Syed Azhar Hussain Quadri

CS 5330 PRCV

Final Project Report

BlindSightSense(Bs2)

Enhancing Blind Navigation with Object Detection using CNNs

According to the World Health Organization, there are currently over 285 million visually impaired individuals worldwide, with 39 million of them being completely blind. With this number expected to rise in the coming years, there is a pressing need for technological solutions that can assist the blind in their daily lives. This report aims to present a project that uses computer vision and convolutional neural networks (CNNs) to aid the visually impaired in navigating their surroundings. The project, titled "BlindSight Sense: Enhancing Blind Navigation with Object Detection using CNNs," focuses on enhancing the navigation experience for the blind by leveraging the power of CNNs for object detection. This report will detail the technical aspects of the project, including the implementation of the CNN-based object detection algorithm and its integration with a hardware stick for blind navigation. Additionally, the report will discuss the potential impact of the project in improving the quality of life for the visually impaired, as well as the challenges faced during its development.

The following are the basic steps involved in the working of this project:

1. Image acquisition: The first step is to capture an image or video stream using a camera, which serves as the input sensor. The camera can be mounted on a stick, which the user can hold and point towards different objects or signs.

2. Object or sign detection: The next step is to apply object detection algorithms to the captured image or video stream. Object detection algorithms are based on convolutional neural networks (CNNs), which are trained on large datasets of images and can detect and localize objects within an image. The algorithm will analyze the input image and identify the presence of objects or signs in the scene.

3. Object or sign recognition: Once the object or sign is detected, the system uses a pre-trained CNN model to recognize the specific object or sign. The CNN model is trained on a large dataset of labeled images, and it can classify the input image into one of several predefined categories.

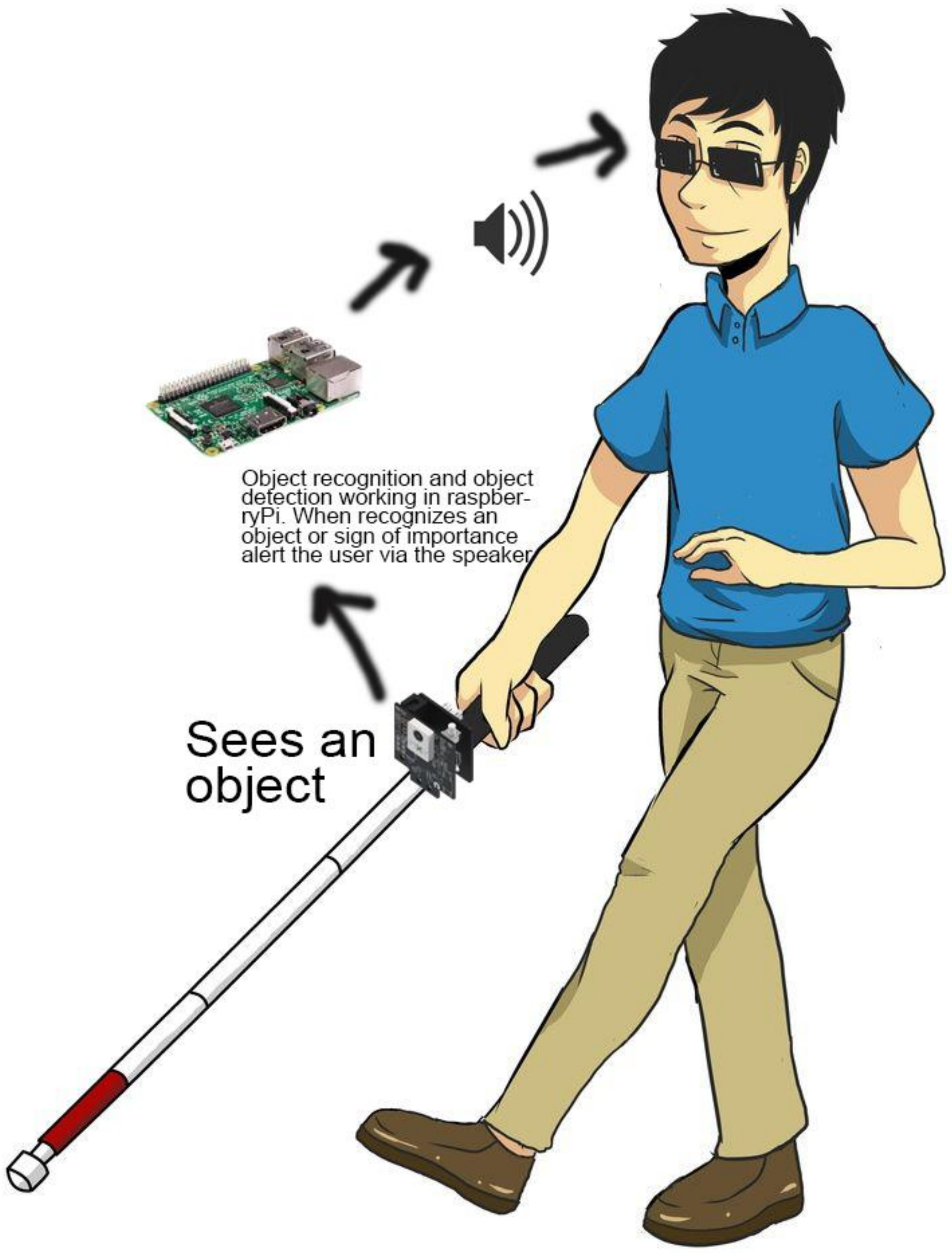
4. Audio feedback: Once the object or sign is recognized, the system will provide feedback to the user in the form of sound as an alert message via a speaker. The system can use text-to-speech (TTS) technology to convert the recognized object or sign into an audible message, which is then played through the speaker.

