BABEŞ BOLYAI UNIVERSITY, CLUJ NAPOCA, ROMÂNIA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

Recognition of Road Signs Using Federated Learning

– Lab 3 report –

Team members
Ebanca Stefan
Gabor Raul
Cadar Damian
Kohan Alexandru
Name, specialisation, group, email
Scooby Doo, Software Engineering, 258, raul.gabor@stud.ubbcluj.ro

2023-2024

Chapter 1

Basic Flow Description

1.0.1 Setting up the Environment

- **Install TFF from GitHub**: Install TensorFlow Federated (TFF) from the GitHub repository. Follow the installation instructions provided in the TFF documentation.

```
pip install --upgrade tensorflow-federated
```

- **Import Libraries**: Import necessary libraries and modules, including TensorFlow and TFF.

```
import tensorflow as tf
import tensorflow_federated as tff
```

1.0.2 Data Collection and Preprocessing

- **Collect Diverse Dataset**: Gather a diverse dataset of road sign images, including images captured under different weather conditions, from various geographic locations, and with varying lighting.
- **Data Preprocessing**: Preprocess the data to ensure it's ready for model training. This includes resizing images, normalizing pixel values, and categorizing road signs based on their classes.

1.0.3 Federated Learning Setup

- **Federated Context**: Define the federated context for the road sign recognition task. This includes specifying the global model and setting up the communication structure for federated learning.
- **Server**: Configure the server to hold the global model.

1.0.4 Model Architecture

- **Design NN Model**: Design a Neural Network (NN) model for road sign recognition. Ensure the model architecture is capable of handling various road sign types and environmental conditions.

```
    def create_road_sign_model():
        model = tf.keras.Sequential([...])
        return model
```

1.0.5 Federated Training Loop

- **Global Model Initialization**: Initialize the global model with the designed NN architecture.

```
    global_model = create_road_sign_model()
    for round_num in range(num_rounds):
        # Distribute global model to client devices
        ...

        # Local Training on Client Devices
        ...

        # Secure Aggregation of Updates
        ...

        # Model Evaluation on a Validation Dataset
        ...
```

- **Iterative Federated Training**:

    1. **Local Training**: Training the model using various data sets which include images of road signs in different environments.
    2. **Secure Aggregation**: Implement secure aggregation techniques to protect model updates during aggregation.

3. **Model Evaluation**: Evaluate the global model's performance after each federated training round. Calculate metrics such as recognition accuracy.

```
evaluation_metrics = ...
evaluation_results = evaluation_metrics(global_model)
```

1.0.6 Testing and Validation

- Test the system for identifying various road signs, including different weather conditions, varying lighting, and geographic locations. Collect data from real-world scenarios to validate model performance.

1.0.7 Description of Application Functionalities

- **Data Collection**: The application collects a diverse range of data, including images of road signs captured under various conditions (day, night, rain, snow, etc.) and from different geographical locations. It also takes into account different types of road signs, such as regulatory, warning, and informational signs.
- **Real-time Recognition**: The system provides real-time recognition of road signs from live camera feeds or static images. It leverages AI to analyze the visual data and identify the type and meaning of each road sign.
- **Data Storage**: The application securely stores data about recognized road signs.

