ECE 579 Intelligent Systems, Winter 2023

Final Project Report

Project Title : Traffic Sign Classification

Name of the Student and their Responsibilities :

1. **Tharun Reddy Pyayala**

Department name : Computer and Information Science Department

Responsibilities : Exploring the data, Importing the required libraries, Learning about the previous studies on our project, Learning about the CNN model, Started Building the CNN model, Building the CNN model, Summary and Compilation of the CNN, Testing our model with our test data.

2. **Mythresh Neerugattu**

Department name : Computer and Information Science Department

Responsibilities : Exploring the data, Study about the libraries required for our project, Importing the dataset, Learning about the CNN model, Knowing about the loss and accuracy of the model, Predicting the results of our model, Fine-tuning of model.

3. **Pavan Kumar Kalisetty**

Department name : Computer and Information Science Department

Responsibilities : Exploring the data, Learning about the previous studies on our project, Information gathering on CNN, Splitting the data into train data set and test data set, Correcting the errors in our model, Training our model, finishing our model.

1. Introduction

The topic of this project is "Traffic Sign Classification", In our project we used Deep Learning Techniques like CNN and YOLO for the German Traffic Sign Recognition Benchmark (GTSRB) dataset.

Traffic sign classification is an essential task in autonomous driving, advanced driver-assistance systems, and intelligent transportation systems. The GTSRB dataset is a widely used benchmark dataset for traffic sign classification algorithms. It consists of over 50,000 traffic sign images captured under different conditions and contains 43 different traffic sign classes, such as stop signs, speed limit signs, and yield signs. Deep learning techniques, particularly convolutional neural networks (CNNs), have shown promising results in various computer vision tasks, including traffic sign classification.

In this report, we present a deep learning-based approach for traffic sign classification using GTSRB dataset. We start by providing a overview of the description of the technologies related to our project. We then describe about our proposed model architecture, which is based on a CNN with multiple layers, and then discuss the training and evaluation process. We also compare our results with the models we have done that is CNN and CNN with more pre-processing and then show that which approach achieves high accuracy in classifying traffic signs. Finally, we conclude with a discussion of the limitations and future directions of our work.

You Only Look Once (YOLO): We actually wanted to try YOLO which is an object detection algorithm but later we came to know that YOLO is realtime algorithm which can detect objects in real-time by processing the entire image in a single forward pass of a convolutional neural network. YOLO is fast and accurate and has been used in various computer vision tasks, including traffic sign detection.

So, finally we are doing our project using CNN but performing more pre-processing techniques and comparing them.

2. Description of technologies related to your project

Convolutional Neural Networks (CNNs): CNNs are a type of deep learning architecture that have given excellent performance in image recognition tasks, including traffic sign classification. CNNs use convolutional layers to extract features from the input images, which are then passed through fully connected layers to classify the images.

CNNs consist of multiple layers, including convolutional layers, pooling layers, and fully connected layers. Convolutional layers are responsible for learning local patterns and features in an image, while pooling layers help to reduce the dimensionality of the input. Fully connected layers are used to combine the learned features and produce the final classification output.

CNNs are well-suited for image classification tasks because they can automatically learn and extract features from raw input data. This means that they can learn to identify patterns and features in the images that are difficult for humans to recognize . With sufficient training data and computing resources, CNNs can achieve state-of-the-art performance on a wide range of image classification tasks.

CNNs have been used in a variety of applications, including autonomous vehicles, medical image analysis, and facial recognition. There are many open-source libraries and frameworks available for building and training CNNs, such as TensorFlow, PyTorch, and Keras, which make it easier for researchers and developers to implement and experiment with these powerful models.

3. Methods used in our project

Methods used in your project:

We used a CNN deep learning approach. First, we trained a CNN model on the GTSRB dataset using our own CNN model with 30x30 pixels which we build for our specific task, Later we performed few more pre-processing techniques like resizing the image size to 60x60, Augmentation, normalization, number of layers, number of nodes etc

The steps involved in our project are

Here are the general steps for traffic sign classification using a Convolutional Neural Network (CNN):

Data Collection : Collect a large dataset of traffic sign images. There are several public datasets available online, such as the German Traffic Sign Recognition Benchmark (GTSRB) .

Data Preprocessing: Preprocess the data by resizing all the images to a common size and converting them to grayscale or RGB depending on the network architecture used.

Data Augmentation: To improve the performance of the CNN, augment the dataset by applying random transformations such as rotations, translations, and flips to create new images.

Data Splitting: Split the dataset into training, validation, and testing sets. The training set is used to train the CNN, the validation set is used to tune the hyperparameters and monitor the training process, and the testing set is used to evaluate the performance of the model on unseen data.

Model Architecture: Design the CNN architecture by choosing the number of convolutional layers, pooling layers, and fully connected layers. The architecture can be chosen based on prior research or by experimentation.

Model Training: Train the CNN using the training set and the chosen architecture. The loss function used should be appropriate for multi-class classification such as cross-entropy loss. The training process can be monitored using the validation set.