5. User feedback: Finally, the user can respond to the audio feedback and navigate their surroundings accordingly. They can move towards the detected object or sign, or take other actions as needed.

This project uses computer vision and machine learning to provide visually impaired individuals with real-time object and sign detection and recognition. The system is portable and can be carried on a stick, making it convenient for users to carry it anywhere they go.

Project working in a pictorial format.

The project contains a camera (I used PIXY camera) that works as an input sensor. The Raspberry Pi is used for running the code and processing the algorithms and finally it would either alert the user via vibration motor attached to the handle of the stick or speaker based on the preference of the user.
All the components will be embedded in the stick itself.


Object recognition and object detection working in raspber-ryPi. When recognizes an object or sign of importance alert the user via the speaker
Sees an object

Main Picture source: https://www.pinterest.com/pin/656118239434248151/

(Edited for educational purpose and for understanding)

MARK1 Working

This code is designed to detect a stop sign from a live video input stream using OpenCV and notify the user through sound playback.

The code first includes the necessary header files for OpenCV and sound playback using Windows.h. Then, a VideoCapture object is created to capture live video input. The code checks if the camera is opened successfully, and if not, it displays an error message and terminates the program.

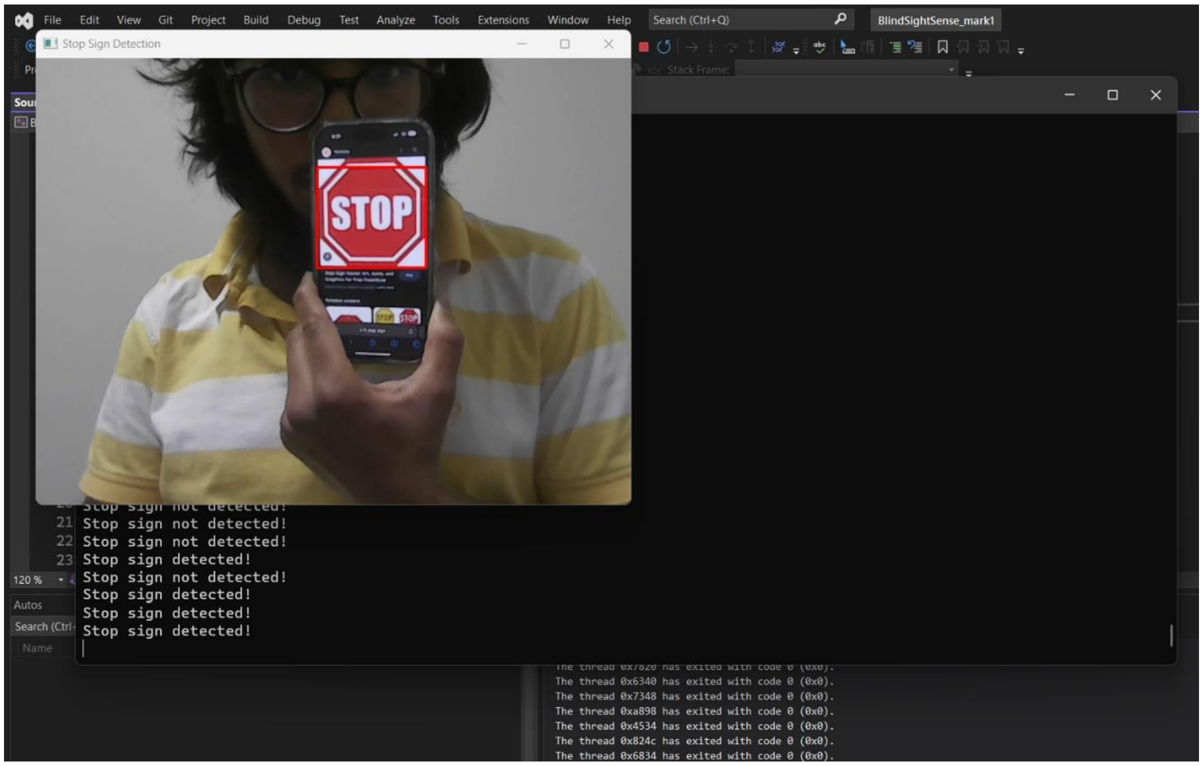
Next, the code loads the sound file to be played when a stop sign is detected. The sound file is converted to a wide string for use with the Windows API.

In the main loop, the code reads a frame from the video input, converts it from BGR to HSV color space, and applies color thresholding to extract the red regions. The code then applies morphological operations to remove noise and fill gaps in the image.

The code finds contours in the binary image and loops over each contour. If the contour area is within a certain range, the code draws a bounding box around it and prints a message indicating that a stop sign has been detected. The sound file is played through the speakers to alert the user. If a stop sign is not detected, the code prints a message indicating that it was not detected.

Finally, the video feed with the identified stop sign is displayed, and the user can exit the program by pressing the ESC key. The VideoCapture object is then released, and all windows are closed.

