Model Learning: Identifying

TRAFFIC SIGNS

Demiko Birden, Spencer Thompson, Morgan Tudor, Priyanka Singh

Introduction

Traffic sign recognition is an important technology for autonomous vehicles and advanced driver systems they help enhance road safety by reducing the risk of accidents.

Vehicles are aware and able to provide this data via Traffic sign recognition and other sensors on the vehicle.

Some vehicles have Autonomous Navigation so they need the ability to recognize and interpret traffic signs to make decisions and navigate. For example, the car needs to be able to understand speed limits, yield signs, and stops signs.

Dataset

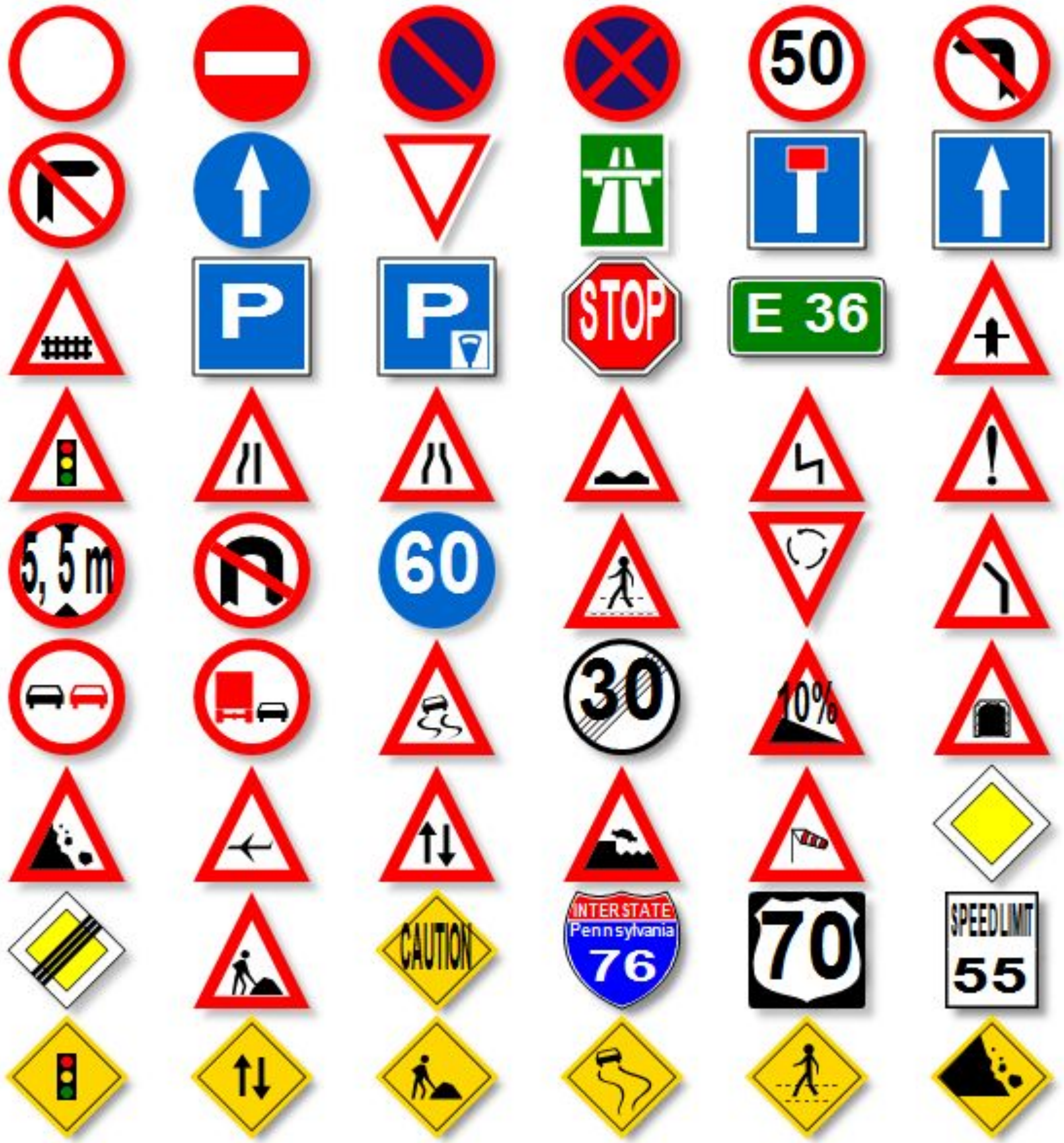
The dataset used in this project is the German Traffic Sign Recognition Benchmark (GTSRB).

