Road Sign Recognition

Maneesha Kondisetty - 16316850
Srujana Sadineni - 16307516
Shravya Muthyala -16320880
Harshavardhan Nakirakommula -16307704

SCHOOL OF COMPUTING AND ENGINEERING
COMPUTER SCIENCES

December 12, 2021

Abstract

The process of automatically detecting traffic and road sign indicators along the road, such as speed limit signs, yield signs, merge signals, and so on, is known as road sign classification. We can construct "smarter cars" if we can recognize road signs automatically. In order to effectively parse and understand the roadway, self-driving cars require traffic sign recognition. Similarly, "driver alert" systems in cars must comprehend the road environment in order to assist and protect drivers. Computer vision and deep learning can help with a variety of issues, including road sign recognition. We will create a deep neural network model that can classify traffic signs in an image into several categories in this project. We can read and interpret them with this model, which is a critical task for all autonomous vehicles.

Contents

1 Introduction

Using Tensorflow, I created and trained a deep neural network to classify German traffic signals for this project. Road sign detection is a high-priority computer vision problem that underpins a wide range of industrial applications, including automotive. We will create a deep learning algorithm that will train on German traffic sign photos and then categorize unlabeled traffic signs in this challenge. The deep learning model will be developed with Keras (tensorflow's high-level API), and we'll learn how to preprocess images with OpenCV and employ a cloud GPU service provider. For our algorithm development, we will use Keras. Keras was chosen because it is simple to learn and implement. Keras is also well-integrated with TensorFlow. Following Tensorflow, Keras appears to be the most extensively used deep learning framework.
Technologies Used:
-Python
-Deep Learning

2 Problem Statement

The primary purpose of this road sign identification project is to solve challenges using machine learning and deep learning.

3 Algorithmic process

We'll follow the steps outlined below, just like any other machine learning model development process:
-Understand the data
-Preprocess the data
-Build the model's architecture
-Test the model -Repeat the process till you get the best results
-For a better user interface, used a graphical user interface.

4 Relevant Work

For the classification of the road signs, we used CNN i.e., convolution neural networks using sequential model. We took German Traffic sig n dataset, with which we trained the model for the purpose of deciphering the traffic signs from the pictures. Then we tested the model with unseen data of traffic signs. After finding the training and testing accuracy we checked for overfitting and tried to decrease it. Later we made some prediction to check the if the model is predicting the correct label for the images.

5 Datasets

We choose German Road Sign Images Dataset for this model. We did train and test data split for the dataset. This data set includes images of all the road signs. This is an open-ended assignment; examples include charting traffic sign images, counting the number of signs, and so on. For doing visualizations in Python, the Matplotlib examples are a wonderful resource. We are training the data with 43 different road signs, indeed they are the 43 classes we used for our model. We have encoded them with label encoder i.e, tocategorical()

6 Data Preprocessing

Data processing acts as a major step in natural language processing for converting the imported data to machine understandable format. The more the data is generalized the more the performance. This preprocessed data is sent to the model to determine the accuracy and the performance. Here all the images are rescaled to a size of 0 to 1 range, converted to categorical data and the manually imported photos are enlarged according to the input data.

7 Model Deployment

A web-based application (Graphical user Interface) is created to illustrate the resilience and compatibility of the provided model. This development clearly demonstrates that this model can be useful for automated cars for the recognition of the road signs automatically.

8 Model

1. The libraries and data are being imported.
2. Don't delve too deeply at the data.
3. Creating a CNN model and determining its correctness
4. Plots of accuracy and loss for train and test data
5. Saving the model.
6. Using the above sequential paradigm, create a graphical user interface.
7. tkinter creation:
8. Prediction function
9. Design
10. Show the forecast
11. Add an image to your image prediction

We used ffmpg to extract frames from the video.

ffmpeg: This is software that contains libraries and logics for converting frames to videos or processing videos to frames.

9 Using GUI:

GUI, Graphical User Interface is an operating system based on graphics that uses mouse, menus and icons that manages the user interaction with the system.

There are multiple options in python for implementing this. Tkinter is the most used method out of all. It is the easiest and fastest way to create multiple GUI applications.

In this project we were using GUI to create a simple interface between the user and system. User will be having a button on the screen where they can upload an image from their system. After uploading the image, the use can be able to preview the uploaded image with a predict button followed by it.

