SignReq: Traffic Sign Image Classification with ML Models

Khadeejah Hossain
Oakland University
Rochester, MI
khossain2@oakland.edu

Mounika Kanakanti Vangala
Oakland University
Rochester, MI
mounikakanakant@oakland.edu

Abstract

We propose the best approach for a Traffic sign recognition system with a high accuracy rate and less computing time. This process is done with the assistance of CNN and Keras. In fully autonomous driving vehicles, recognizing traffic signs with minimal computing time and a high accuracy rate is a very challenging, yet an important task. So, to address this problem, our approach is to first explore the sample traffic sign dataset. Next, images are sorted and their labels are set into a list and those lists are converted into NumPy arrays for feeding to the model. Secondly, the CNN model is built to classify the images into their respective categories, this is the best approach for image classification. After building the model, the model is trained, validated, and tested using the test dataset. Finally, the graphical user interface is built for traffic sign recognition using Tkinter. The results of the traffic sign recognition model provided a 95% accuracy in identifying traffic signs . We successfully achieved our project goal of developing a CNN model that can detect traffic signs with a high accuracy percentage.

1. Introduction

The research focuses on the recognition of traffic signs using a Convolutional Neural Network (CNN) and the Keras library. The importance of this problem lies in its relevance to Advanced Driver Assistance Systems (ADAS) and the development of autonomous vehicles. Accurate traffic sign detection is crucial for vehicle safety, as these signs convey essential information about road conditions and regulations. The CNN model, implemented with Keras, achieves traffic sign classification with over 95% accuracy. This deep learning project addresses the need for precise recognition of traffic signs, a key component in ensuring road safety and supporting advancements in autonomous driving technology

2. Related Work

There are multiple publications that have explored utilizing convolutional neural network models for traffic sign image classification. One such research is Liu Shangzheng's research paper entitled "A Traffic Sign Image Recognition and Classification Approach Based on the Convolution Neural Network." Liu's research

provides an insight into an improvised version of CNN that can detect traffic signs with an accuracy rate of 99-100%. The proposed model implements the HOG feature extraction principle in order to perform image pre-processing and manipulation. The final model includes a HOG extraction followed by a CNN architecture to form a HOG-CNN architecture.

The HOG Feature Extraction is a multi-step process that starts off by modifying all the images into grayscale and then using a gamma correction algorithm to remove all image shadows and highlights that are a direct result of graying the image (Liu, 2019). Next, the image gradient is calculated and each image is divided into equal sized square cell units consisting of n x n pixels.The feature vectors are then extracted and the cell units are combined to create a large connected interval that overlap one another. Once this is complete, the HOG features are obtained and this overlapped interval forms the basis of the HOG features. Liu's research revealed that the HOG-CNN model has a higher ability to detect traffic signs than a model consisting of only CNN model; this is primarily due to it's ability to reduce false detection within images (Liu, 2019). Furthermore, Liu's experimentation revealed that the most optimal training occurs when the training set contains weight and bias learning rate factors of both 20, 20 20.