Overall, this code provides a useful application of computer vision and sound playback for detecting stop signs in real-time video input, which could be useful for the visually impaired.



```
Stop sign not detected!
Stop sign not detected!
Stop sign not detected!
Stop sign detected!
Stop sign not detected!
Stop sign detected!
Stop sign detected!
Stop sign detected!
```

```
The thread 0x7820 has exited with code 0 (0x0).
The thread 0x6340 has exited with code 0 (0x0).
The thread 0x7348 has exited with code 0 (0x0).
The thread 0xa898 has exited with code 0 (0x0).
The thread 0x4534 has exited with code 0 (0x0).
The thread 0x824c has exited with code 0 (0x0).
The thread 0x6834 has exited with code 0 (0x0).
```

Drawback of MARK1:

One major drawback of this approach is that it only detects stop signs that are red in color. While this is typically the case for stop signs, it is possible for stop signs to be other colors, such as yellow or even blue in certain countries.

Additionally, the code assumes that the stop sign is the only red object in the frame. If there are other red objects present in the scene, the code may detect them as stop signs and produce false positives.

Finally, the code only detects stop signs and does not provide any additional information about the scene or surroundings. This could be limiting for a visually impaired user who may require more comprehensive information to navigate their environment safely.

MARK2 focus and aim:

To overcome the first and most important drawback of the mark1, which is the limited detection of only red stop signs, a more sophisticated approach can be implemented using a Convolutional Neural Network (CNN). This approach involves training a model using a large dataset of stop sign images to recognize different variations of stop signs, regardless of their color or shape. The model can then be used to detect and classify stop signs in real-time video frames.

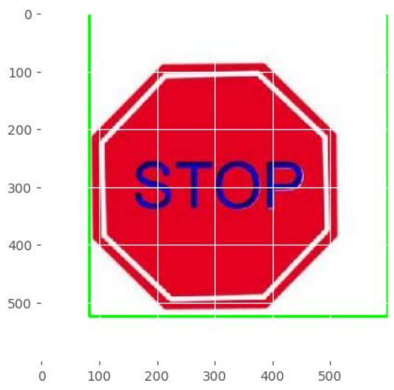
The model is designed to detect stop signs in different orientations and scales, as well as under different lighting conditions and backgrounds. The trained model is then used to perform real-time detection of stop signs in a live video stream.

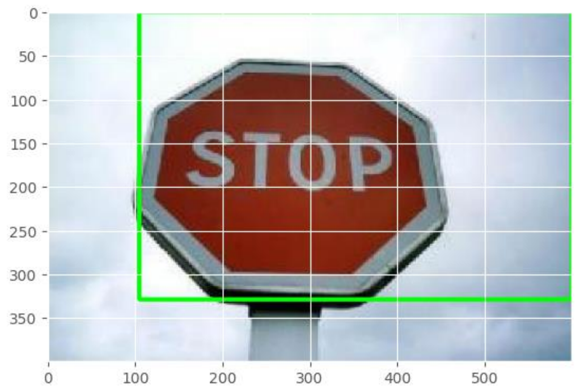
Using a CNN approach provides several advantages over the initial code. First, it allows for more accurate detection of stop signs, regardless of their color or shape. Second, it can handle situations where there are multiple red objects in the frame without producing false positives. Finally, it provides a more versatile solution that can be applied to other object detection tasks, beyond just stop signs.

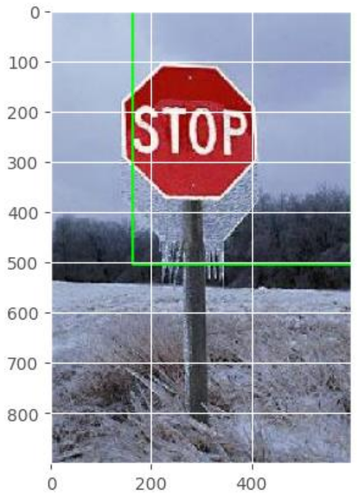
MARK2 Working

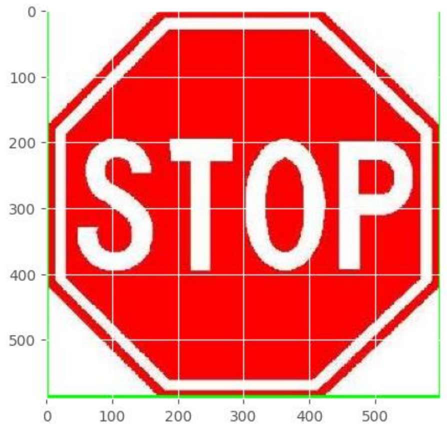
This approach works on building a stop sign detector using bounding box regression. The script reads in a set of images of stop signs and their corresponding bounding boxes from a text file, preprocesses the images and bounding boxes, trains a convolutional neural network to predict the bounding boxes for new stop sign images, and saves the model to a file. The script also generates a plot of the loss during training and predicts the bounding boxes for a set of test images, saving the predicted bounding boxes to a file.

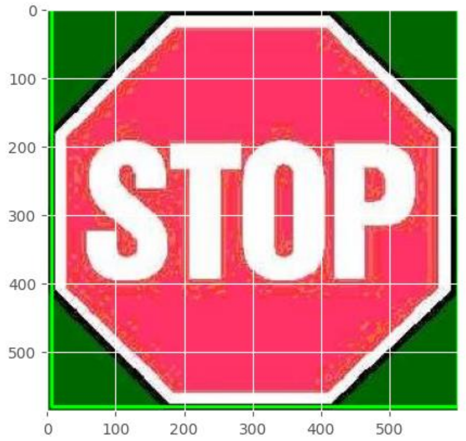
The script uses the VGG16 architecture as a base for the neural network, with the final layer replaced with a regression head consisting of four dense layers with ReLU activation and a final sigmoid activation layer. The model is trained using mean squared error loss and the Adam optimizer. The script also uses the imutils and OpenCV libraries for image preprocessing and manipulation, and the scikit-learn and matplotlib libraries for data splitting and visualization, respectively.