Then the user can click on predict button which will show the name of the input image sign.

We preprocessed the GUI window. We created methods upload_sign_board(), display_prediction(), prediction(). On click on upload button, upload_sign_board method will be called which takes the path of the image but using askopenfilemethod. Then we displayed the image with photo image method and called the display prediction method. In this we added some UI styles to the button and called prediction method. This will take the path of the input image which is passed to the model to predict the value, which will be passed to classes dictionary to get the label for the image uploaded.

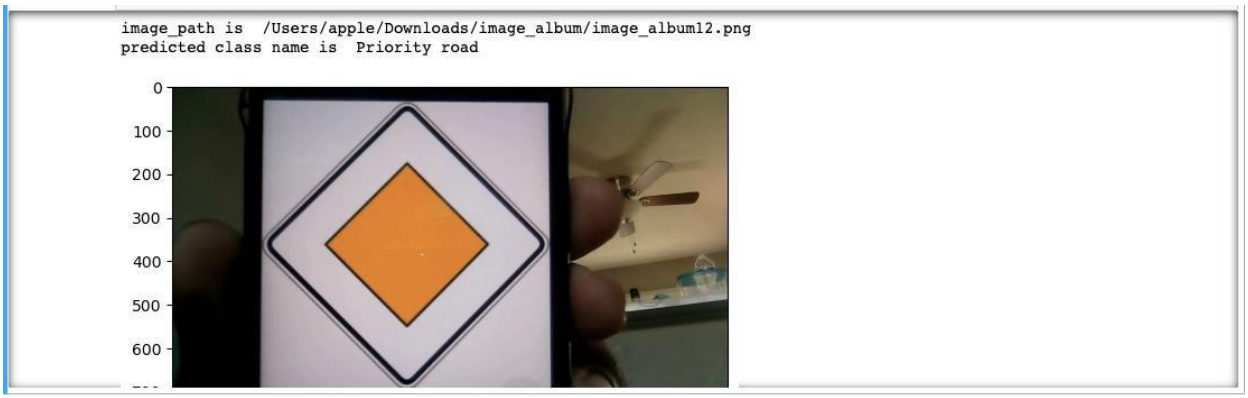
1. Design

```
[ ] #https://realpython.com/python-gui-tkinter/

    window=tk.Tk()
    window.geometry('800x600')
    window.title('Traffic sign board detector')
    window.configure(background='#466df0')
    label=Label(window,background='#466df0', font=('arial',15,'bold'))
    sign_image = Label(window)
```

```
[ ] #Uplolading the image and classifying the type of image
    upload=Button(window,text="Upload an image",command=upload_sign_board,padx=10,pady=5)  #Button configuration
    upload.configure(background='#466df0', foreground='yellow',font=('calibri',10,'bold'))
    upload.pack(side=BOTTOM,pady=50)
    sign_image.pack(side=BOTTOM,expand=True) #Button location
    label.pack(side=BOTTOM,expand=True)
    heading = Label(window, text="Predict the traffic sign",pady=20, font=('calibri',20,'bold'))
    heading.configure(background='#466df0', foreground='white')
    heading.pack()
    window.mainloop()
```

For Video :



10 Tkinter

Tkinter is the only GUI framework included in the Python standard library. Tkinter is endowed with a number of advantages.The window is the most fundamental part of a Tkinter GUI. All additional GUI elements are contained within Windows. Widgets are other graphical user interface elements like text boxes, labels, and buttons. Widgets are little pieces of software that can be placed into windows. A window() is an instance of tkinter class. When you. pack() a widget into a window, Tkinter shrinks the window to the smallest size possible while still enclosing the widget completely. So, we added multiple widgets like buttons and images to our window. A button "Upload image" is used to upload an image from the local desktop. Functionalities for this have been assigned appropriately.

Our model is evaluated with test data from the dataset which is unseen by the model while training. We have also recorded a video and converted it to frames (.png format) and these images were evaluated using our model. We were able to get the appropriate results for this data.

