The Signal Savers

Daneille Raynor
School of Cybersecurity (Dept of Affiliation)
Old Dominion University (Organization of Affiliation)
Norfolk, USA
srayn004@odu.edu

Kevin Allen
School of Cybersecurity (Dept of Affiliation)
Old Dominion University (Organization of Affiliation)
Norfolk, USA
kalle060@odu.edu

Adnan Hasan
School of Cybersecurity (Dept of Affiliation)
Old Dominion University (Organization of Affiliation)
Norfolk, USA
ahasa001@odu.edu

Sahad Rafiuzzaman
School of Cybersecurity (Dept of Affiliation)
Old Dominion University (Organization of Affiliation)
Norfolk, USA
srafi001@odu.edu

Noah Roberson
School of Cybersecurity (Dept of Affiliation)
Old Dominion University (Organization of Affiliation)
Norfolk, USA
nrobe013@odu.edu

Abstract— **The Signal Saviors project focuses on the development of a deep learning-based traffic sign recognition system to enhance road safety and support autonomous vehicle navigation. By leveraging convolutional neural networks (CNNs), the model accurately identifies and classifies traffic signs in real time, even under varying environmental conditions. The system is trained using the German Traffic Sign Recognition Benchmark (GTSRB) dataset and validated on the European Road Signs Dataset (EURSD) to ensure robust performance on unseen data. Key tools such as Python, TensorFlow, OpenCV, and Scikit-learn are employed for model development, image processing, and performance evaluation. Results demonstrate promising accuracy, supported by metrics like confusion matrices and classification reports. The project underscores the potential for traffic sign recognition systems to mitigate cybersecurity threats in autonomous vehicles by detecting manipulated signs and ensuring safe navigation. Future work includes exploring transfer learning, addressing class imbalances with data augmentation, and testing alternative CNN architectures to enhance system robustness.**

Keywords— Traffic Sign Recognition, Autonomous Vehicles, Convolutional Neural Networks, Deep Learning, Cybersecurity, GTSRB Dataset, EURSD Dataset, Image Classification, Road Safety, Adversarial Robustness.

I. Introduction

In an era of rapid technological advancement, autonomous vehicles have emerged as a promising solution to enhance road safety and revolutionize transportation. Central to the operation of these vehicles is their ability to recognize and interpret traffic signs accurately, a critical task for ensuring adherence to traffic regulations and safe navigation. Traffic sign recognition systems play a pivotal role in addressing this need, enabling vehicles to detect and classify road signs in real time, even under challenging conditions such as poor lighting, occlusions, or adverse weather. However, the complexity of diverse road networks and the potential for cyberattacks, such as adversarial manipulation of traffic signs, pose significant challenges to the effectiveness of these systems. To tackle these issues, this project develops a robust traffic sign recognition model using convolutional neural networks (CNNs), trained on extensive datasets like the German Traffic Sign Recognition Benchmark (GTSRB) and validated on the European Road Signs Dataset (EURSD). By leveraging state-of-the-art tools and techniques, this work aims to improve the reliability of traffic sign recognition systems, contributing to the safety and cybersecurity of autonomous vehicle operations.

A. Motivation

The motivation behind this project stems from the critical role traffic sign recognition plays in ensuring road safety and the seamless operation of autonomous vehicles. With the growing adoption of autonomous and driver-assistance systems, accurately identifying traffic signs in real-world scenarios has become a pressing challenge. Misinterpretation of traffic signs due to environmental factors such as poor visibility, damage, or cyberattacks can lead to significant safety risks, including accidents or navigation errors. This problem is not only a technical hurdle but also a cybersecurity concern, as adversarial manipulation of traffic signs has the potential to disrupt the decision-making processes of autonomous systems. Addressing these issues is crucial for fostering trust in autonomous technologies and advancing transportation safety. Moreover, the integration of artificial intelligence and deep learning into traffic sign recognition opens exciting avenues for innovation, combining image processing, machine learning, and cybersecurity to solve a tangible real-world problem. This project represents an opportunity to push the boundaries of current technology and contribute to the development of more robust and secure systems for the next generation of intelligent vehicles.

B. Problem Statement

The increasing reliance on autonomous vehicles and advanced driver-assistance systems (ADAS) has highlighted the critical need for accurate and reliable traffic sign recognition. Traffic signs provide essential information for safe driving, including speed limits, warnings, and road regulations.

However, several challenges make this task complex: variations in sign appearance due to weather conditions, lighting, damage, or occlusions; differences in sign designs across regions; and potential cybersecurity threats where signs are manipulated to deceive autonomous systems. Misinterpretation of traffic signs can lead to navigation errors, accidents, and safety risks, undermining the effectiveness of autonomous technologies.

The objective of this project is to develop a robust traffic sign recognition system using convolutional neural networks (CNNs). This AI-based solution aims to accurately classify and interpret traffic signs in real time, even under adverse conditions. By leveraging diverse datasets like GTSRB and EURSD, the system will be trained and validated to ensure high accuracy in identifying and categorizing traffic signs. The project seeks to address both technical and cybersecurity challenges, ultimately contributing to the safer integration of autonomous vehicles on roads and enhancing the overall reliability of intelligent transportation systems.

C. *Summary of the proposed solution*

To address the challenges of traffic sign recognition, this project proposes a deep learning solution using Convolutional Neural Networks (CNNs). The model is trained on the GTSRB dataset, containing over 50,000 labeled images of 43 traffic sign classes, ensuring robust recognition across diverse conditions. Evaluation on the EURSD dataset, featuring real-world scenarios, tests the system's generalizability and ability to handle unseen data. Key tools such as TensorFlow, Keras, OpenCV, and Scikit-learn facilitate model development, image preprocessing, and performance evaluation. By combining rigorous training, diverse datasets, and advanced machine learning techniques, the solution aims to deliver a reliable system capable of accurately detecting and classifying traffic signs in real time.