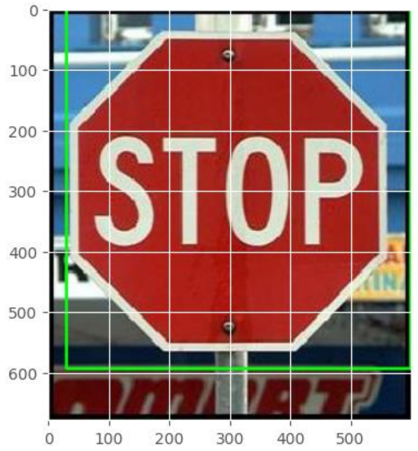










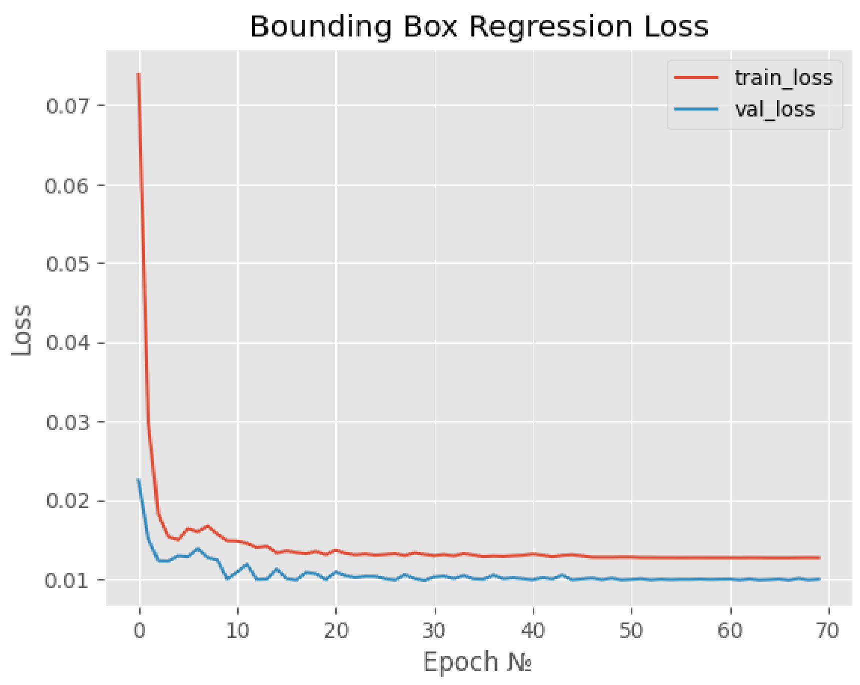


Approach:

The code reads in the annotation file that contains the coordinates of the bounding box for the stop signs in the images. It then reads in the corresponding image and scales the bounding box coordinates relative to the dimensions of the image. The images are resized to 224x224 and preprocessed to be fed into the VGG16 model. The output of the model is then passed through a series of dense layers to generate the final predictions of the bounding box coordinates.

The model is trained using the mean squared error loss function and the Adam optimizer. The loss function is used to evaluate the difference between the predicted bounding box and the ground truth bounding box. The model is trained for 70 epochs with a batch size of 32.

The performance of the model is evaluated on a test set, and the loss is plotted for both the training and validation sets over the 70 epochs. Finally, the model is used to predict the bounding box coordinates for a set of test images.

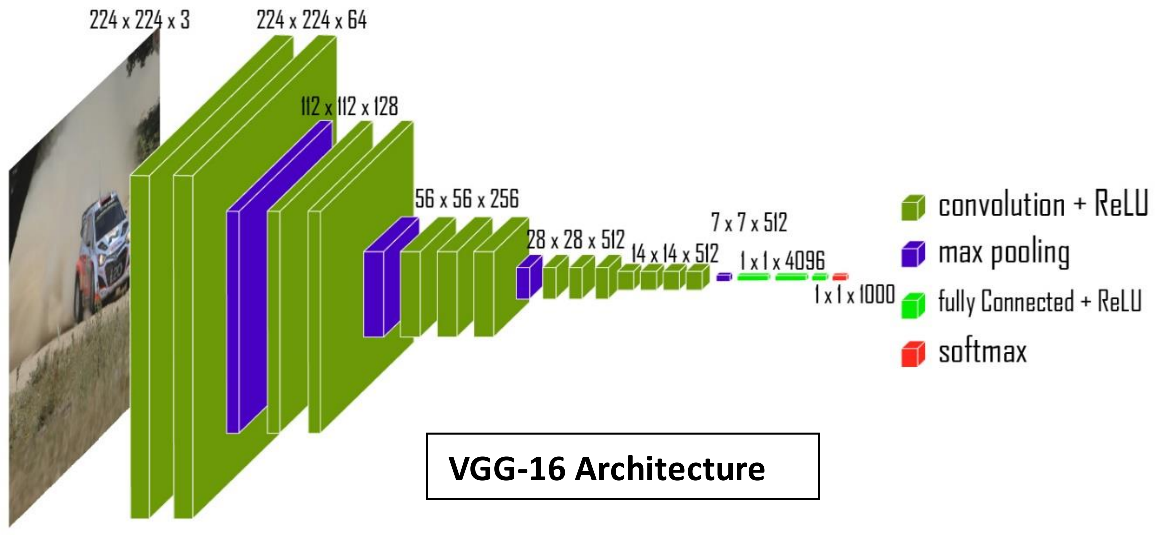


About VGG16:

