1:self.C + 5]
            )
        )

        box_targets = exists_box * target[..., self.C + 1:self.C + 5]

        box_predictions[..., 2:4] = torch.sign(box_predictions[..., 2:4]) * torch.sqrt(
            torch.abs(box_predictions[..., 2:4] + 1e-6)
        )
        box_targets[..., 2:4] = torch.sqrt(box_targets[..., 2:4])

        box_loss = self.mse(
            torch.flatten(box_predictions, end_dim=-2),
            torch.flatten(box_targets, end_dim=-2),
        )

        pred_box = (
            bestbox * predictions[..., self.C + 5:self.C + 6] + (1 - bestbox) * predictions[..., self.C:self.C + 1]
        )

        object_loss = self.mse(
            torch.flatten(exists_box * pred_box),
            torch.flatten(exists_box * target[..., self.C:self.C + 1]),
        )

        no_object_loss = self.mse(
            torch.flatten((1 - exists_box) * predictions[..., self.C:self.C + 1], start_dim=1),
            torch.flatten((1 - exists_box) * target[..., self.C:self.C + 1], start_dim=1),
        )

        no_object_loss += self.mse(
            torch.flatten((1 - exists_box) * predictions[..., self.C + 5:self.C + 6], start_dim=1),
            torch.flatten((1 - exists_box) * target[..., self.C:self.C + 1], start_dim=1)
        )


        class_loss = self.mse(
            torch.flatten(exists_box * predictions[..., :self.C], end_dim=-2,),
            torch.flatten(exists_box * target[..., :self.C], end_dim=-2,),
        )

        loss = (
            self.lambda_coord * box_loss + object_loss + self.lambda_noobj * no_object_loss + class_loss
        )
        return loss
```

Listing 2: Архитектура модели

5 Реализация класса для обучения модели

```python
class YOLOLearner(pl.LightningModule):
    def __init__(self):
        super(YOLOLearner, self).__init__()
        self.model = YOLO()
        self.criterion = YoloLoss()
        self.optimizer = torch.optim.Adam(self.model.parameters(), lr=1e-5)

    def forward(self, x):
        return self.model(x)

    def training_step(self, batch, batch_idx):
        x, target = batch
        predictions = self.model(x)
        loss = self.criterion(predictions, target)
        self.log('train_loss', loss, on_step=True, on_epoch=True)
        return loss

    def validation_step(self, batch, batch_idx):
        x, target = batch
        predictions = self.model(x)
        loss = self.criterion(predictions, target)
        self.log('val_loss', loss, on_step=True, on_epoch=True, prog_bar=True)
        return loss

    def on_train_epoch_end(self):
        pred_boxes, true_boxes = get_bboxes(train_loader, model=self.model,
        ↪  iou_threshold=0.5, threshold=0.4)
        map_score = mean_average_precision(pred_boxes, true_boxes,iou_threshold=0.5,
        ↪  box_format="midpoint")
        self.log('train_mean_average_precision', map_score, on_step=False,
        ↪  on_epoch=True)

    def predict(self, batch):
        x, y = batch
        y_pred = self.model(x)
        return y_pred
    def configure_optimizers(self):
        scheduler = {
            'scheduler': ReduceLROnPlateau(self.optimizer, factor=0.1, patience=3,
            ↪  mode='max', verbose=True),
            'monitor': 'train_mean_average_precision',
        }
        return [self.optimizer], [scheduler]
```

Listing 3: Класс для обучения модели

6 Результаты обучения

6.1 графики метрик

Метрики отслеживались с помощью wandb.

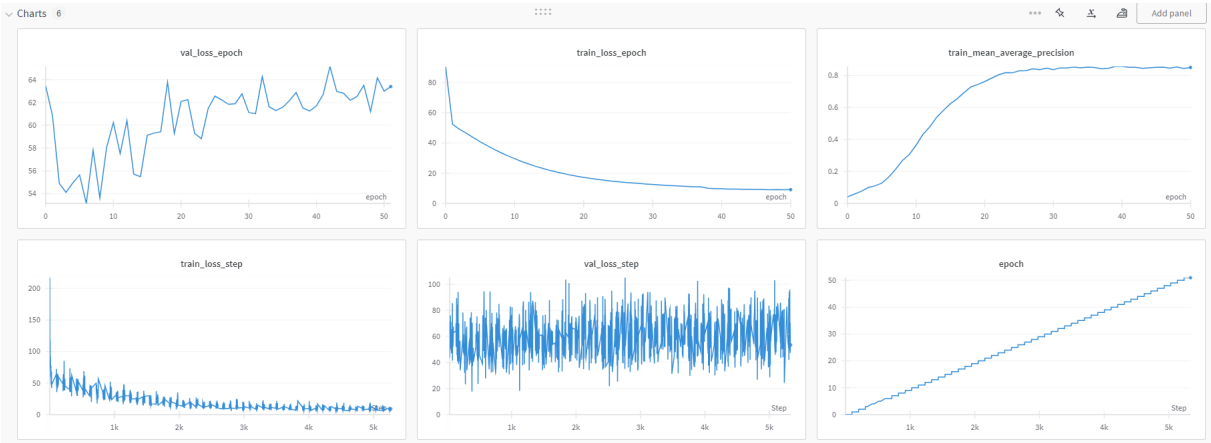

Рис. 5: Enter Caption

6.2 Результат работы модели

Рис. 6: Первая строка картинок

7 Выводы

В ходе проектного практикума была реализована модель YOLOv1 с использованием архитектуры ResNet50. Этот опыт позволил нам глубже понять ключевые аспекты детекции объектов и эффективно использовать современные технологии для улучшения процесса разработки и обучения моделей.