GTSRB is a widely used benchmark in the field of machine learning to evaluate the performance of traffic sign recognition systems.

The dataset contains 43 different classes of traffic signs, which include speed limits, prohibitory signs, danger signs, and more.

It contains more than 50,000 images. The images in the dataset vary in terms of lighting conditions and angles. Some signs are partially obscured, while others may be photographed in varying weather conditions. This diversity helps in training robust models capable of performing well in real-world conditions.

Dataset

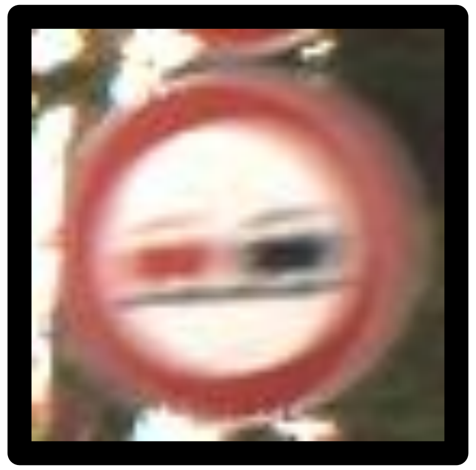


These are examples of the German road sign types that are included in the meta data set. There are 43 (0-42) individual signs included as high quality png files. These are signs that would be imperative for an autonomous vehicle to recognize to follow road safety laws and ensure the protection of drivers, other roadway users, and pedestrians.

Expectations vs Reality

While these png files are great for teaching our model to identify the signs, they're not an accurate representation of what a camera mounted on a moving vehicle might need to classify in real life.

Our model needed to have sufficient training to be able to take into account vehicle speed (motion blur), lighting, background noise in the image, color differences, angle, and size.



Building the learning model

Libraries and Dependencies

```
[1] #import libraries and dependencies
    import os
    import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    from PIL import Image
    from sklearn.model_selection import train_test_split
    import tensorflow as tf
    from tensorflow.keras.utils import to_categorical
    from tensorflow.keras.models import Sequential
    from keras.layers import Conv2D, Dense, Flatten, MaxPool2D, Dropout
    from keras.models import load_model
    from sklearn.metrics import accuracy_score
```

Utilized:

Numpy
Pandas
MatPlotLib
Pillow
SKLearn
Tensorflow/
Keras

Connect to Drive

```
[2] #Load Data from Google Drive
    from google.colab import drive
    drive.mount('/content/drive')
```

Create file paths

```
#Define file paths - some to be used later on
train_file_path = '/content/drive/My Drive/TrafficSign_Data/data/Train'
#validation_file_path = '/content/drive/My Drive/TrafficSign_Data/valid'
#test_file_path = '/content/drive/My Drive/TrafficSign_Data/test'
```

Open and store training file data

```python
#create lists to store information
data = []
labels = []
CLASSES = 43

# using for loop to access each image in the training folder
for i in range(CLASSES):
    img_path = os.path.join(train_file_path, str(i))
    for img in os.listdir(img_path):
        im = Image.open(os.path.join(train_file_path, str(i), img))
        im = im.resize((30,30))
        im = np.array(im)
        data.append(im)
        labels.append(i)

#format the data as an array
data = np.array(data)
labels = np.array(labels)
```