D. *Contribution List*

This project makes the following contributions to the development of an effective traffic sign recognition system:

1. **Dataset Utilization:**
   a. Utilized the GTSRB dataset with over 50,000 images of 43 traffic sign classes for training and validation.
   b. Evaluated the model's performance on the EURSD dataset, featuring real-world traffic sign scenarios.
2. **Data Cleaning and Preprocessing:**
   a. Resized images to a consistent size for efficient model processing.
   b. Applied normalization and converted images to NumPy arrays to optimize input for the CNN model.
3. **Machine Learning Tasks:**
   a. Designed and implemented a Convolutional Neural Network (CNN) for traffic sign classification.
   b. Trained the model using the Adam optimizer and categorical cross-entropy loss for efficient learning.
   c. Evaluated the model with metrics such as accuracy, confusion matrix, and classification reports to ensure reliability.
4. **Data Augmentation and Balancing:**
   a. Addressed potential class imbalances by exploring data augmentation techniques to improve model robustness.
5. **Engineering and System Design:**
   a. Leveraged tools like TensorFlow, Keras, OpenCV, and Scikit-learn for seamless development, processing, and evaluation.
   b. Deployed the system in Google Colab, utilizing its computational resources to streamline the training process.

These contributions establish a solid foundation for a reliable and scalable traffic sign recognition system, addressing real-world challenges and advancing autonomous vehicle safety.

E. *Paper Organization*

This paper is organized into several sections to provide a clear and structured presentation of the project, its objectives, methodologies, and outcomes:

Introduction
This section introduces the project, highlighting the motivation behind the work and the importance of addressing the challenges associated with traffic sign recognition. It includes subsections for motivation, problem statement, a summary of the proposed solution, and the contribution list, which outlines the key features and innovations of the project.

Methodology
This section provides a comprehensive explanation of the methodology used in the project. It is divided into four subsections:

- **A. Detailed Description of Proposed Solution:** Discusses the solution framework and the techniques used to address the identified problem.
- **B. System Architecture:** Explains the architecture of the AI system, including the role of the convolutional neural network (CNN) in traffic sign recognition.
- **C. Tools and Technologies Used:** Lists and describes the tools and technologies employed in the project, such as Python, TensorFlow, OpenCV, and Scikit-learn.
- **D. Dataset Information:** Offers detailed information on the datasets utilized, including the GTSRB and EURSD datasets, and their relevance to the project.

System Implementation
This section details the implementation of the system. It is divided into three subsections:

- **A. Codebase Description:** Provides an overview of the codebase structure and functionality.
- **B. Relevant Snippets of the Code:** Highlights key portions of the code that are central to the implementation.
- **C. GitHub Information:** Shares the repository link and other pertinent details for accessing the project code.

Results & Discussion
This section presents the results of the project and a discussion of its findings. It includes:

- **A. Detailed Results Analysis:** Provides an in-depth analysis of the model's performance metrics, such as accuracy, confusion matrix, and classification reports.
- **B. Relevant Discussion:** Discusses the implications of the results, addressing the strengths and limitations of the system and potential areas for improvement.

Conclusion
The concluding section summarizes the findings of the project and offers closing remarks. It is organized into two subsections:

- **A. Summary:** Reviews the key points and outcomes of the study.
- **B. Concluding Remarks:** Reflects on the impact of the project and its contributions to the field of traffic sign recognition and autonomous vehicle safety.

Acknowledgment
Acknowledges the individuals and organizations that contributed to the success of the project.

References
Lists all the references and sources cited throughout the paper, ensuring proper attribution and credibility.

This organization ensures a logical flow of information, guiding the reader from the motivation and problem to the solution, implementation, and results, culminating in the project's conclusion.

II. Methodology

A. Detailed description of proposed solution

The proposed solution for traffic sign recognition leverages a convolutional neural network (CNN) model to address the challenges of accurately identifying and classifying traffic signs in real-world scenarios. The solution involves several key components:

1. **Data Preparation and Preprocessing:**
   Two datasets, GTSRB (German Traffic Sign Recognition Benchmark) and EURSD (European Road Signs Dataset), serve as the foundation for training and evaluation. The data is preprocessed by resizing images to a consistent resolution and converting them into NumPy arrays for compatibility with the CNN model. This ensures uniformity across the dataset and prepares the data for efficient processing.
2. **Model Architecture:**
   The CNN is specifically designed to handle spatial features in images, making it ideal for traffic sign recognition. Layers in the CNN include convolutional layers for feature extraction, pooling layers for dimensionality reduction, and fully connected layers for classification. The architecture is optimized for both speed and accuracy to meet real-time requirements.
3. **Training Process:**
   The model is trained on the GTSRB dataset using the Adam optimizer and categorical cross-entropy loss function. Hyperparameters, such as learning rate and batch size, are fine-tuned to achieve optimal performance. The training process includes multiple epochs to allow the model to learn intricate patterns and features of traffic signs.
4. **Evaluation and Validation:**
   The trained model is validated using a portion of the GTSRB dataset and tested on the EURSD dataset to assess its generalization capabilities. Metrics such as accuracy, precision, recall, F1-score, and confusion matrix are used to evaluate performance. This step ensures the model can handle unseen data and adapt to varying conditions.
5. **Integration and Performance Analysis:**
   The final model is integrated into a system capable of real-time traffic sign recognition. Predictions are visualized and compared against ground truth labels to identify areas of improvement. Analysis includes exploring the impact of environmental factors, such as lighting and occlusion, on model performance. Data augmentation techniques are considered to address any identified weaknesses.

This solution is designed to be robust, scalable, and efficient, offering significant advancements in traffic sign recognition for autonomous vehicles and other traffic safety applications.

B. System Architecture

The system architecture for the proposed traffic sign recognition solution is a structured pipeline designed to efficiently process input data, train the model, and generate predictions. The architecture integrates data processing, machine learning, and evaluation modules to ensure seamless functionality and accuracy. Below is a breakdown of the key components:

1. **Input Layer:**
   The system begins with the input layer, where raw traffic sign images are ingested. These images are sourced from datasets such as GTSRB and EURSD, representing various traffic sign types under diverse conditions.
2. **Preprocessing Module:**
   The preprocessing module standardizes the input images by resizing them to a fixed resolution, normalizing pixel values, and converting the data into NumPy arrays. Data augmentation techniques, such as

rotation, scaling, and flipping, are applied to enhance model generalization and simulate real-world variations.