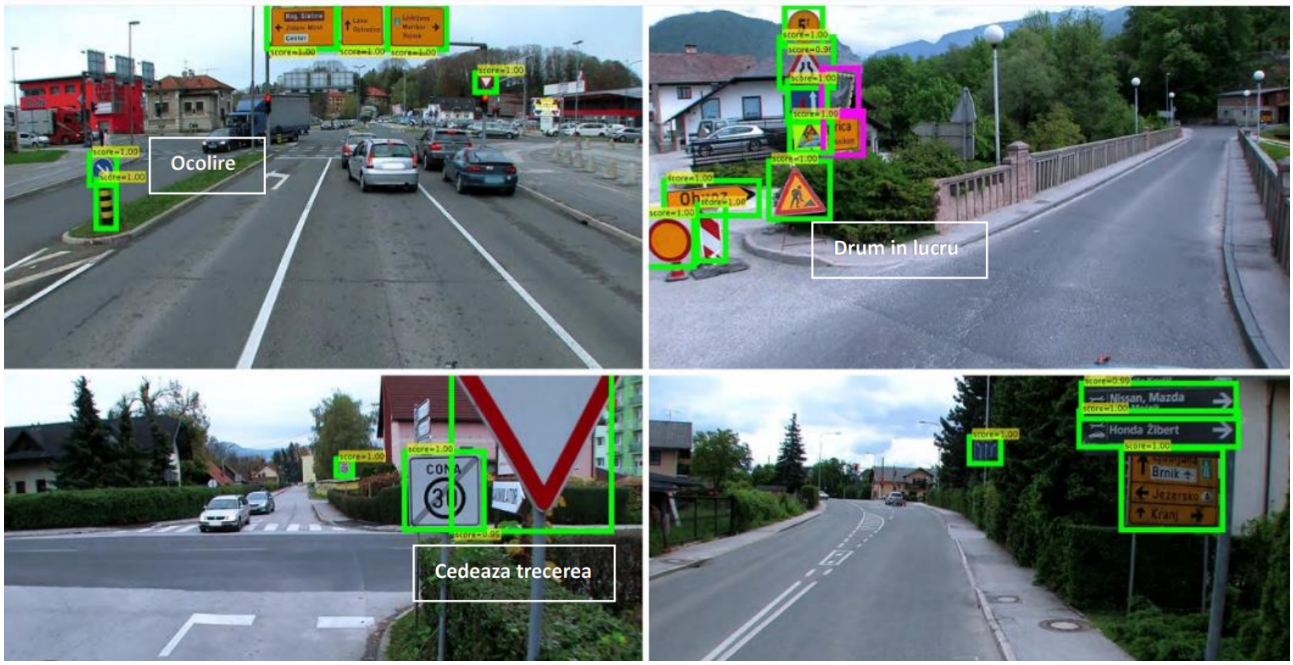

Figure 1.1: Interface

Chapter 2

Report

2.0.1 Description of Application Functionalities

- **Data Collection**: The application collects a diverse range of data, including images of road signs captured under various conditions (day, night, rain, snow, etc.) and from different geographical locations. It also takes into account different types of road signs, such as regulatory, warning, and informational signs.
- **Real-time Recognition**: The system provides real-time recognition of road signs from live camera feeds or static images. It leverages AI to analyze the visual data and identify the type and meaning of each road sign.
- **Data Storage**: The application securely stores data about recognized road signs, enabling post-trip analysis and report generation.

2.0.2 Description of the Problem Solved with AI (Plastic and Formal)

- **Layman's Explanation**: The application solves the problem of assisting drivers in recognizing and understanding road signs in real-time. It uses AI to "see" and "interpret" the road signs, under various weather conditions.
- **Formal Definition**: The problem addressed is road sign recognition, a computer vision task. The application employs deep learning, specifically neural networks (NNs), to classify road signs into predefined categories. The problem is characterized by the need for real-time recognition of road signs from visual data, with a focus on accuracy, robustness, and the ability to handle diverse environmental conditions.

2.0.3 Related Work & Useful Tools and Technologies

- **Related Work**: Several related works have explored computer vision and object recognition, including road sign recognition. Notable papers and research efforts include "Traffic Sign Recognition with Multi-Scale Convolutional Networks" by Sermanet et al. (2011) and "Deep Residual Learning for Image Recognition" by He et al. (2015). These studies laid the foundation for effective CNN-based recognition systems.

- **Useful Tools and Technologies**:

  - **TensorFlow**: TensorFlow provides the core deep learning framework for building and training the CNN model.

  - **OpenCV**: OpenCV is used for image preprocessing, manipulation, and feature extraction.

  - **Secure Data Handling**: Security measures ensure the protection of personal data, especially in systems where live camera feeds are involved.

Chapter 3

Inteligent solution description

The purpose of this report is to provide an overview of the intelligent solution developed using the provided code and Python notebook. The solution aims to address road sign detection and classification using YOLOv5, and this report will describe the logic of the solution, the technologies used, the workload, debugging procedures, and bug identification methods.

3.0.1 Description of the Intelligent Solution

The intelligent solution is designed for road sign detection and classification. It uses the YOLOv5 model for object detection and classification. The main logic of the solution involves preprocessing input data, training the model, and using the trained model for inference.

The preprocessing step involves converting annotations from the VOC format to YOLO format, allowing for a more efficient training process. This step is implemented in Python using the `xml.etree.ElementTree` library and is performed for each annotation file.

The solution leverages the YOLOv5 framework for training and inference. It is designed to detect and classify road signs in images. The model is trained on a custom dataset, which includes four classes: "trafficlight," "speedlimit," "crosswalk," and "stop."

3.0.2 Technology Used

The solution utilizes the following technologies:

- Python: The primary programming language for implementing the solution.
- YOLOv5: The framework used for object detection and classification.
- XML Parsing: The `xml.etree.ElementTree` library for parsing and processing annotation files.

- OpenCV: Used for image manipulation and visualization.

3.0.3 Workload

The workload of the solution includes data preprocessing, training the YOLOv5 model, and performing inference on input images. Preprocessing involves converting annotation files from the VOC format to YOLO format. Training the model involves specifying hyperparameters and running the training script. Inference is performed on a set of random images to detect and classify road signs.

3.0.4 Debugging

The solution incorporates debugging techniques to identify and rectify errors. Python's built-in debugging capabilities, as well as external tools like print statements and error logs, are used to monitor and debug the code. The goal is to ensure that the model is training correctly and that inference produces accurate results.

3.0.5 Bug Identification

To identify and address bugs, the solution relies on systematic error tracking. Errors or anomalies in the code are logged, and debugging tools are used to identify the source of issues. Regular testing and validation are conducted to catch and rectify bugs as early as possible.