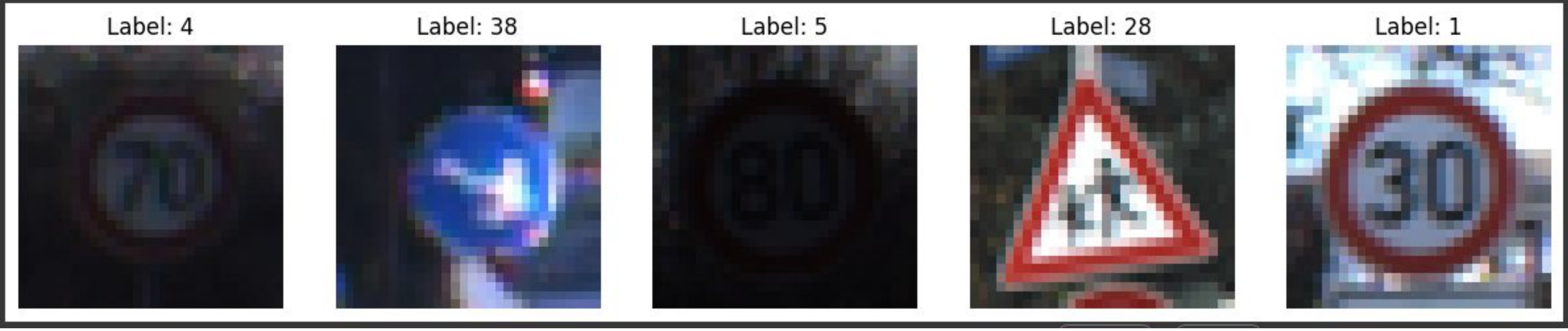
Opens the image files, uniformly sizes them, converts them to an array format, and creates lists with the data and relevant label (supervised learning)

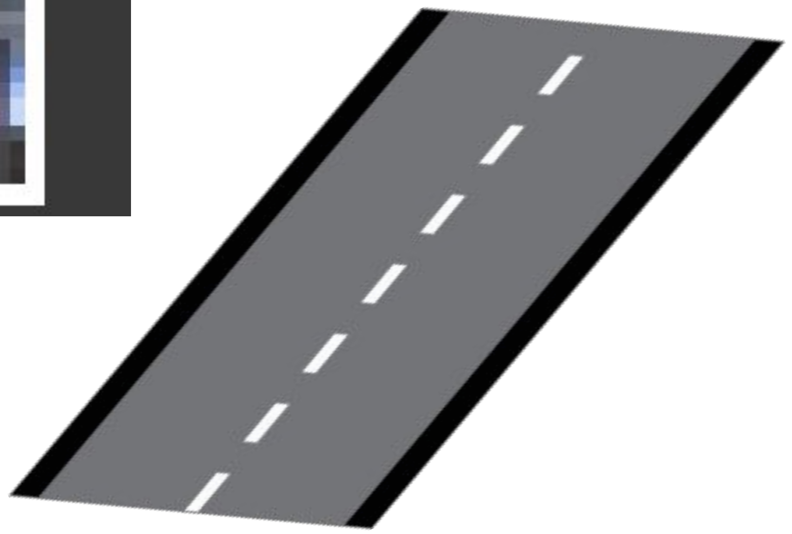
Visualizing training data

```
#displaying 5 random images to show what we are working with (credit: chat gpt)
import random

# Get 5 random indices
random_indices = random.sample(range(len(data)), 5)

# Plot the images
plt.figure(figsize=(15, 5))
for i, idx in enumerate(random_indices):
    plt.subplot(1, 5, i + 1)
    plt.imshow(data[idx])
    plt.title(f"Label: {labels[idx]}")
    plt.axis('off')
plt.show()
```

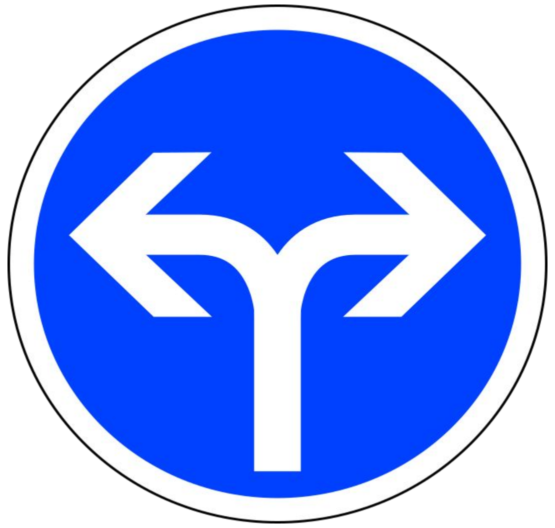




Creating our training and test sets and introducing the dummy variables (one hot)

```
#split data into different sets
x_train, x_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=42)

# convert integer label to one-hot data (binary for reading) through our complete list of classes
y_train = to_categorical(y_train, 43)
y_test = to_categorical(y_test, 43)
```