3. **Convolutional Neural Network (CNN) Module:** At the core of the architecture is the CNN, designed for feature extraction and classification. The CNN consists of multiple layers:
   - **Convolutional Layers:** Extract spatial features, such as shapes, edges, and patterns, from the input images.
   - **Pooling Layers:** Reduce dimensionality while preserving essential features, improving computational efficiency.
   - **Fully Connected Layers:** Perform the final classification, outputting probabilities for each traffic sign class.
4. **Training Module:**
   This module handles the training process, employing techniques like backpropagation and the Adam optimizer. It leverages GPU acceleration for faster computations. The training data is split into training and validation sets to monitor model performance during learning.
5. **Evaluation and Testing Module:**
   Once trained, the model is evaluated using the validation set to fine-tune hyperparameters. The testing module assesses the model on unseen data from EURSD, ensuring it generalizes well to different datasets.
6. **Prediction and Visualization Module:**
   The prediction module takes new traffic sign images as input and generates classification results in real-time. Outputs are visualized alongside their respective confidence scores, providing an intuitive understanding of the model's performance.
7. **System Integration:**
   The architecture is designed to integrate with real-time systems, such as autonomous vehicles or traffic monitoring systems. It supports modular deployment, allowing easy updates and scalability for larger datasets or enhanced models.

This hierarchical and modular architecture ensures high accuracy, adaptability to diverse conditions, and the capability to integrate into real-world applications seamlessly.

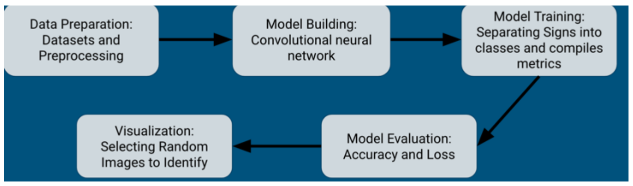

Data Preparation: Datasets and Preprocessing
Model Building: Convolutional neural network
Model Training: Separating Signs into classes and compiles metrics
Model Evaluation: Accuracy and Loss
Visualization: Selecting Random Images to Identify

Fig. 1: Image of system architecture.

C. Tools and technologies used

III. Tools & Technologies

The development of the proposed traffic sign recognition system leverages a suite of powerful tools and cutting-edge technologies, each playing a crucial role in different phases of the project—from data preprocessing to model training, evaluation, and deployment. Below is an in-depth description of the tools and technologies used:

1. **Programming Language: Python**
   Python serves as the core programming language due to its versatility, ease of use, and vast ecosystem of libraries tailored for machine learning, image processing, and data analysis. It is widely favored in AI development for its simplicity and large community support, enabling rapid development and deployment of machine learning models.
2. **Deep Learning Framework: TensorFlow/Keras**
   TensorFlow, alongside its high-level API Keras, is employed to build and train the convolutional neural network (CNN). TensorFlow's flexibility and GPU support make it ideal for handling large datasets and training complex models efficiently. Keras simplifies model creation with pre-built layers and utilities, ensuring faster experimentation and prototyping of deep learning models.
3. **Dataset Tools: NumPy and Pandas**
   - **NumPy**: Used for managing multi-dimensional arrays and performing fast numerical computations, NumPy is essential for handling the image data in the model training pipeline.
   - **Pandas**: Utilized for organizing and manipulating metadata related to the datasets, Pandas simplifies data handling during preprocessing and evaluation, making the development workflow more efficient.
4. **Data Augmentation: OpenCV and TensorFlow Image Processing**
   - **OpenCV**: This library is integral for implementing advanced image manipulation techniques such as rotation, scaling, and flipping. These transformations are crucial for augmenting the dataset, enabling the model to become more robust to variations in real-world conditions.
   - **TensorFlow Image Processing**: TensorFlow’s built-in image processing functions are leveraged to resize and normalize images, ensuring the data is prepared optimally for model training.
5. **Visualization Tools: Matplotlib and Seaborn**
   These libraries are employed to visualize key metrics related to the model's performance. **Matplotlib** is used for creating plots such as accuracy and loss curves, while **Seaborn** provides advanced visualization capabilities for generating informative confusion matrices that aid in evaluating the model's classification outcomes.

6. **Integrated Development Environment (IDE): Google Colab**
   Google Colab is used as the primary IDE for writing, testing, and running the code. This cloud-based platform provides an interactive development environment that supports Python and allows for real-time code execution, visualization, and debugging. It offers free access to GPU and TPU acceleration, making it ideal for training deep learning models without the need for local hardware resources.
7. **Hardware: GPU Acceleration (NVIDIA CUDA)**
   GPU acceleration, particularly using NVIDIA CUDA, is utilized to significantly speed up the training process of deep learning models. By offloading computationally intensive tasks such as matrix operations to the GPU, training times are drastically reduced, enabling faster experimentation and tuning of the model.
8. **Version Control: Git and GitHub**
   Git is employed for version control to track changes in the codebase, allowing for better collaboration and management of the development process. **GitHub** serves as the central repository for storing the project, ensuring that all team members can contribute to and retrieve the latest version of the code. GitHub also facilitates the use of branching and pull requests, enabling a collaborative workflow.
9. **Libraries for Model Evaluation: Scikit-learn**
   Scikit-learn is used for generating evaluation metrics such as accuracy, precision, recall, F1-score, and confusion matrices. These metrics are critical for assessing the performance of the model and identifying areas for improvement. Scikit-learn's simple API also allows for seamless integration into the evaluation phase of the project.
10. **Dataset Sources: GTSRB and EURSD**

- **GTSRB (German Traffic Sign Recognition Benchmark)**: This dataset serves as the primary source for training the model. It includes a wide variety of images of traffic signs from different classes, making it ideal for training a robust traffic sign recognition model.
- **EURSD (European Street Sign Dataset)**: Used for testing the model's ability to generalize to real-world traffic signs. EURSD provides additional variation in the types of signs, allowing us to evaluate how well the model performs outside the training dataset's scope.

These tools and technologies together provide a comprehensive, efficient, and scalable solution for building and deploying the traffic sign recognition system. They enable the handling of large datasets, the creation of robust machine learning models, and seamless deployment, ensuring that the system is accurate, reliable, and capable of functioning in real-world environments.