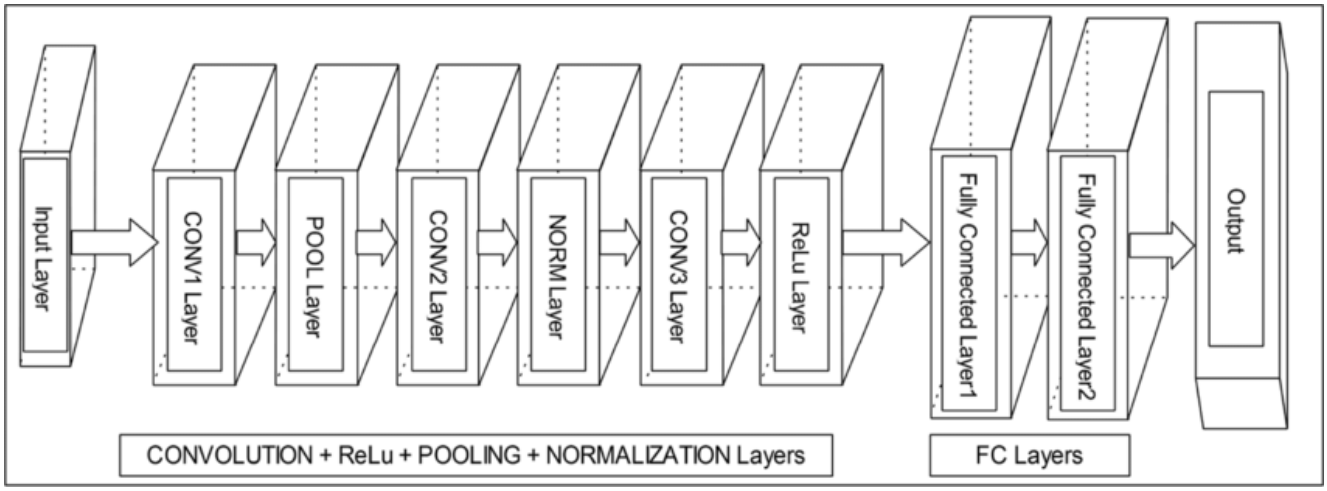
The data from this implementation revealed that the optimal accuracy percentage is obtained when the CNN model used three convolution neural layers(each layer consisting of a multi-layer structure of 5 x 5, 3 x 3, and 3 x 3 convolution cores), two Max Pooling layers, and two fully connected layers. Our CNN implementation approach is similar to Liu's since we also utilize an epoch of 20 and found it to provide the best accuracy rate. Our experimentation with various amounts of epoch layers revealed that 20 epochs provided the best accuracy rate. Although the CNN model is the most popular image recognition architecture within the computer vision field, the latest research within computer vision suggests that the Transformer model has better performance than CNN. In fact, the Vision Transformer model proved to be one of the only models that had performance similar to CNN or exceeded its performance (Chen, Fan, & Panda, 2021, pp 358). One downside to the Vision transformer is that it requires large datasets to perform optimally (Chen, Fan, & Panda, 2021, pp 358). The vision transformer model uses a sequence of embedded images and has multiple layers. Nevertheless, the transformer model is difficult to implement and many models require preprocessing of data, which was a very time consuming process, which would be out of the scope of this project and given our time constraints, we wouldn't be able to develop within the limited time frame.

3. Methods

The approach for solving the problem of traffic sign detection and identification involves the utilization of a convolutional neural network (CNN) combined with the Keras library. The key steps include image preprocessing, CNN architecture implementation, model training, and the development of a graphical user interface (GUI) for practical use.

The CNN-based approach with Keras is suitable for traffic sign recognition due to its adeptness at image processing, deep learning capabilities for automatic feature extraction, and consistently high recognition rates. The adaptability of CNNs to varied perspectives and their efficiency in model training with Keras contribute to their effectiveness. The consideration of new datasets, region-specific recognition, and the development of a user-friendly GUI enhance the approach's practical applicability. Overall, the reported success and the alignment of CNN strengths with image recognition tasks justify the appropriateness of this approach.

Various CNN architectures are used in the studies, such as a modified LeNet-5 network and a CNN architecture with different settings. The choice of CNN is crucial for achieving high recognition rates, and in one case, the proposed CNN architecture attains 100% precision in experiments. We'll use a CNN model to classify the photos into their appropriate groups (Convolutional Neural Network). For picture categorization, CNN is the best option.



Alternative approach:

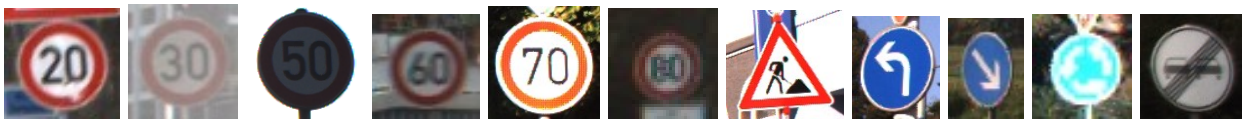
Initially we implemented an image classification model using CNN model. Further research into the topic revealed that the vision transformer model performs image classification with up to 4 times greater accuracy than the CNN model(). Our research aims to analyze and implement both the CNN architecture, nevertheless, we have considered alternative models.Throughout our project, we considered alternative approaches, including the transformer model. However, the implementation of the transformer model proved to be rather challenging, as transformer models require various

4. Data

The dataset utilized within our CNN image classification architecture was retrieved from **Kaggle's "German Traffic Sign Recognition Benchmark"** Competition. This dataset contains more than 50,000 images that can be used to train our model. For simplicity and ease, the dataset has initially already been divided up into 3 sections: a test dataset, a train dataset, and a meta dataset.The test dataset consisted of a total of 12,631 images, while the multi-class training dataset consisting of 42 classes of traffic signs. Meanwhile, 45 images made up the meta dataset. The test dataset consists of 39,209 images. Each of the images were in a .png format and varied in image size.