Building the model

```python
#build the model
model = Sequential()

#layer
model.add(Conv2D(filters=32, kernel_size=(5,5), activation="relu", input_shape=x_train.shape[1:]))

#layer
model.add(Conv2D(filters=32, kernel_size=(5,5), activation="relu"))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(rate=0.25))

#layer
model.add(Conv2D(filters=64, kernel_size=(3,3), activation="relu"))

#layer
model.add(Conv2D(filters=64, kernel_size=(3,3), activation="relu"))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(rate=0.25))

#flatten
model.add(Flatten())

#layer
model.add(Dense(256, activation="relu"))
model.add(Dropout(rate=0.5))

#softmax output - best for multi-class image outputs to help it converge efficiently
model.add(Dense(43, activation="softmax"))


model.summary()
```

1) Sequential model - creates a linear stack of layers where layers are added sequentially (CNN)

2) Conv2D adds our first convolutional layer with 32 filters and a relu activation.

3) A second conv2D layer to extract more features.

4) 4) MaxPool2D - downsamples the spatial dimension of input data

Building the model

5) Dropout - helps to regularize the data and prevent overfitting

6) Flatten - takes 2D output and converts it to a 1D array, prepares it for a connected layer

7) Fully connected layer with 256 units

8) Softmax layer with 43 units (our classes) which uses probability scores to assign labels to our images

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        2432

 conv2d_1 (Conv2D)           (None, 22, 22, 32)        25632

 max_pooling2d (MaxPooling2  (None, 11, 11, 32)        0
 D)

 dropout (Dropout)           (None, 11, 11, 32)        0

 conv2d_2 (Conv2D)           (None, 9, 9, 64)          18496

 conv2d_3 (Conv2D)           (None, 7, 7, 64)          36928

 max_pooling2d_1 (MaxPoolin  (None, 3, 3, 64)          0
 g2D)

 dropout_1 (Dropout)         (None, 3, 3, 64)          0

 flatten (Flatten)           (None, 576)               0

 dense (Dense)               (None, 256)               147712

 dropout_2 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 43)                11051

=================================================================
Total params: 242251 (946.29 KB)
Trainable params: 242251 (946.29 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Compiling the model

```
[8] #compile model - categorical crossentropy better to use with softmax apparently
    model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
```