Model Evaluation: Evaluate the trained CNN on the testing set by computing the accuracy, precision, recall, and F1 score. It is also recommended to visualize the confusion matrix to identify which classes are misclassified.

Model Fine-tuning: If the performance of the model is not satisfactory, fine-tune the model by adjusting the hyperparameters such as learning rate, batch size, and regularization strength.

4. Experiments and conduction

We used a CNN (Convolutional Neural Network) deep learning approach for our traffic sign recognition project. We trained our CNN model on the GTSRB (German Traffic Sign Recognition Benchmark) dataset, which contains thousands of images of traffic signs labeled with their corresponding class labels. Our CNN model had an input size of 30x30 pixels, which we built for our specific task.

To improve the performance of our model, we performed several pre-processing techniques on the images before training, including resizing the image size to 60x60, augmentation, and normalization. Resizing the image size allowed our model to capture more detailed features in the images, while augmentation increased the diversity of our training data and prevented overfitting. Normalization improved the stability and convergence of our model during training.

We also adjusted the number of layers and nodes in our network to optimize its performance. Adding more layers and nodes increased the complexity of our model, which allowed it to capture more complex features in the images, but also increased the risk of overfitting and required more computing resources to train.

Overall, our approach involved a combination of pre-processing techniques and careful selection of model architecture and input size to achieve the best performance for our traffic sign recognition task.

MODEL 1

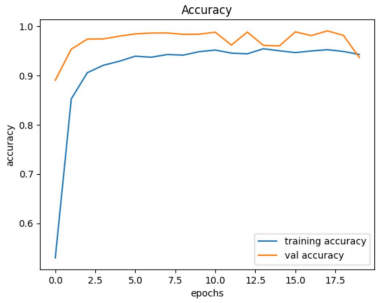
```
for i in range(classes):
    path = os.path.join(cur_path,'Train',str(i))
    images = os.listdir(path)
    for a in images:
        try:
            image = Image.open(path + '\\'+ a)
            image = image.resize((30,30))
            image = np.array(image)
            data.append(image)
            labels.append(i)
        except Exception as e:
            print(e)
```

Pre-processing

```
model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu', input_shape=X_train.shape[1:]))
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))
# We have 43 classes that's why we have defined 43 in the dense
model.add(Dense(43, activation='softmax'))
```

Building Model

```
Epoch 16/20
981/981 [==============================] - 79s 80ms/step - loss: 0.2276 - accuracy: 0.9466 - val_loss: 0.0505 - val_accurac
y: 0.9888
Epoch 17/20
981/981 [==============================] - 77s 79ms/step - loss: 0.2125 - accuracy: 0.9499 - val_loss: 0.0667 - val_accurac
y: 0.9811
Epoch 18/20
981/981 [==============================] - 79s 81ms/step - loss: 0.1973 - accuracy: 0.9525 - val_loss: 0.0361 - val_accurac
y: 0.9908
Epoch 19/20
981/981 [==============================] - 80s 82ms/step - loss: 0.2194 - accuracy: 0.9489 - val_loss: 0.0726 - val_accurac
y: 0.9816
Epoch 20/20
981/981 [==============================] - 75s 77ms/step - loss: 0.2394 - accuracy: 0.9427 - val_loss: 0.2396 - val_accurac
y: 0.9365
```

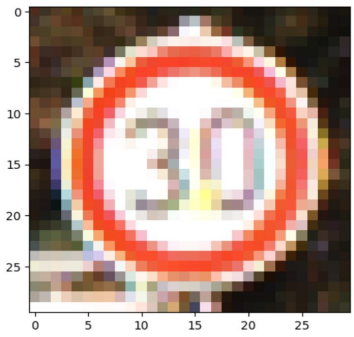


20 Epochs

Accuracy

```
plot, prediction = test_on_img(r'C:\Users\Tharun\OneDrive\Desktop\Intelligent_systems\Traffic_Sign_Recognition\Test\00024.png
a = np.argmax(prediction)
# print("Predicted traffic sign is: ", classes[a])
print("Predicted traffic sign is: ", classes.get(a))

plt.imshow(plot)
```

```
1/1 [==============================] - 0s 35ms/step
Predicted traffic sign is:  Speed limit (30km/h)

<matplotlib.image.AxesImage at 0x200af6ed0d0>
```



Result

Output

MODEL 2

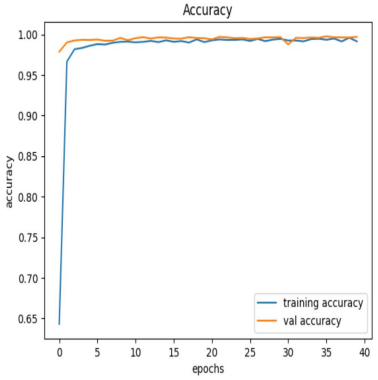
```
# cur_path = os.getcwd()
data_path = os.path.join(cur_path, "Train")

# Preprocessing parameters
img_size = 64
batch_size = 32

# Load the data and perform pre-processing
data = []
labels = []

for i in range(43):
    path = os.path.join(data_path, str(i))
    images = os.listdir(path)
    for a in images:
        try:
            image = Image.open(os.path.join(path, a))
            # Resize the images to a larger size
            image = image.resize((img_size, img_size))
            image = np.array(image)
            # Normalize the pixel values to [0, 1]
            image = image / 255.0
            data.append(image)
            labels.append(i)
        except Exception as e:
            print(e)
```

```
from tensorflow.keras.layers import LeakyReLU
model = Sequential()

model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(img_size, img_size, 3)))
model.add(Conv2D(filters=32, kernel_size=(3, 3), 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(Conv2D(filters=128, kernel_size=(3, 3), activation='relu'))
model.add(Conv2D(filters=128, 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(units=512))
model.add(LeakyReLU(alpha=0.1))
model.add(Dropout(rate=0.5))
model.add(Dense(units=43, activation='softmax'))
```