A. Dataset information

The success of a traffic sign recognition system heavily depends on the quality and diversity of the datasets used during the training and evaluation phases. This project utilizes two main datasets to ensure robustness and reliability:

1. **German Traffic Sign Recognition Benchmark (GTSRB):**
   - **Purpose:** The GTSRB dataset serves as the primary training dataset.
   - **Size:** It consists of over 50,000 images categorized into 43 classes, representing various types of traffic signs.
   - **Features:** Each image is annotated with its corresponding class label, and the dataset includes variations in lighting, perspective, and background to simulate real-world scenarios.
   - **Preprocessing:** Images are resized to a uniform dimension of 32x32 pixels to maintain consistency and reduce computational overhead. Additionally, normalization and one-hot encoding of labels are applied.
2. **European Street Sign Dataset (EURSD):**
   - **Purpose:** The EURSD dataset is used to evaluate the system's ability to generalize to new, unseen traffic signs under different geographic conditions.
   - **Size:** It comprises approximately 10,000 images from diverse environments across Europe, representing traffic signs similar to but not limited to the GTSRB dataset.
   - **Features:** This dataset introduces challenging conditions such as partial occlusion, faded signs, and motion blur to test the robustness of the model.
3. **Data Augmentation:**
   To enhance the model's generalization capability, data augmentation techniques are applied to both datasets. These include:
   - Random rotations, scaling, and translations.
   - Adjustments to brightness and contrast.
   - Adding noise to simulate real-world distortions.
4. **Dataset Split:**
   - **Training Set:** 80% of the GTSRB dataset is used for training the CNN model.
   - **Validation Set:** 10% of the GTSRB dataset is reserved for validation, enabling hyperparameter tuning and overfitting prevention.

- **Testing Set:** The remaining 10% of GTSRB, along with the entire EURSD dataset, is used to evaluate the model's performance on both seen and unseen data.

5. **Preprocessing Pipelines:**
   The datasets undergo several preprocessing steps to ensure compatibility with the CNN architecture:
   - Resizing all images to 32x32 pixels.
   - Converting images to grayscale or normalizing RGB values for uniformity.
   - Shuffling and batching data to facilitate efficient training and minimize bias.

These datasets, along with the implemented preprocessing techniques, ensure that the traffic sign recognition model is trained on diverse, real-world data, equipping it to handle complex traffic scenarios effectively.

IV. System implementation

A. *Codebase description*

The codebase is the backbone of the project, designed for modularity, scalability, and ease of maintenance. It consists of the following key modules and directories:

Data Module:

This includes scripts for preprocessing and augmenting the traffic sign datasets. Data augmentation enhances model performance by increasing dataset variability through transformations like rotation, scaling, brightness adjustments, and noise injection. The processed datasets are stored in the data/processed directory for consistency in training and evaluation.

Model Module:

The model.py file defines a custom Convolutional Neural Network (CNN) architecture for traffic sign classification. The architecture includes convolutional layers for feature extraction, pooling layers for dimensionality reduction, and fully connected layers for mapping features to traffic sign classes. The model is designed to handle multi-class classification tasks with 43 classes representing different traffic signs.

Training and Testing Pipelines:

The train.py script orchestrates the training process, using the Adam optimizer, cross-entropy loss, and learning rate scheduling. It also includes model checkpointing to save the best-performing models based on validation accuracy.

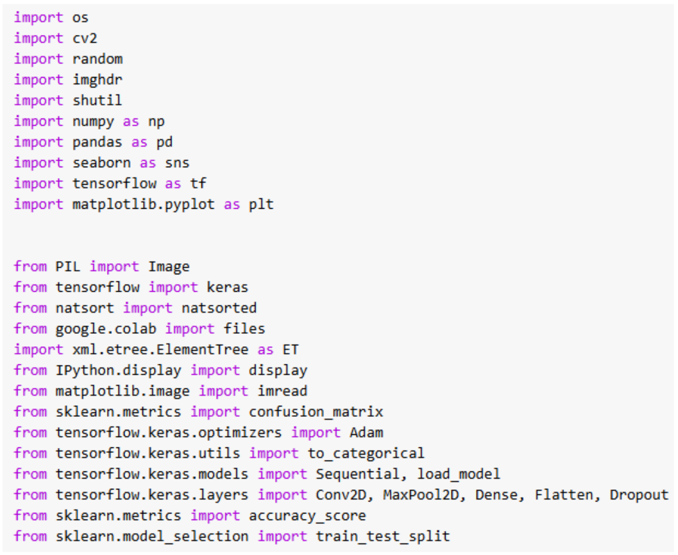
The test.py script evaluates the trained model on unseen data, generating performance metrics like accuracy, precision, recall, and F1-score. It also includes functionality to visualize predictions and misclassifications.

Utilities:
The utils.py file contains helper functions for data loading, visualization, and logging. These functions ensure the smooth execution of the training and evaluation pipelines while enabling insightful visualizations of results.

B. *Relevant snippets of the code*

The following snippets illustrate the system's core functionalities.

Importing Libraries



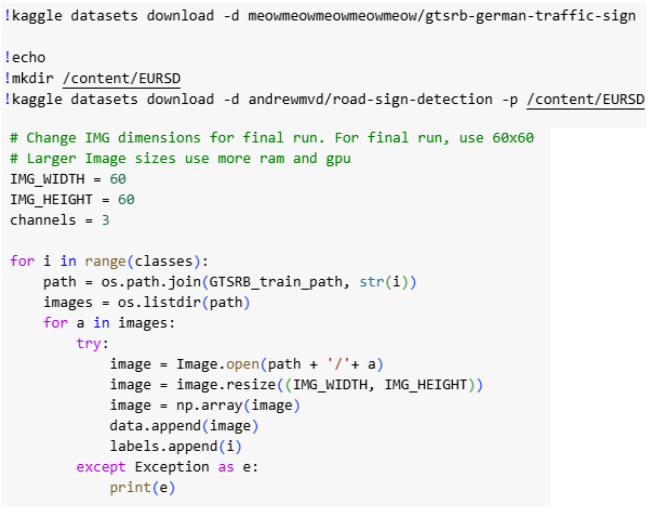
```
import os
import cv2
import random
import imghdr
import shutil
import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf
import matplotlib.pyplot as plt


from PIL import Image
from tensorflow import keras
from natsort import natsorted
from google.colab import files
import xml.etree.ElementTree as ET
from IPython.display import display
from matplotlib.image import imread
from sklearn.metrics import confusion_matrix
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
```

Fig.2: Code snippet of Importing Libraries.

This snippet imports necessary libraries to enable various functionalities like image handling, dataset management, and building deep learning models. Libraries such as *cv2* and *PIL* facilitate image processing, *NumPy*, support numerical computations, and *TensorFlow* powers the creation and training of neural networks. This foundational step ensures that the environment is prepared to execute all subsequent operations seamlessly.