Fitting the model

```
[9] #fit the model and run a batch size because the data is so big - compare it with the validation data
    #the validation data helps us to know when the data might be overfit - if it's accuracy goes down but the validation data is high, we know there is something going on
    fit_model = model.fit(x_train, y_train, epochs=25, batch_size=75, validation_data=(x_test, y_test))
```

```
Epoch 1/25
419/419 [==============================] - 68s 158ms/step - loss: 2.4927 - accuracy: 0.3624 - val_loss: 1.1208 - val_accuracy: 0.7222
Epoch 2/25
419/419 [==============================] - 66s 157ms/step - loss: 1.1794 - accuracy: 0.6532 - val_loss: 0.4405 - val_accuracy: 0.8886
Epoch 3/25
419/419 [==============================] - 66s 159ms/step - loss: 0.6398 - accuracy: 0.8093 - val_loss: 0.2380 - val_accuracy: 0.9375
Epoch 4/25
419/419 [==============================] - 66s 157ms/step - loss: 0.4530 - accuracy: 0.8670 - val_loss: 0.1467 - val_accuracy: 0.9663
Epoch 5/25
419/419 [==============================] - 70s 167ms/step - loss: 0.3413 - accuracy: 0.8993 - val_loss: 0.1400 - val_accuracy: 0.9633
Epoch 6/25
419/419 [==============================] - 67s 159ms/step - loss: 0.2886 - accuracy: 0.9145 - val_loss: 0.1087 - val_accuracy: 0.9728
Epoch 7/25
419/419 [==============================] - 66s 158ms/step - loss: 0.2548 - accuracy: 0.9243 - val_loss: 0.0630 - val_accuracy: 0.9829
Epoch 8/25
419/419 [==============================] - 66s 157ms/step - loss: 0.2093 - accuracy: 0.9379 - val_loss: 0.0679 - val_accuracy: 0.9799
Epoch 9/25
419/419 [==============================] - 66s 158ms/step - loss: 0.2122 - accuracy: 0.9382 - val_loss: 0.0656 - val_accuracy: 0.9842
Epoch 10/25
419/419 [==============================] - 66s 158ms/step - loss: 0.1890 - accuracy: 0.9438 - val_loss: 0.0900 - val_accuracy: 0.9760
Epoch 11/25
419/419 [==============================] - 66s 157ms/step - loss: 0.1778 - accuracy: 0.9480 - val_loss: 0.0549 - val_accuracy: 0.9853
Epoch 12/25
419/419 [==============================] - 66s 157ms/step - loss: 0.1811 - accuracy: 0.9486 - val_loss: 0.0453 - val_accuracy: 0.9870
Epoch 13/25
419/419 [==============================] - 66s 158ms/step - loss: 0.1728 - accuracy: 0.9535 - val_loss: 0.0487 - val_accuracy: 0.9869
Epoch 14/25
419/419 [==============================] - 66s 157ms/step - loss: 0.1675 - accuracy: 0.9529 - val_loss: 0.0814 - val_accuracy: 0.9770
Epoch 15/25
419/419 [==============================] - 65s 156ms/step - loss: 0.1501 - accuracy: 0.9565 - val_loss: 0.0450 - val_accuracy: 0.9880
Epoch 16/25
419/419 [==============================] - 66s 157ms/step - loss: 0.1532 - accuracy: 0.9560 - val_loss: 0.0537 - val_accuracy: 0.9838
```