Prior to the CNN model implementation, the test and training images have been resized with dimensions of 30 x 30 pixels. Since the CNN model requires that we feed the model with a consistent image input, we have resized the test and train datasets into a uniform, consistent image size of 30 x 30 pixels. In fact, according to Hashemi (2019), defining a larger image size for the CNN image classification is beneficial because it reduces the need for image compression, which in turn leads to higher accuracy due to the decreased feature deformations. We strategically chose this dataset due to it being well-documented and clean data, minimizing the need to take extra pre-processing, filtering, and data cleaning steps. For our training, we utilized varying batch sizes and determined that a training batch size of 64 provided optimal results.

Example training data :



5. Experiments

Because we have several classes to categorize, we compile the model with Adam optimizer, which performs well, and the loss is "categorical_crossentropy." After constructing the model architecture, we use model.fit to train the model ().

We tried batch sizes of 32 and 64 and 128. With 64 batches, our model fared better. The accuracy was stable after 15 epochs. On the training dataset, our model had a 95% accuracy rate. We visualize the graph for accuracy and loss using matplotlib. 4991 The details connected to the image path and their appropriate class labels are contained in a test.csv file in our dataset. Using pandas, we extract the image path and labels. Then, in order to forecast the model, we must scale our photographs to 30 x 30 pixels and create a NumPy array with all of the image data. We used the accuracy score from sklearn.metrics to see how our model predicted the real labels. In this model, we were able to attain a 95% accuracy rate. Now we are going to build a graphical user interface for our traffic signs classifier with Tkinter. Tkinter is a GUI toolkit in the standard python library.

To understand the influence of epochs within our CNN architecture, we have experimented with varying levels of epochs. This helped us conclude that a size of 20 for epoch was the sweet spot since it resulted in the lowest loss of 19%, a 95% accuracy, and a value accuracy of 98.88%. A summary of our findings is listed in the table below.

Number of Epochs	Loss Rate (%)	Accuracy(%)	Value Accuracy(%)
15	19%	94.93%	97.92%
18	29.32%	94.04%	98.88%
19	22%	94.69%	98.53%
20	19.31%	95.88%	98.83%
21	28%	94.56%	98.20%
23	29%	94.21%	98.36%
25	27%	94.22%	98.44%

```
# Experimenting model with different batch_sizes
```

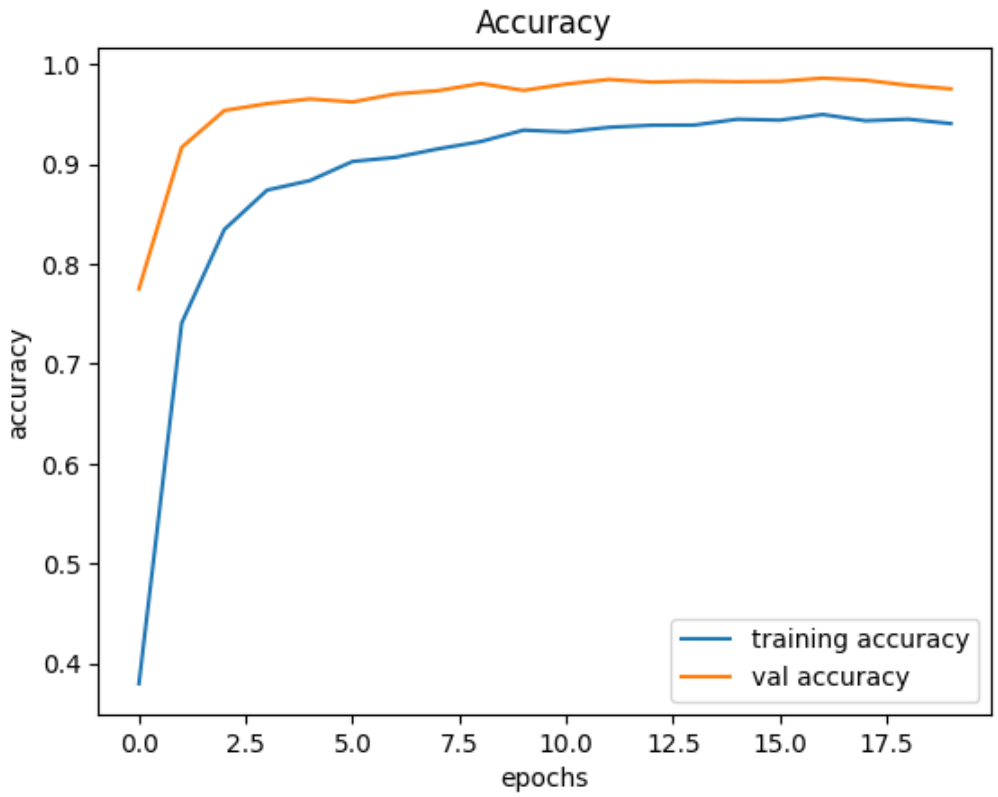
```
## Experimenting
epochs = 20
batch_size = [16, 32, 64, 128]
# history_train_accuracy = []
# history_train_accuracy = []
# history_training_loss = []
# history_val_loss = []
prediction_score = []
for batch in batch_size:
    history = model.fit(X_train, y_train, batch_size=batch, epochs=epochs, validation_data=(X_test, y_test))
#     history_accuracy.append(history.history['accuracy'])
#     print('Trainng Accuracy    : ',history.history['accuracy'])
#     print('Training Loss       : ', history.history['val_accuracy'])
#     print('Validation Accuracy : ',history.history['accuracy'])
#     print('Validation Loss     : ', history.history['val_accuracy'])
    Y_pred = np.argmax(model.predict(X_test_actual),axis=1)
#     Y_pred = model.predict(X_test_actual)
    prediction_score.append(accuracy_score(label, Y_pred))
    print('Batch size          : ', batch)
    print('Prediction Score    : ', accuracy_score(label, Y_pred))
```