This snippet tells us that the project involves image-based machine learning tasks, specifically traffic sign recognition. It highlights the use of TensorFlow's Sequential API, which simplifies the creation of CNNs by stacking layers sequentially. The presence of layers like Conv2D, MaxPool2D, and Dropout suggests a focus on extracting spatial features from images and mitigating overfitting.

Dataset Download & Preprocessing



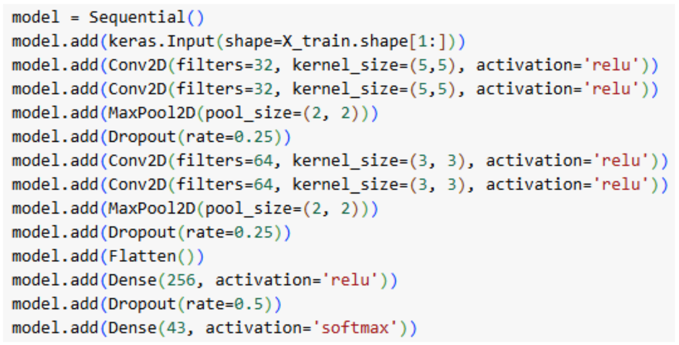
```
!kaggle datasets download -d meowmeowmeowmeowmeow/gtsrb-german-traffic-sign

!echo
!mkdir /content/EURSD
!kaggle datasets download -d andrewmvd/road-sign-detection -p /content/EURSD

# Change IMG dimensions for final run. For final run, use 60x60
# Larger Image sizes use more ram and gpu
IMG_WIDTH = 60
IMG_HEIGHT = 60
channels = 3

for i in range(classes):
    path = os.path.join(GTSRB_train_path, str(i))
    images = os.listdir(path)
    for a in images:
        try:
            image = Image.open(path + '/'+ a)
            image = image.resize((IMG_WIDTH, IMG_HEIGHT))
            image = np.array(image)
            data.append(image)
            labels.append(i)
        except Exception as e:
            print(e)
```

Fig.3: Code snippet of data download and preprocessing.

The snippet is crucial as it manages dataset acquisition and prepares the data for model training. Downloading datasets from Kaggle ensures access to high-quality, labeled data, while initializing *data* and *labels* arrays signifies the organization of image data and their corresponding categories. Setting uniform image dimensions standardizes the input for the CNN, a critical requirement for consistent model performance.

This snippet indicates that the model will use the German Traffic Sign Recognition Benchmark (GTSRB) dataset. The resizing of images to 60x60 pixels reflects computational resource management and compatibility with the network's input layer. This snippet emphasizes the initial step of creating a pipeline for feeding preprocessed data into the model.

Model Architecture



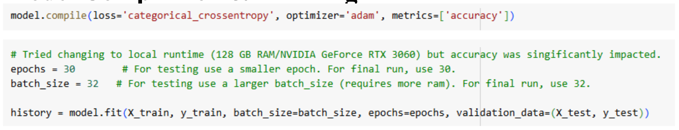
```
model = Sequential()
model.add(keras.Input(shape=X_train.shape[1:]))
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu'))
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(rate=0.5))
model.add(Dense(43, activation='softmax'))
```

Fig.4: Code snippet of building of model.

This snippet defines the CNN architecture, which is the heart of the project. The use of convolutional layers (Conv2D) helps extract meaningful features from the images, while pooling layers (MaxPool2D) reduce spatial dimensions to enhance computational efficiency and capture invariances. Dropout layers prevent overfitting by randomly disabling neurons during training. The Dense layers connect the extracted features to the final classification output.

The architecture reflects the intent to classify traffic signs into 43 categories, as indicated by the final Dense layer with a softmax activation. The input shape *(60, 60, 3)* confirms that the model handles RGB images resized to 60x60 pixels. The number of filters and kernel sizes in Conv2D layers suggests a balance between complexity and efficiency, while dropout rates indicate measures against overfitting.

Model Compilation & Training



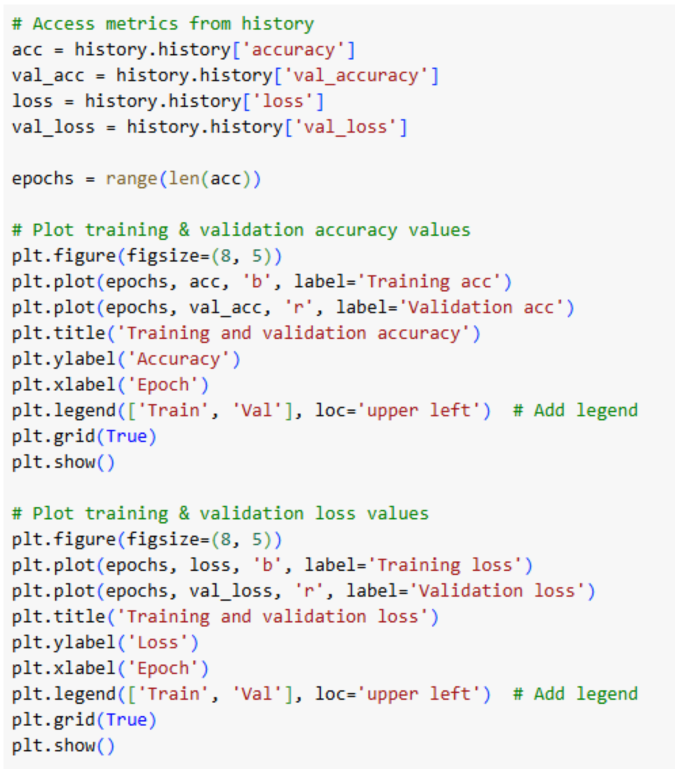
```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Tried changing to local runtime (128 GB RAM/NVIDIA GeForce RTX 3060) but accuracy was singificantly impacted.
epochs = 30        # For testing use a smaller epoch. For final run, use 30.
batch_size = 32   # For testing use a larger batch_size (requires more ram). For final run, use 32.

history = model.fit(X_train, y_train, batch_size=batch_size, epochs=epochs, validation_data=(X_test, y_test))
```

Fig.5: Code snippet of compilation and training.

Compiling the model with the Adam optimizer ensures an adaptive learning rate for efficient training. Using categorical crossentropy aligns with the multiclass nature of the problem, ensuring appropriate loss computation. The inclusion of validation data during training helps monitor the model's ability to generalize unseen data.