Pre-processing

Building Model

```
history = model.fit(X_train, y_train, batch_size=32, epochs=40, validation_data=(X_test, y_test))
racy: 0.9960
Epoch 35/40
981/981 [==============================] - 243s 247ms/step - loss: 0.0282 - accuracy: 0.9946 - val_loss: 0.0301 - val_accu
racy: 0.9957
Epoch 36/40
981/981 [==============================] - 294s 300ms/step - loss: 0.0332 - accuracy: 0.9933 - val_loss: 0.0377 - val_accu
racy: 0.9974
Epoch 37/40
981/981 [==============================] - 307s 313ms/step - loss: 0.0283 - accuracy: 0.9948 - val_loss: 0.0369 - val_accu
racy: 0.9963
Epoch 38/40
981/981 [==============================] - 296s 302ms/step - loss: 0.0473 - accuracy: 0.9913 - val_loss: 0.0313 - val_accu
racy: 0.9962
Epoch 39/40
981/981 [==============================] - 293s 299ms/step - loss: 0.0208 - accuracy: 0.9958 - val_loss: 0.0317 - val_accu
racy: 0.9960
Epoch 40/40
981/981 [==============================] - 274s 279ms/step - loss: 0.0466 - accuracy: 0.9913 - val_loss: 0.0325 - val_accu
racy: 0.9968
```



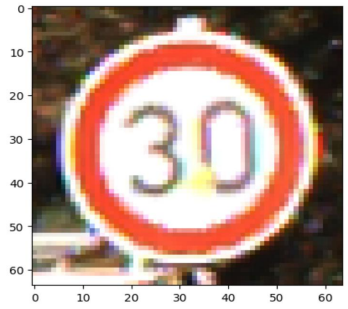
40 Epochs

Accuracy

```
plot, prediction = test_on_img(r'C:\Users\Tharun\OneDrive\Desktop\Intelligent_systems\Traffic_Sign_Recognition\Test\00024.png
a = np.argmax(prediction)
# print("Predicted traffic sign is: ", classes[a])
print("Predicted traffic sign is: ", classes.get(a))

plt.imshow(plot)
```

Results

```
1/1 [==============================] - 0s 167ms/step
Predicted traffic sign is:  Speed limit (30km/h)
<matplotlib.image.AxesImage at 0x133cf60ad00>
```



Output

5. Conclusion

In this project, we have developed a traffic sign classification system using convolutional neural networks with the GTSRB dataset. We started by exploring the dataset and preprocessing the images by resizing and normalizing them. We then built a CNN model using Keras and trained it on the dataset using various hyperparameters and optimization techniques. We evaluated the model's performance using various metrics such as accuracy, precision and fine-tuned it using data augmentation techniques to achieve better results.

From this project, we have learned several key concepts and techniques in deep learning, including:

Image preprocessing, Convolutional neural networks,Hyperparameters and optimization techniques,Model evaluations,Data augmentation.we also learnt about the YOLO but we haven't use it as it a realtime algorithm.

Overall, this project gave experience in developing and training a CNN model for traffic sign , classification. We have learned several key concepts and techniques in deep learning and gained practical skills that can be applied to other image recognition tasks.

6. References .

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, no. 7553, pp. 436-444, 2015.

[2] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. MIT Press, 2016.

[3] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," arXiv preprint arXiv:1804.02767, 2018.

[4] L. T. Tung and T. M. Nguyen, "Detecting Vietnamese Traffic Signs Using YOLOv3," in 2019 IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA), 2019, pp. 1-5. doi: 10.1109/CIVEMSA.2019.8801703.

[5] P. Sermanet and Y. LeCun, "Traffic sign recognition with multi-scale convolutional networks," in Proceedings of the International Joint Conference on Neural Networks (IJCNN), 2011, pp. 2809-2813

[6] S. Maldonado-Bascon, M. Marrón-Romera, and B. Guijarro-Berdiñas, "German Traffic Sign Detection and Recognition Using Convolutional Neural Networks," Sensors, vol. 19, no. 17, pp. 3732, 2019.

[7] D. Liang, X. Li, Y. Li, X. Yang, and Y. Li, "Traffic sign recognition based on convolutional neural network with improved data augmentation," Journal of Electronic Imaging, vol. 28, no. 3, pp. 033005, 2019.