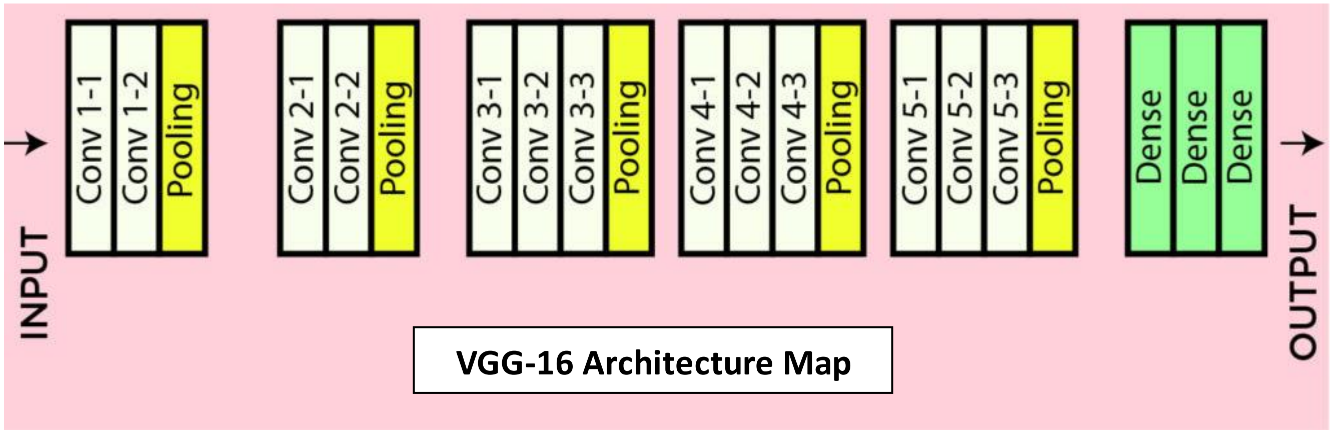
VGG16 is a convolutional neural network model that was developed by the Visual Geometry Group at the University of Oxford. It has been trained in a large-scale image classification task, i.e., to classify images into one of 1,000 different categories. VGG16 consists of 13 convolutional layers followed by 3 fully connected layers. It is a widely used model for computer vision tasks, including image classification and object detection.

Important terms

- VGG16: A convolutional neural network model that has been pre-trained on the ImageNet dataset.
- ReLU: Rectified Linear Unit is an activation function used in neural networks that sets all negative values to zero.
- Flatten: A layer in a neural network that flattens the output of a convolutional layer into a 1D array.
- Dense: A fully connected layer in a neural network.
- Input: A layer that defines the shape of the input data.
- Adam: An optimization algorithm used to update the weights of a neural network during training.
- Preprocessing: Preparing the input data in a format that can be fed into the neural network.
- Mean squared error: A loss function used to evaluate the performance of the model.
- Bounding box: A rectangular region that encloses an object of interest in an image.
- Test set: A set of images used to evaluate the performance of the model after training.



VGG-16 Architecture



VGG-16 Architecture Map

<u>MARK3 focus and aim:</u>

The Mark 1 had a limitation in that it could only detect stop signs, which was not resolved in the Mark 2. However, this issue has been addressed in the Mark 3 by incorporating a wider variety of important traffic signs.

To detect traffic signs in images, the YOLOv5 (which stands for "You Only Look Once”) algorithm was used. YOLOv5 is a well-known object detection algorithm that utilizes a single neural network for both object recognition and bounding box regression. Essentially, it takes an image as input and outputs the coordinates and labels of the objects in the image.

For this specific implementation, a pre-trained YOLOv5 model was fine-tuned on a dataset of traffic sign images and labels using transfer learning. The model was then utilized to detect traffic signs in test images.

Overall, this approach demonstrates the effectiveness of YOLOv5 in detecting traffic signs and can be useful for various real-world applications such as traffic management systems and self-driving cars.

MARK3 Working

The code is performing data preprocessing tasks for the training of an object detection model using the YOLOv5 algorithm. The main tasks performed in the code include:

- Reading and parsing annotation XML files containing the information about the location of objects in the images.
- Converting the object bounding box coordinates from the "voc" format to the "yolo" format.
- Generating the labels for each image in the format expected by the YOLOv5 algorithm.
- Splitting the data into training and validation sets.