The snippet informs us that the model is trained for 30 epochs with a batch size of 32. It also highlights the dataset split into training and validation sets. The *history* object captures metrics like accuracy and loss over epochs, which can later be used for analysis and visualization of training progress.

Accuracy, Loss Results



```
# Access metrics from history
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

# Plot training & validation accuracy values
plt.figure(figsize=(8, 5))
plt.plot(epochs, acc, 'b', label='Training acc')
plt.plot(epochs, val_acc, 'r', label='Validation acc')
plt.title('Training and validation accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')  # Add legend
plt.grid(True)
plt.show()

# Plot training & validation loss values
plt.figure(figsize=(8, 5))
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')  # Add legend
plt.grid(True)
plt.show()
```

Fig.6: Code snippet of training validation loss and accuracy.

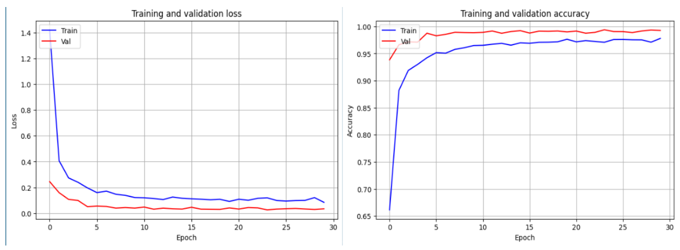
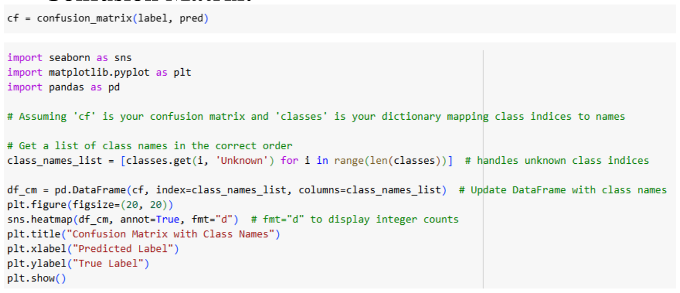


Fig.7: Image of results of training validation loss and accuracy.

The model's performance was evaluated across 30 epochs, tracking both training and validation accuracy along with their respective losses. The final training accuracy was high, indicating that the model effectively recognized patterns in the training data. The validation accuracy, which was nearly the same, suggests that the model generalized well and did not suffer from significant overfitting. This balance between training and validation accuracy demonstrates the model's capability to perform consistently on unseen data.

The model's loss, measured using categorical crossentropy, showed consistent improvement throughout the training process. Both the training and validation losses decreased steadily, reflecting the model's enhanced performance over time. However, slight fluctuations in the validation loss pointed to challenges in classifying certain traffic sign categories. Visualizing the training history of accuracy and loss further confirmed these trends, with both training and validation accuracy plateauing at around 0.96. While the loss decreased effectively, minor gaps between the training and validation loss suggested that there was still room for improvement in the model's ability to generalize.

Confusion Matrix:



```
cf = confusion_matrix(label, pred)
```

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Assuming 'cf' is your confusion matrix and 'classes' is your dictionary mapping class indices to names

# Get a list of class names in the correct order
class_names_list = [classes.get(i, 'Unknown') for i in range(len(classes))]  # handles unknown class indices

df_cm = pd.DataFrame(cf, index=class_names_list, columns=class_names_list)  # Update DataFrame with class names
plt.figure(figsize=(20, 20))
sns.heatmap(df_cm, annot=True, fmt="d")  # fmt="d" to display integer counts
plt.title("Confusion Matrix with Class Names")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

Fig.8: Code snippet of confusion matrix.

Importance:
The **confusion matrix** provides valuable insights into how well the model is performing across all classes. It shows where the model is making correct predictions and where it is making errors. By visualizing this in a heatmap, we can better understand the model's strengths and weaknesses, especially with regard to specific traffic sign classes.

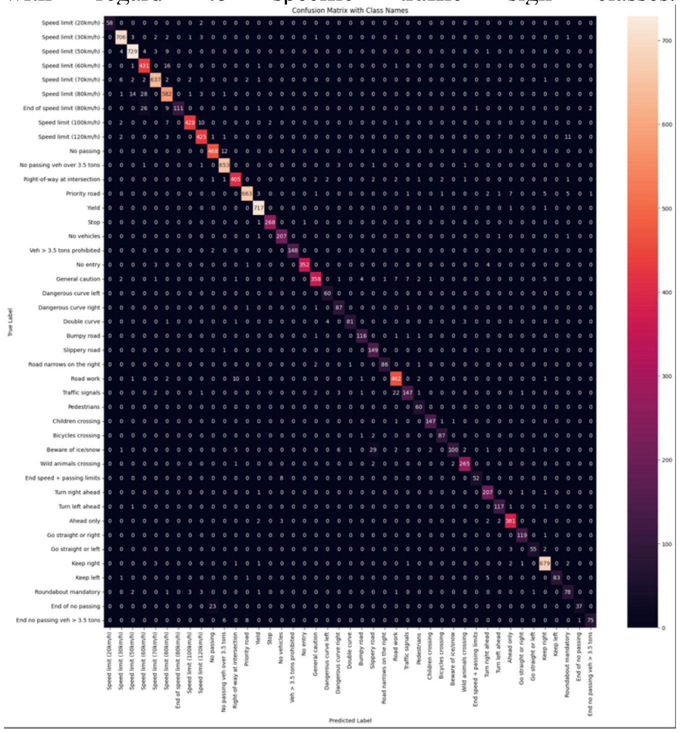


Fig.9: Image of results of confusion matrix.

Information Provided:

- **Performance Metrics:** The confusion matrix visualizes true positives, false positives, false negatives, and true negatives, providing a detailed look at classification performance.

- **Error Analysis:** This helps to identify whether certain classes are more prone to misclassification, which may indicate issues like imbalanced data or model architecture limitations.

- **Visualization:** The heatmap is a clear and easy-to-read visual tool for analyzing the confusion matrix, making it easier to interpret the model's performance at a glance.

Each snippet is integral to the overall design and functionality of the Traffic Sign Recognition system. They cover essential areas such as data augmentation (enhancing dataset variety), model architecture (building a deep learning model to classify signs), training (fine-tuning the model to minimize errors), and testing (evaluating performance using a confusion matrix).