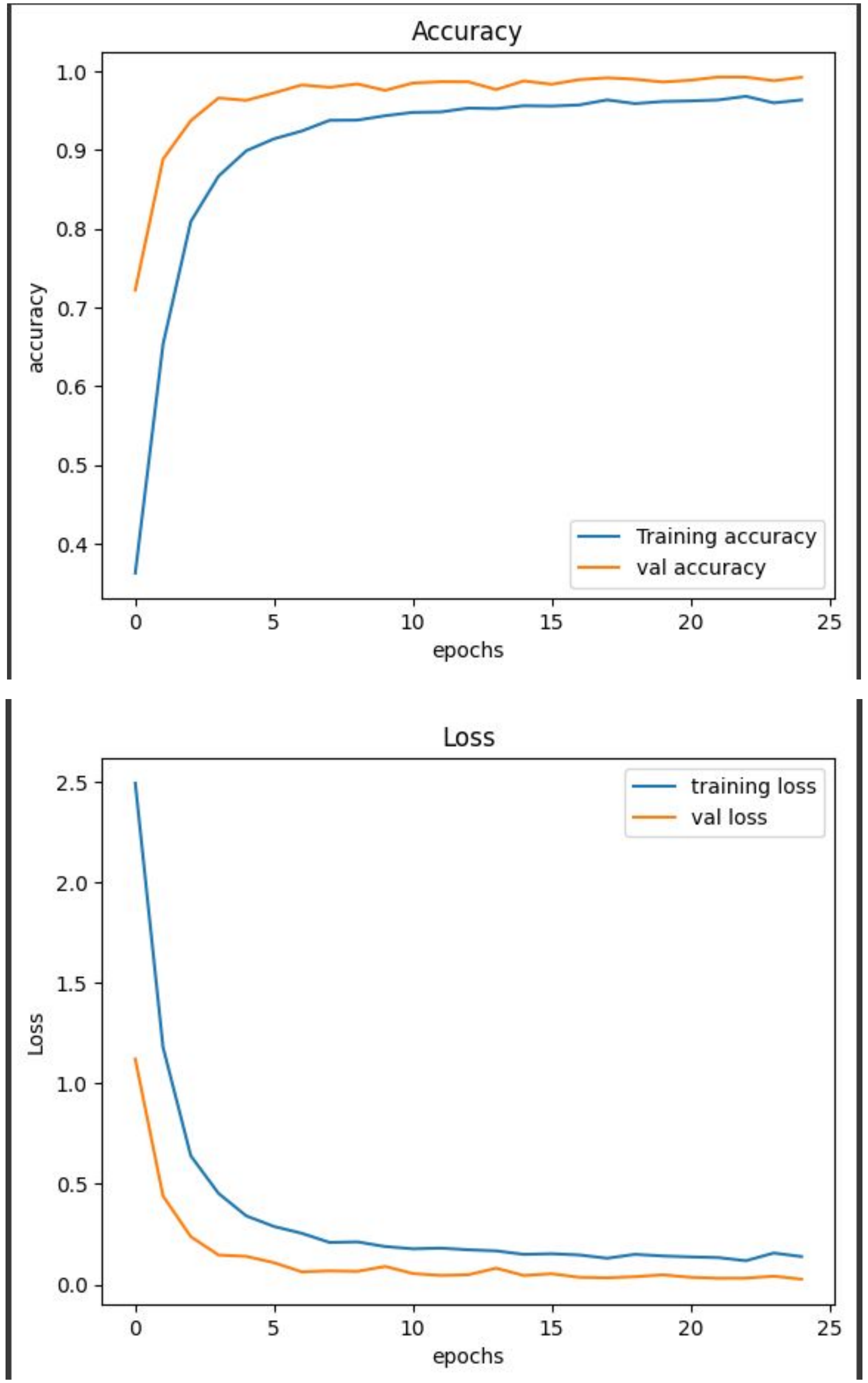
Fitting the model

```
Epoch 17/25
419/419 [==============================] - 66s 157ms/step - loss: 0.1471 - accuracy: 0.9575 - val_loss: 0.0360 - val_accuracy: 0.9897
Epoch 18/25
419/419 [==============================] - 65s 156ms/step - loss: 0.1308 - accuracy: 0.9639 - val_loss: 0.0335 - val_accuracy: 0.9920
Epoch 19/25
419/419 [==============================] - 79s 188ms/step - loss: 0.1501 - accuracy: 0.9592 - val_loss: 0.0392 - val_accuracy: 0.9902
Epoch 20/25
419/419 [==============================] - 66s 156ms/step - loss: 0.1421 - accuracy: 0.9619 - val_loss: 0.0483 - val_accuracy: 0.9866
Epoch 21/25
419/419 [==============================] - 66s 158ms/step - loss: 0.1374 - accuracy: 0.9626 - val_loss: 0.0361 - val_accuracy: 0.9889
Epoch 22/25
419/419 [==============================] - 67s 159ms/step - loss: 0.1341 - accuracy: 0.9638 - val_loss: 0.0311 - val_accuracy: 0.9930
Epoch 23/25
419/419 [==============================] - 66s 157ms/step - loss: 0.1184 - accuracy: 0.9685 - val_loss: 0.0319 - val_accuracy: 0.9929
Epoch 24/25
419/419 [==============================] - 66s 158ms/step - loss: 0.1567 - accuracy: 0.9602 - val_loss: 0.0417 - val_accuracy: 0.9883
Epoch 25/25
419/419 [==============================] - 66s 157ms/step - loss: 0.1390 - accuracy: 0.9638 - val_loss: 0.0268 - val_accuracy: 0.9926
```

Plotting the results

```
#AccuracyPlot
plt.figure(0)
plt.plot(fit_model.history['accuracy'], label="Training accuracy")
plt.plot(fit_model.history['val_accuracy'], label="val accuracy")
plt.title("Accuracy")
plt.xlabel("epochs")
plt.ylabel("accuracy")
plt.legend()

#LossPlot
plt.figure(1)
plt.plot(fit_model.history['loss'], label="training loss")
plt.plot(fit_model.history['val_loss'], label="val loss")
plt.title("Loss")
plt.xlabel("epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



Testing and evaluating the model