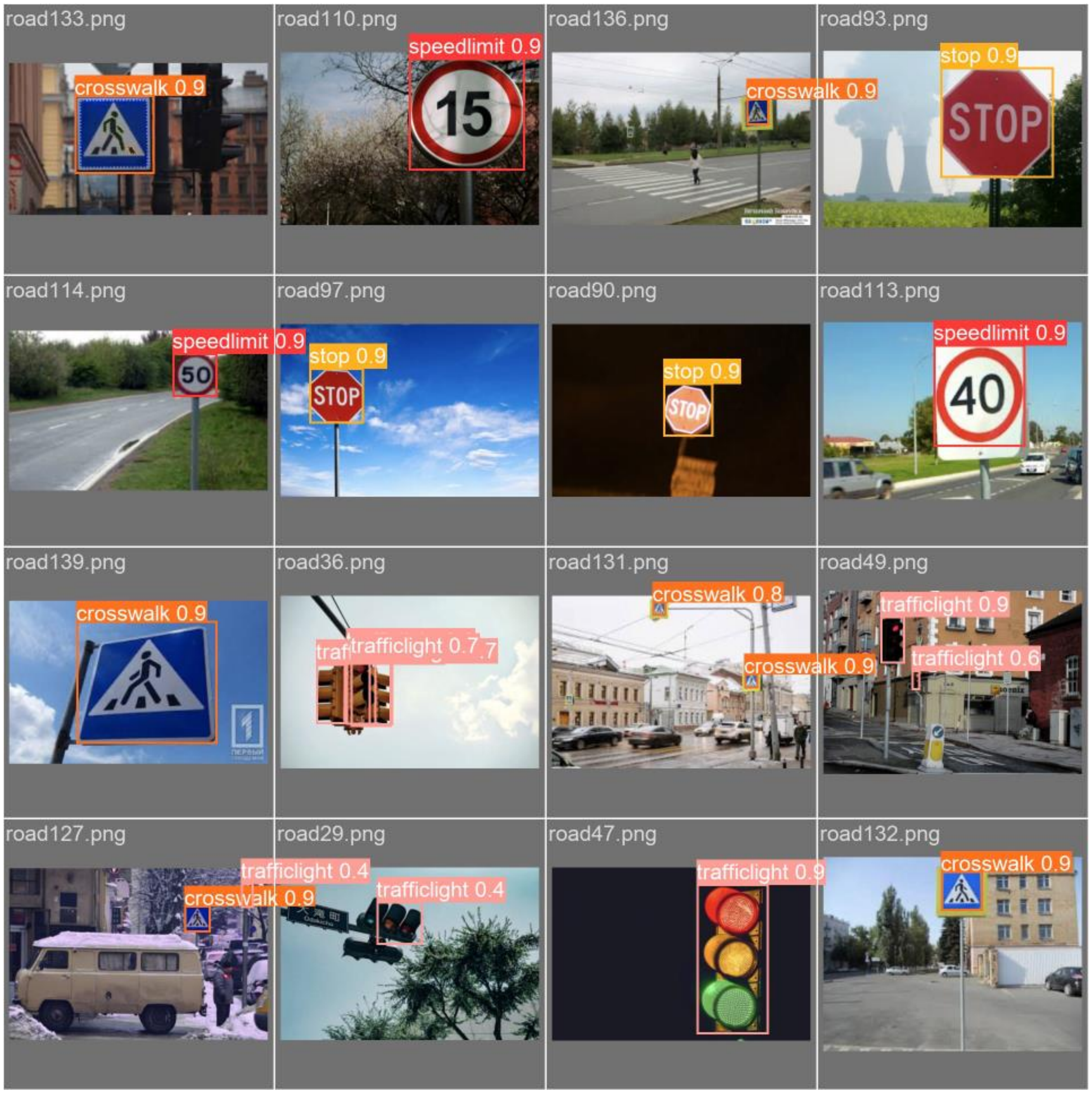
The code begins by cloning the YOLOv5 repository and installing the required Python packages. It then reads the annotations from the specified folder using the ElementTree library and extracts the required information about the image, object labels, and bounding box coordinates. This information is then stored in a Pandas dataframe for further processing.

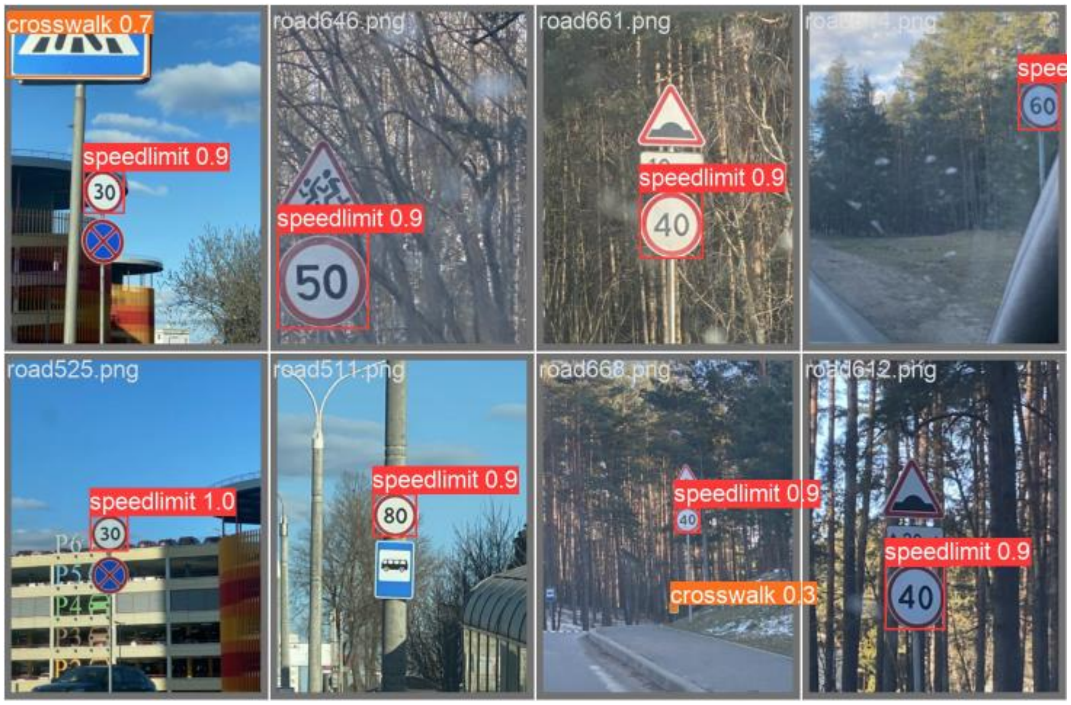
Next, the code converts the bounding box coordinates from the "voc" format to the "yolo" format. This involves calculating the center coordinates of the bounding box and normalizing them with respect to the image width and height. The class numbers associated with each object label are also added to the dataframe.