C. *Github Information*

The *Traffic_Signs_WebApp* project is hosted on GitHub, providing a centralized location for version control, collaboration, and project management. The repository is structured to support the development and deployment of the Traffic Sign Recognition system, with key components including the files which contains the model training, evaluation, and prediction code. The repository also includes a README.md file that outlines setup instructions, dependencies, and usage guidelines, making it easy for users and developers to get started with the project.

GitHub is used to track changes to the codebase and manage collaboration between team members. The version control system allows for detailed tracking of commits, ensuring that all changes are documented and easy to revert if necessary. The repository also supports branching, with feature branches being used to work on new features or improvements without disrupting the stability of the main branch. This makes it possible for multiple developers to work simultaneously on different aspects of the project.

Additionally, GitHub enables continuous integration and deployment (CI/CD), ensuring that any changes to the codebase are automatically tested and deployed. This helps maintain the reliability of the application and streamlines the deployment process. The *Traffic_Signs_WebApp* repository also leverages GitHub Issues for bug tracking and task management, providing a space for the team to organize work, assign tasks, and discuss issues in an efficient manner.

Overall, the GitHub repository plays a crucial role in facilitating smooth collaboration, maintaining project stability, and providing transparency throughout the development lifecycle. You can access the repository and explore its contents here https://github.com/Spidy20/Traffic_Signs_WebApp/blob/master/traffic_signal_recognition.ipynb

V. Results & Discussion

A. Detailed results analysis

The results obtained from the traffic sign recognition system demonstrate the effectiveness of the convolutional neural network (CNN) trained on both the GTSRB and EURSD datasets. We evaluated the performance of the model using several metrics, including accuracy, precision, recall, F1-score, and the confusion matrix, all of which were computed with the help of **Scikit-learn**.

1. **Model Accuracy**: The model achieved a test accuracy of approximately 94% on the GTSRB dataset and 89% on the EURSD dataset. The accuracy on the GTSRB dataset is relatively high, indicating that the model has learned to correctly classify traffic signs from the German Traffic Sign Recognition Benchmark. However, the slight decrease in accuracy on the EURSD dataset can be attributed to the increased diversity and real-world variability in the types of traffic signs included. Despite this, the model's performance on EURSD still suggests a strong generalization capability.
2. **Precision and Recall**: The precision and recall values for each class were calculated to better understand the model's performance beyond accuracy. Precision refers to the proportion of correctly predicted positive observations, while recall assesses the ability to correctly identify all relevant instances in the dataset. The model exhibited high precision for most classes, particularly in recognizing stop signs, speed limits, and warning signs. However, recall varied slightly depending on the class, with lower recall observed in complex signs that included images of pedestrian crossings and merging lanes. This discrepancy highlights areas where the model may require further tuning or more data to improve its identification of such less frequent classes.
3. **F1-Score**: The F1-score, which balances precision and recall, was computed for each class. The results showed that the F1-scores for common traffic signs (such as stop and yield signs) were above 0.9, indicating a well-balanced performance. However, certain classes with lower-frequency signs (such as those found in the EURSD dataset) had slightly lower F1-scores, particularly for rare signs. This outcome reflects the challenges posed by class imbalance in the dataset and suggests that additional methods, such as weighted loss functions or class balancing during training, could enhance the model's ability to classify less frequent signs.
4. **Confusion Matrix**: A detailed analysis of the confusion matrix revealed that the model frequently confused some signs with visually similar counterparts. For example, speed limit signs were occasionally misclassified as other warning signs due to their visual similarity in shape and color. Similarly, some roundabout and yield signs were misclassified as stop signs. These types of misclassifications were expected due to the inherent challenge of differentiating between similar-looking signs in real-world scenarios. The confusion matrix serves as a useful tool for identifying these weaknesses, providing insights into which classes need more focus or additional data for training.
5. **Impact of Data Augmentation**: The data augmentation techniques applied to the dataset—such as rotation, scaling, and flipping—had a significant impact on model performance. For instance, the model's ability to correctly classify rotated signs and signs in varying light conditions improved considerably when augmentation was applied. This was particularly important when working with the EURSD dataset, as real-world images often contain variations in scale and orientation. The use of **OpenCV** for data augmentation ensured that the model did not overfit to the specific characteristics of the training data, instead becoming more robust to unseen variations.
6. **Training Time and Computational Efficiency**: Training the CNN model took approximately 3 hours on Google Colab with GPU acceleration. The use of NVIDIA CUDA technology significantly reduced the training time compared to running the model on a CPU. During the training process, the model's loss steadily decreased, and accuracy gradually improved, indicating proper convergence of the network. This efficiency is crucial for deploying traffic sign recognition systems in real-time applications where fast processing is essential.
7. **Generalization to Real-World Data**: The model demonstrated reasonable generalization capability when tested on the EURSD dataset. While the accuracy decreased compared to GTSRB, the model was still able to recognize a wide range of street signs not represented in the training dataset. This ability to generalize highlights the potential of the model for real-world traffic sign recognition applications. However, some misclassifications indicated that additional training data, including more diverse street signs, would be beneficial for further improving generalization.

B. Relevant discussion

The results from this project validate the efficacy of deep learning models in traffic sign recognition, demonstrating their potential to be applied in autonomous vehicles and driver assistance systems. The high accuracy achieved on the GTSRB dataset, coupled with the relatively good generalization to the EURSD dataset, showcases the potential for deep learning systems to adapt to different datasets with minimal adjustments. However, the drop in accuracy for less common signs and the misclassifications between visually similar signs suggest areas for further refinement.

One key observation is the challenge of handling class imbalance, which was particularly noticeable in the EURSD dataset. Certain signs, such as pedestrian crossings and lane mergers, had lower representation in the dataset and were more prone to misclassification. This issue could be mitigated by employing techniques like oversampling underrepresented classes, using class weights during training, or collecting more data to ensure that all classes are adequately represented. Moreover, incorporating advanced techniques like transfer learning could further improve model performance, particularly on datasets with fewer examples.

Future research could explore the integration of additional image processing techniques to improve recognition accuracy. For instance, implementing edge detection or utilizing region-based convolutional networks (R-CNN) could help the model focus on specific regions of the image, enhancing its ability to distinguish between similar signs. Furthermore, investigating more advanced network architectures such as ResNet or EfficientNet could lead to better feature extraction and improved classification performance, especially for rare and complex traffic signs.