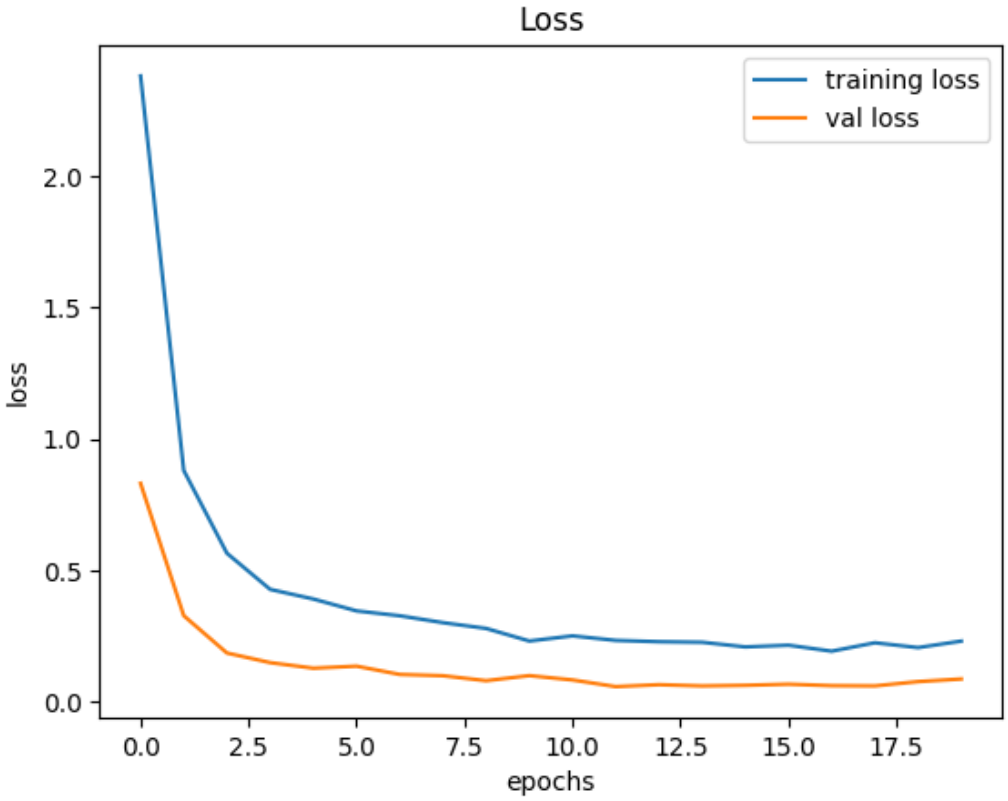
```
Epoch 20/20
31367/31367 [==============================] - 16s 499us/step - loss: 0.6999 - accuracy: 0.8775 - val_loss: 0.1841 - val_accura
cy: 0.9607
Batch size          :  16
Prediction Score    :  0.9280285035629454
Train on 31367 samples, validate on 7842 samples


Epoch 20/20
31367/31367 [==============================] - 9s 275us/step - loss: 0.5012 - accuracy: 0.9146 - val_loss: 0.3007 - val_accurac
y: 0.9328
Batch size          :  32
Prediction Score    :  0.8894695170229612
Train on 31367 samples, validate on 7842 samples


Epoch 20/20
31367/31367 [==============================] - 5s 171us/step - loss: 0.2669 - accuracy: 0.9510 - val_loss: 0.0906 - val_accurac
y: 0.9872
Batch size          :  64
Prediction Score    :  0.9551860649247823
Train on 31367 samples, validate on 7842 samples


Epoch 20/20
31367/31367 [==============================] - 4s 127us/step - loss: 0.2382 - accuracy: 0.9566 - val_loss: 0.0726 - val_accurac
y: 0.9857
Batch size          :  128
Prediction Score    :  0.9522565320665083
```