The code then generates a dictionary that contains the label information for each image. The dictionary key is the image name, and the value is a list of strings, where each string represents the label information for a single object in the image.

The label information is written to text files in the required format for the YOLOv5 algorithm. The labels are stored in a directory named "labels" in the "data" folder of the YOLOv5 repository.

Finally, the code splits the data into training and validation sets by randomly shuffling the image filenames and copying them to the corresponding "images" folder in the "train" and "val" directories. The corresponding label files are also copied to the respective "labels" folder in each directory.

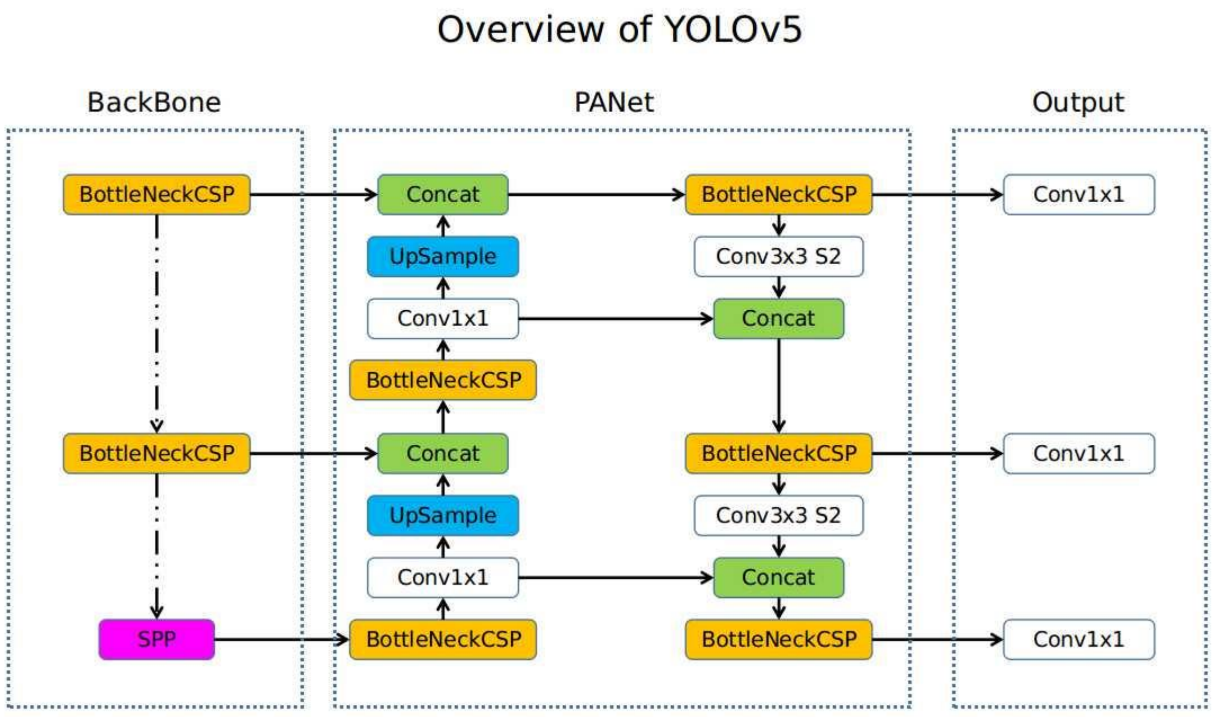

road133.png
crosswalk 0.9
road110.png
speedlimit 0.9
road136.png
crosswalk 0.9
road93.png
stop 0.9
road114.png
speedlimit 0.9
road97.png
stop 0.9
road90.png
stop 0.9
road113.png
speedlimit 0.9
road139.png
crosswalk 0.9
road36.png
trafficlight 0.7
road131.png
crosswalk 0.8
crosswalk 0.9
road49.png
trafficlight 0.9
trafficlight 0.6
road127.png
crosswalk 0.9
trafficlight 0.4
road29.png
trafficlight 0.4
road47.png
trafficlight 0.9
road132.png
crosswalk 0.9