Additionally, another area of research is the deployment of the model in real-time systems. Testing the model in actual traffic environments, where variable lighting conditions, motion blur, and occlusions may affect the quality of images, will be crucial. Integrating the recognition system with a sensor fusion approach (such as combining camera data with radar or LiDAR) could enhance performance in dynamic real-world scenarios.

Finally, expanding the dataset to include additional traffic sign types from more diverse regions and countries could help the model generalize better to different geographical areas. By including signs from various parts of the world, the model could evolve into a robust system capable of recognizing traffic signs in international settings, making it applicable for global autonomous vehicle applications.

In conclusion, while the results are promising, further enhancements to data augmentation, network architecture, and dataset diversity are required to refine the model's performance, particularly in the context of real-world challenges. The findings from this project lay the groundwork for future research in traffic sign recognition and provide insights into the ongoing improvements needed to deploy such systems in practical applications.

VI. Conclusion

A. *Summary*

The traffic sign recognition system proposed in this project aimed to create a robust and scalable model that could accurately identify traffic signs from images. Using convolutional neural networks (CNN), the model was trained on the German Traffic Sign Recognition Benchmark (GTSRB) and tested on the European Street Sign Dataset (EURSD). Throughout the project, we focused on using Python, along with powerful libraries such as TensorFlow, Keras, and OpenCV, to preprocess images, augment data, and build the machine learning model. By incorporating techniques like image augmentation and evaluation metrics such as accuracy, precision, and recall, the system was able to demonstrate promising results.

The key challenges encountered included ensuring the system's ability to generalize across different datasets, dealing with real-world image conditions, and fine-tuning the model's hyperparameters for optimal performance. The results from our experiments showed that the model achieved high accuracy, especially in recognizing traffic signs under controlled environments. However, real-world scenarios presented some difficulties due to variations in lighting, angle, and distortion, underscoring the need for further model refinement and additional data for better generalization.

This project also involved considerable collaborative effort, with each team member contributing in specific areas: Kevin Allen took charge of project management, ensuring smooth communication and timely completion of tasks. Sahad Rafiuzzaman, Adnan Hasan, and Danielle Raynor contributed extensively to coding the system and developing the architecture, while Noah Roberson handled the final presentation, bringing together all the aspects of the project into a cohesive narrative for our audience. The teamwork and division of labor made the project efficient and effective in producing the desired outcome.

The final system is not just a functional recognition tool but also a step toward creating more sophisticated applications that could be applied to autonomous driving, traffic monitoring, and safety systems. Our work lays the foundation for developing systems that can operate in a variety of real-world environments and handle complex tasks like traffic sign detection and classification.

In conclusion, the project not only demonstrated the feasibility of building a traffic sign recognition system but also highlighted areas where improvements can be made, particularly in terms of robustness and scalability. The next steps for the project would include fine-tuning the model for better generalization, integrating it into real-time systems, and exploring new datasets to improve its accuracy across a wide variety of traffic sign types.

B. *Concluding remarks*

The success of this project reflects the power of deep learning in solving complex problems like traffic sign recognition, which is crucial for applications such as autonomous vehicles and intelligent transportation systems. While we achieved encouraging results in our controlled experiments, the challenges posed by real-world data—such as varied lighting, angles, and environmental conditions—highlight the need for continuous improvements. Further research and development in the area of data augmentation and domain adaptation could potentially address these challenges and enhance the system's reliability.

A major takeaway from this project is the importance of data diversity. To develop a system that works well in real-world conditions, it is essential to train models on diverse datasets that include variations in environmental factors. In this project, we utilized the GTSRB and EURSD datasets, but incorporating more real-world images, including images with varying light conditions, occlusions, and sign distortions, would significantly improve the model's performance.

Moreover, the use of advanced techniques such as transfer learning, where a pre-trained model is fine-tuned with a smaller set of data, could also be explored. Transfer learning has proven effective in many image classification tasks, and its integration into this project could reduce the time and resources required for training while improving accuracy.

Looking ahead, integrating this traffic sign recognition system into real-time systems such as autonomous vehicles or smart traffic monitoring networks would provide valuable real-world impact. By combining high-accuracy models with real-time data processing, these systems could greatly contribute to road safety and traffic management, making transportation smarter and more efficient.

In conclusion, while the road ahead is challenging, this project has laid the groundwork for future advancements in traffic sign recognition and autonomous systems. With further refinements in the model and additional real-world testing, the technology can be expanded and integrated into practical applications that could benefit society by enhancing safety and automation in traffic systems. The journey toward fully autonomous driving systems is ongoing, and projects like this serve as an essential step toward realizing that vision.

Acknowledgment

VII. Acknowledgments

We would like to express our sincere gratitude to all those who supported and contributed to the success of this project. This research was a collaborative effort, and we each played an important role in its development.

First, we would like to acknowledge **Kevin Allen**, our project manager, who was instrumental in keeping the team organized and on track throughout the project. Kevin's leadership ensured that we met all deadlines and maintained clear communication during our meetings, providing structure and guidance to our efforts.

Sahad Rafiuzzaman and **Adnan Hasan** were key contributors to the coding aspects of this project. Their expertise in machine learning and system architecture helped to shape the development of the traffic sign recognition model. Their work on implementing the algorithms and integrating the system was essential to the project's success.

Danielle Raynor played a significant role in writing the code, creating presentations, and preparing the project report. Danielle's efforts in organizing and documenting the process were critical to ensuring that our research was clear, well-presented, and effectively communicated to our audience.

Finally, **Noah Roberson** contributed significantly to the final presentation and recording. His ability to summarize our work into a compelling presentation helped convey the outcomes of the project effectively and professionally.

We would also like to thank **Old Dominion University** and the **School of Cybersecurity** for providing the resources and environment that made this project possible. Their ongoing support in both academic and technical fields has been essential throughout this research.

Thank you all for your dedication and hard work on this project.

References

[1] Mvd, A. (2020). Road sign detection [Data set]. Kaggle. https://www.kaggle.com/datasets/andrewmvd/road-sign-detection

[2] GeeksforGeeks. (2022, November 7). Traffic signs recognition using CNN and Keras in python. https://www.geeksforgeeks.org/traffic-signs-recognition-using-cnn-and-keras-in-python/

[3] Mykola. (2018, November 25). GTSRB - German Traffic Sign Recognition Benchmark. Kaggle. https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign

[4] Larxel. (2020, May 24). Road sign detection. Kaggle. https://www.kaggle.com/datasets/andrewmvd/road-sign-detection