Accuracy
accuracy
epochs
training accuracy
val accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Loss
loss
epochs
training loss
val loss
2.0
1.5
1.0
0.5
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

GUI of our project

The GUI provides an interface to classify any uploaded traffic sign image.

1. The image can be browser and selected using the the “Upload an image tab”
2. Upon successfully uploading the image when we click on the classify image, the GUI uses the CNN model in the background and classifies the image and then shows the classification class.
3. The two screenshots below show the classification of the uploaded image for “Speed limit (30km/h)” and “Right-of way at intersection”

6. Conclusion

The key results indicate that the CNN-Keras approach for traffic sign recognition achieved high accuracy rates, surpassing 96.8% in some instances. The integration of advanced techniques, such as color-based segmentation and Hough Transform, contributed to improved performance. The successful development of a user-friendly GUI using Tkinter further enhances practical usability. Throughout this project, we have learned about the functionality of

Ideas for future extensions or new applications could include:

Real-time Implementation: Extend the model for real-time traffic sign recognition, enabling it to process video streams and provide instantaneous feedback to drivers or autonomous vehicles.

Continued Model Optimization: Continuously optimize the CNN architecture, experiment with different neural network structures, and fine-tune hyperparameters to further improve accuracy and efficiency.

If given additional time, we would expand our model to develop a CNN with Attention model. Additionally we would experiment with the Transformer model. Thus, we would have three different implementations of the Traffic Sign Recognition models, which would allow us to further perform performance analysis to determine which model performs best for traffic sign recognition. This would allow us to

7. Supplemental Material

- Github Link : Mounika266/Traffic-Sign-Classifier-Project (github.com)

8. References:

1. Boesch, G. (2023, November 9). Vision transformers (Vit) in image recognition - 2024 guide - viso.ai. Retrieved from https://viso.ai/deep-learning/vision-transformer-vit/#:~:text=However%2C%20recent%20advancements%20in%20transformer,results%20in%20image%20classification%20tasks

2. Chen, C. R., Fan, Q., & Panda, R. (2021). CrossViT: Cross-attention multi-scale vision transformer for image classification. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. doi:10.1109/iccv48922.2021.00041

3. Q. Li, W. Cai, X. Wang, Y. Zhou, D. D. Feng and M. Chen, "Medical image classification with convolutional neural network," 2014 13th International Conference on Control Automation Robotics & Vision (ICARCV), Singapore, 2014, pp. 844-848, doi: 10.1109/ICARCV.2014.7064414.

4. Shustanov, A., & Yakimov, P. (2017). CNN design for real-time traffic sign recognition. *Procedia Engineering*, *201*, 718-725. doi:10.1016/j.proeng.2017.09.594

5. Hashemi, M. (2019). Enlarging smaller images before inputting into convolutional neural network: Zero-padding vs. interpolation. *Journal of Big Data*, *6*(1). doi:10.1186/s40537-019-0263-7 (https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0263-7#:~:text=Since%20neural%20networks%20receive%20inputs,and%20patterns%20inside%20the%20image.)

6. https://tinyurl.com/yc9acxmt

7. Traffic Signs Recognition using CNN and Keras in Python (analyticsvidhya.com)

8. Dataset:
    a. GTSRB - German Traffic Sign Recognition Benchmark (kaggle.com)