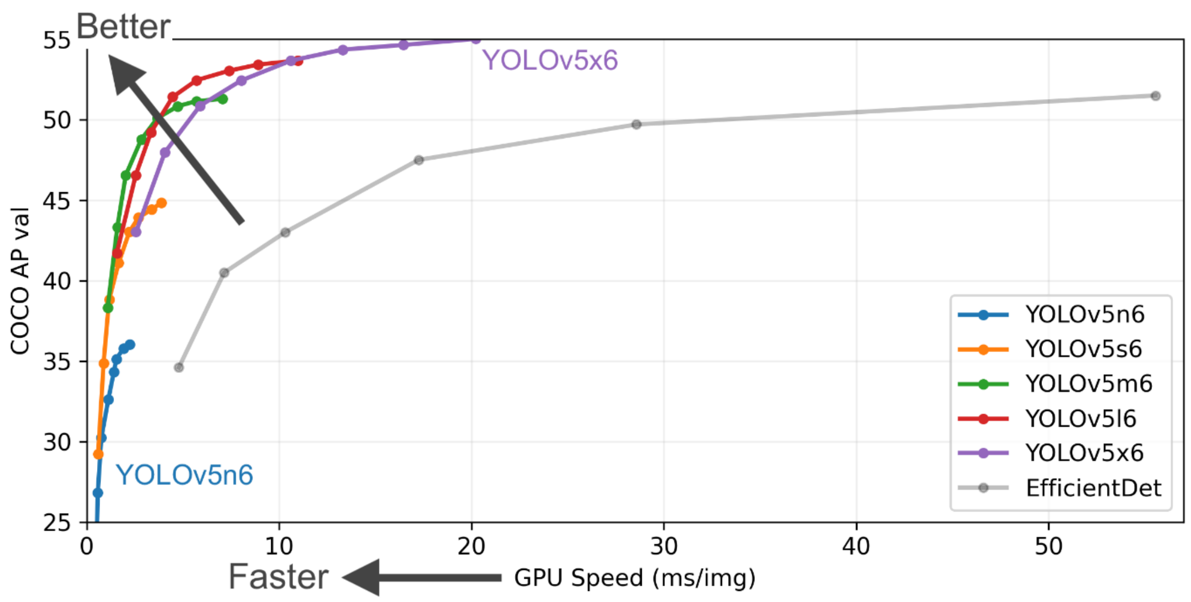

crosswalk 0.7
speedlimit 0.9
road646.png
speedlimit 0.9
road661.png
speedlimit 0.9
road525.png
speedlimit 1.0
road511.png
speedlimit 0.9
road668.png
speedlimit 0.9
crosswalk 0.3
road612.png
speedlimit 0.9

About YOLOv5:

The YOLOv5 algorithm is a state-of-the-art object detection algorithm that uses convolutional neural networks to detect and classify objects in images. The algorithm uses anchor boxes to define the shape and size of the objects and predicts the class probabilities and bounding box coordinates for each anchor box. The YOLOv5 algorithm is trained using a combination of image and label data, where the label data contains the information about the object locations and classes in the images.

The main advantage of the YOLOv5 algorithm over other object detection algorithms is its speed and accuracy. The algorithm can detect objects in real-time with high accuracy and can be trained on large datasets using modern GPUs. The algorithm has several variations, including YOLOv5s, YOLOv5m, YOLOv5l, and YOLOv5x, with varying numbers of layers and model sizes.





Learning And Reflection:

Working on this final project involving computer vision algorithms such as CNN, VGG-16, and YOLO was an exciting and challenging experience for me. These projects involved analyzing and detecting objects from images and videos, which required me to dive deep into image processing techniques and machine learning concepts.

The first version (MARK 1) of my project was simple as it only required basic knowledge of color matching. However, the limitations and drawbacks of the project motivated me to work on enhancing and advancing it.

When working on the mark2, I worked on the analysis of stop sign images using CNN and VGG-16. This involved building a deep neural network that could accurately detect stop signs in images. I had to preprocess the images, split them into training and testing sets, and train the model using the TensorFlow library. I used VGG-16, a pre-trained convolutional neural network, as the base model for transfer learning, and fine-tuned it to improve its accuracy in detecting stop signs.

For the mark3 my work was on the detection of traffic signs using YOLOv5. I had to train a custom YOLOv5 model on the traffic sign dataset, which involved labeling the images, generating the training and validation datasets, and fine-tuning the YOLOv5 architecture for the specific task of traffic sign detection. The project was challenging due to the complexity of the YOLOv5 architecture and the need to carefully tune the hyperparameters to improve the model's accuracy.

Throughout these iterations and improvements, I learned a lot about deep learning techniques, computer vision, and image processing. I also gained experience in working with large datasets, data preprocessing, and data augmentation techniques. One of the key takeaways from this project is the importance of hyperparameter tuning, as this significantly affects the performance of the models. I also learned the importance of patience and persistence when training deep learning models, as it can take a long time to train and fine-tune these models.

In conclusion, these projects provided me with a hands-on experience in computer vision and deep learning, allowing me to apply the theoretical concepts I had learned to real-world applications. I feel that I have gained valuable skills and knowledge that will be useful in future projects and have a greater appreciation for the complexity of computer vision tasks.

Sources and References:

Stack overflow

Github

Medium

Geeksforgeeks

https://builtin.com/machine-learning/vgg16

https://oyyarko.medium.com/opencv-brief-note-on-how-you-can-access-webcam-in-google-colab-d4d84efc301f

https://www.mathworks.com/help/gpucoder/ug/code-generation-for-traffic-sign-detection-and-recognition-networks.html

https://github.com/ultralytics/yolov5

https://github.com/satojkovic/DeepTrafficSign

https://builtin.com/machine-learning/vgg16

https://stackoverflow.com/questions/40821954/no-module-named-imutils-perspective-after-pip-installing

https://pytorch.org/hub/ultralytics_yolov5/#:~:text=YOLOv5%20%F0%9F%9A%80%20is%20a%20family,to%20ONNX%2C%20CoreML%20and%20TFLite.