3.0.6 Python code

The provided code is responsible for converting annotations from VOC format to YOLO format, a crucial step in preparing data for object detection tasks using YOLOv5. It consists of Python functions and a main conversion function, convert_voc_to_yolo(). The process begins by iterating through XML annotation files located in the './data/labels' directory. For each file, the code extracts the base filename without the extension, opens a corresponding text file for writing in YOLO format, and parses the XML using Python's ElementTree module. It retrieves image size information (width and height) and defines a list of class names ('trafficlight', 'speedlimit', 'crosswalk', 'stop') relevant to the object detection task. The code then iterates through object annotations in the XML file, checking if the class name is one of the predefined classes and if the object is not marked as 'difficult' (determined by a difficulty score of 1). For each valid object, it retrieves the coordinates of the bounding box in VOC format and utilizes the convert_box() function to convert these coordinates to YOLO format. Finally, it writes the class ID and YOLO-formatted coordinates to a text file, ensuring that the data is ready for training the YOLOv5 model.

```
import os
import xml.etree.ElementTree as ET

def convert_box(size, box):
    dw, dh = 1. / size[0], 1. / size[1]
    x, y, w, h = (box[0] + box[1]) / 2.0 - 1, (box[2] + box[3]) / 2.0 -
        1, box[1] - box[0], box[3] - box[2]
    return x * dw, y * dh, w * dw, h * dh

def convert_voc_to_yolo():
    for anno in os.listdir('./data/labels'):
        if anno.split('.')[1] == 'xml':
            file_name = anno.split('.')[0]
            out_file = open(f'./data/labels/{file_name}.txt', 'w')

            tree = ET.parse(os.path.join('data','labels', anno))
            root = tree.getroot()
            size = root.find('size')
            w = int(size.find('width').text)
            h = int(size.find('height').text)

            names = ['trafficlight', 'speedlimit', 'crosswalk', 'stop']

            for obj in root.iter('object'):
                cls = obj.find('name').text
                if cls in names and int(obj.find('difficult').text) !=
                   1:
                    xmlbox = obj.find('bndbox')
                    bb = convert_box((w, h), [float(xmlbox.find(x).text
                       ) for x in ('xmin', 'xmax', 'ymin', 'ymax')])
                    cls_id = names.index(cls)  # class id
                    out_file.write("␣".join([str(a) for a in (cls_id, *
```

```
                    bb)]) + '\n')
```

3.0.7 Inclusion of Jupyter Notebook Code

Here is the code from the provided Jupyter notebook:

```
%%bash
mkdir ~/.kaggle
cp kaggle.json ~/.kaggle/kaggle.json
chmod 600 ~/.kaggle/kaggle.json
pip install kaggle
kaggle datasets download -d andrewmvd/road-sign-detection
unzip road-sign-detection.zip -d data
mv ./data/annotations ./data/labels

# Setup Yolov5
%%bash
git clone https://github.com/ultralytics/yolov5
cd yolov5
pip install -r requirements.txt
from preprocessing import convert_voc_to_yolo
convert_voc_to_yolo()
```

3.0.8 Output on a set of images

The output of the algorithm was provided based on processing images in a span of 10 epochs.

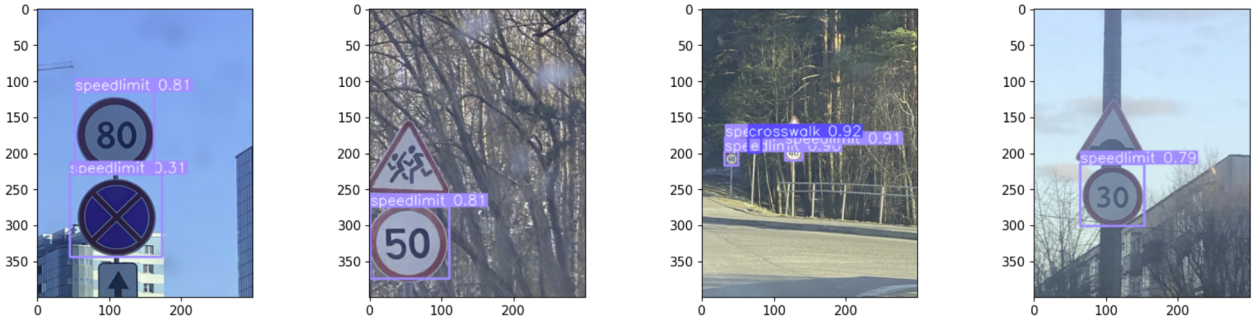

Figure 3.1: Images with captions and their accuracies

3.0.9 Conclusion

The intelligent solution for road sign detection and classification based on YOLOv5 is a robust and effective approach. It leverages state-of-the-art object detection techniques to detect and classify road signs accurately. The solution's logic, technology stack, workload, debugging procedures, and bug identification methods collectively contribute to its success in addressing the problem.

References:

- YOLOv5 GitHub Repository: https://github.com/ultralytics/yolov5
- Python Official Documentation: https://www.python.org/doc/
- OpenCV Documentation: https://docs.opencv.org/