```
[35] #test and evaluate the model
     test_file_path = '/content/drive/My Drive/TrafficSign_Data/data/'
     Y_test = pd.read_csv(test_file_path + 'Test.csv')
     test_labels = Y_test["ClassId"].values
     test_images = Y_test["Path"].values

     from keras.preprocessing.image import load_img

     output = list()
     for img in test_images:
         image = load_img(os.path.join(test_file_path, img), target_size=(30, 30))
         output.append(np.array(image))

     X_test=np.array(output)
     pred = model.predict(X_test)
     pred=np.argmax(pred, axis=1)

     #Accuracy with the test data
     print('Test Data accuracy: ',accuracy_score(test_labels, pred)*100)

     395/395 [==============================] - 8s 20ms/step
     Test Data accuracy:  94.60807600950119
```

Visualizing the results

```
[ ]  #visualizing the predictions
     plt.figure(figsize = (16, 16))
     pred = model.predict(x_test)

     start_index = 0
     for i in range(25):
         plt.subplot(5, 5, i + 1)
         plt.grid(False)
         plt.xticks([])
         plt.yticks([])
         prediction = pred[start_index + i]
         actual = test_labels[start_index + i]
         col = 'g'
         if prediction != actual:
             col = 'r'
         plt.xlabel('Actual={} || Pred={}'.format(actual, prediction), color = col)
         plt.imshow(X_test[start_index + i])
     plt.show()
```

Benefits

Visual Aid

Can be an aid to those on the road suffering from visual impairment.

Concentration

Enables drivers to be more focused in complicated situations or areas they aren't familiar driving.

Autonomous Driving

Allows "self-driving" cars recognize the scenarios they are approaching.

Human Error

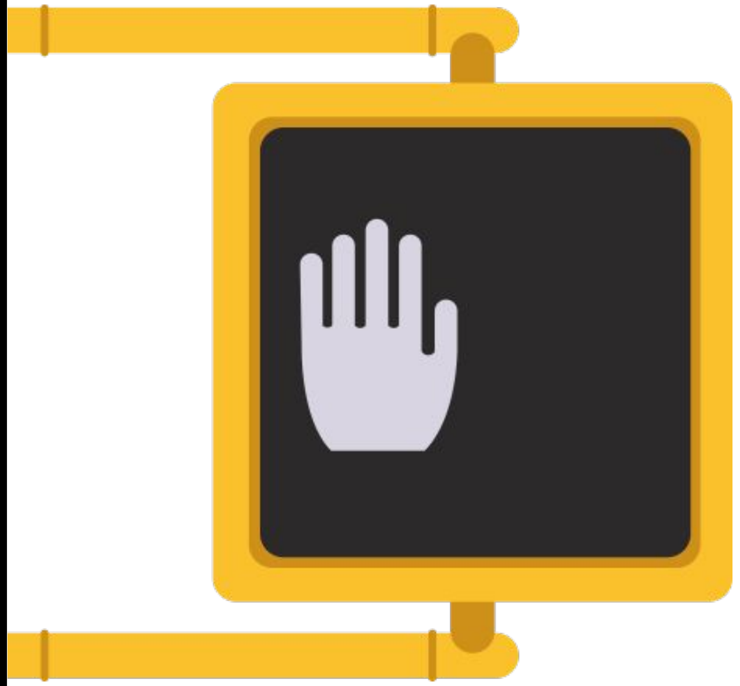
Helps with reducing the chances of human error.

Data effect on Reality

Given that the data has 94% accuracy, it is important to understand how this would actually translate to the real world. For example: 6 out of every 100 cars driving autonomously would not be able to properly follow all road signs legally and safely.

With this knowledge, it would be important for car companies working with autonomous vehicles to take the proper safety steps and trainings to ensure safe driving on the road.

Conclusion



Traffic sign recognition is crucial for advanced driving technologies and autonomous vehicles to ensure the safety of those on the road. While the data has given us a result of 94% accuracy, this cannot ensure guaranteed reliance and safety of this technology.