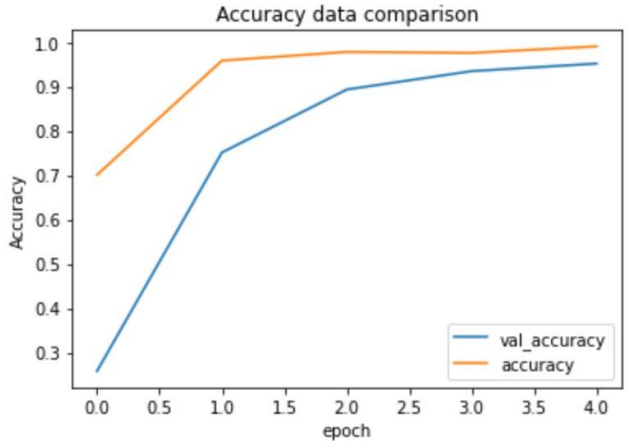
11 Model Evaluation

Following are the attachments for model evaluation, accuracy and loss plots.

```
Epoch 1/5
123/123 [==============================] - 93s 748ms/step - loss: 3.6135 - accuracy: 0.2582 - val_loss: 1.2857 - val_accuracy: 0.7006
Epoch 2/5
123/123 [==============================] - 90s 735ms/step - loss: 0.9293 - accuracy: 0.7512 - val_loss: 0.2108 - val_accuracy: 0.9584
Epoch 3/5
123/123 [==============================] - 100s 811ms/step - loss: 0.4106 - accuracy: 0.8933 - val_loss: 0.1212 - val_accuracy: 0.9781
Epoch 4/5
123/123 [==============================] - 98s 796ms/step - loss: 0.2502 - accuracy: 0.9347 - val_loss: 0.1062 - val_accuracy: 0.9758
Epoch 5/5
123/123 [==============================] - 128s 1s/step - loss: 0.1861 - accuracy: 0.9518 - val_loss: 0.0512 - val_accuracy: 0.9904
```

For Model Accuracy:

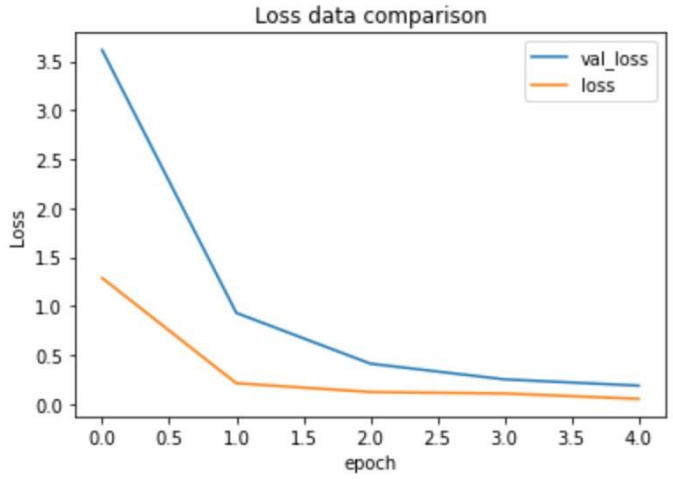
```
# Plotting the Accuracy for both training data and validation data using the history object.
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['val_accuracy', 'accuracy'], loc='lower right')
plt.title('Accuracy data comparison')
plt.ylabel('Accuracy')
plt.xlabel('epoch')
plt.show()
```



For Loss Accuracy:

```
# Plotting the loss for both training data and validation data using the history object.
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Loss data comparison')
plt.legend(['val_loss', 'loss'], loc='upper right')
plt.xlabel('epoch')
plt.ylabel('Loss')
plt.show()
```

...



Output Predictions

12 Conclusion

- Created a model which can predict the label of the sign with accuracy of 98%
- Graphical User Interface was presented which will have a connection between user and system, where user can insert the image and the model can predict the sign of the image.
- Using grid we developed a model where video can be converted to images which are passed into the model for the prediction of the sign in the image.
- This model can we better suitable for the automated cars which can detects the signs and represent the label in the user understandable language.

13 References

-Training data : https://www.kaggle.com/shivamsinghal1012/traffic-sign-data-set
-https://realpython.com/python-gui-tkinter/
-https://www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign
-https://www.kaggle.com/syamkakarla/traffic-sign-classification-using-resnet
-https://www.kaggle.com/valentynsichkar/traffic-signs-classification-with-cnn#%F0%9F%8F%97%EF%B8%8F-Building-set-of-models-of-CNN-with-Keras