Traffic Sign Recognition

Implementing the CNN Architecture to Accurately Identify Road Signs

Hephaestus Applied Artificial Intelligence Association

Authors:

Member	Role
Emma Mora	Co-Head
Merve Şentürk	Co-Head
Utku Bahcivanoglu	Member
Tommaso Ferracina	Member
George Morris	Member
Alisia Picciano	Member
Paul Schappert	Member
Ianina Tarca	Member
Matilde Zambiasi	Member

Milan, February 24, 2024

Contents

1 | Introduction

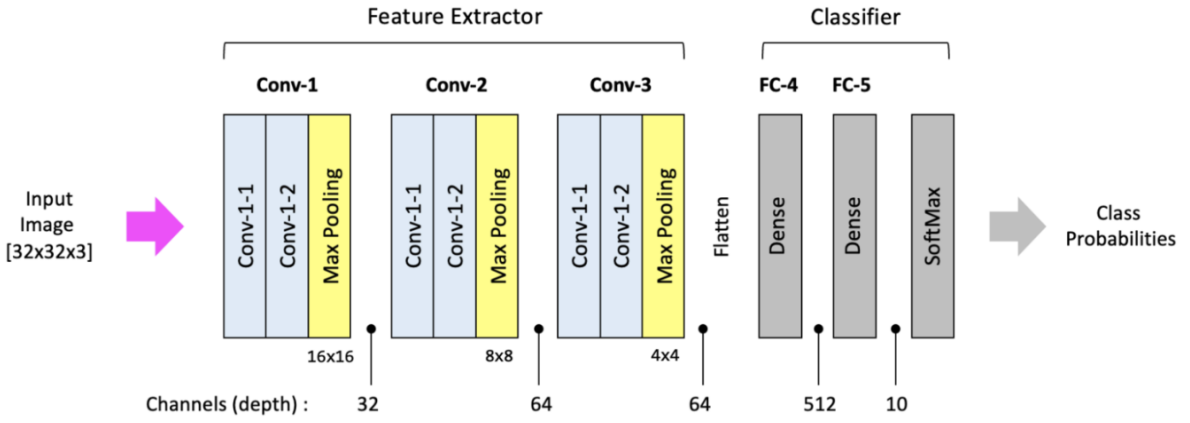
In these past few years, all car manufacturing companies have been investing heavily on research regarding self driving vehicles. This field of technology involves multiple areas of science, from mechanical engineering to data analysis, with the ultimate goal of obtaining reliable and safe automated driving. Artificial Intelligence is one of the tools that can be used to avoid the necessity for human intervention. In this specific case, we will be focusing on the development of a model that will be capable of recognising traffic signs starting from pictures. This is obviously one of the most important features in any self-driving system which hopes to achieve safety on the road. The goal of attaining self-driving vehicles is a prominent and well-known application of AI which is of great interest to multinationals such as Tesla, Google and many others. Road sign recognition is a key component of any self-driving system which hopes to achieve safety on the road. In this project we aimed to develop a model which uses deep learning techniques to do just this and classify specific road signs from images.

1.1 | Overview of the project

We began with a set of 50,000 images composed of 43 different types of road sign which we will split in two to first train and then test our model. As this project is focused solely on classifying the road signs our program will not check if an image is indeed that of a road sign or not - an added level of complexity which a self-driving system will need.
As ours is an image classification problem we built a CNN (Convolutional Neural Network) model, trained and validated it on our data and finally tested its accuracy.

1.2 | Basic knowledge about CNNs

Convolutional Neural Networks (CNNs) represent a class of deep learning models specifically designed for processing and analysing visual data, making them particularly effective in image recognition tasks. The architecture of CNNs is inspired by the visual processing mechanisms in the human brain. At their core, CNNs employ convolutional layers to extract hierarchical features from input images. These layers consist of filters or kernels that slide across the input, capturing local patterns and learning to recognize low to high-level features. Pooling layers follow the convolutional layers, reducing spatial dimensions while retaining essential information. This reduction in dimensionality helps in managing computational complexity and can lead to a more efficient representation of the learned features. This hierarchical feature extraction allows CNNs to automatically learn intricate patterns and representations from raw pixel data. The subsequent fully connected layers then use these features for classification or other tasks. Next, we implement a dropout layer, which randomly eliminates information to avoid overfitting the model. The strength of CNNs lies in their ability to recognise spatial hierarchies and relationships within images, making them well-suited for tasks like image classification, object detection, and segmentation. The convolutional and pooling operations enable CNNs to efficiently capture relevant features, preserving spatial dependencies crucial for understanding complex visual information. In summary, CNNs leverage convolutional and pooling layers to automatically learn hierarchical features from images, making them powerful tools for various image-related tasks due to their ability to capture spatial relationships and patterns.



1.3 | Basic knowledge about Tensorflow and Keras

Python is a preferred language for Convolutional Neural Networks (CNNs) due to its simplicity, readability, and extensive libraries like TensorFlow and Keras. TensorFlow is an open-source machine learning library developed by the Google Brain team. Launched in 2015, it has become one of the most popular frameworks for building and training machine learning models. Its key feature is its computational graph, where mathematical operations are represented as nodes and executed efficiently on various devices, including CPUs and GPUs. TensorFlow supports a wide range of applications, from language processing to speech and image recognition, which is the goal we strive to achieve. With its high-level APIs like Keras, developers can easily build and deploy neural networks. Keras is a high-level neural network API written in Python that serves as an interface for the TensorFlow library. Developed with a focus on user-friendliness and modularity, Keras enables rapid prototyping and experimentation in building neural networks. Introduced as a standalone project, Keras became an integral part of TensorFlow in 2017, making it the default high-level API for the framework. It supports various neural network architectures, including convolutional and recurrent networks, and facilitates quick implementation of image classification.

2 | Model and dataset

2.1 | Data set

As previously mentioned our dataset consists of 50,000 images made up of 43 different types of road signs in Germany, it can be found on Kaggle and is part of the public domain. First, we split the dataset in two to first train and then test the model at a ratio of about 4:1. Second, as CNN is in the supervised category of neural networks we must have a labelled dataset. In this case each image was ‘“labelled” with the type of road sign it is, to do this we used an OS module and the PIL library to store every image with its corresponding label into lists. Finally, we converted these lists into a NumPy array and our data is ready to be fed into the model. The shape of this final formatted data was (39209, 30, 30, 3), where 39209 is the number of images (training data), each image has 30*30 pixels and 3 represents the RGB value (color).

```
cur_path = os.getcwd()
cur_path = os.path.join(cur_path, "main_data")
data =[]
labels=[]
classes = 43
#this is because in meta file we see that we are working with 43 unique labels-Utku

#going to change this part to normal uploading

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:
        print("Error loading image")
data = np.array(data)
labels = np.array(labels)

X_1, X_2, y_1,y_2 = train_test_split(data, labels, test_size=0.2, random_state=42)


y_1= keras.utils.to_categorical(y_1, 43)
y_2=keras.utils.to_categorical(y_2, 43)
```

2.2 | The model

The model initiates with a sequential layout, allowing for the linear stacking of layers. The architecture includes multiple 2D convolutional layers, each followed by ReLU activation functions to introduce non-linearity, enabling the model to learn complex patterns in the data. These convolutional layers, with varying numbers of filters (32, 64, and 128), are crucial for feature extraction directly from the input images. To mitigate the risk of overfitting—a common issue in deep learning models—dropout layers with rates of 0.25 and 0.5 are interspersed between the convolutional layers, randomly dropping a portion of the input units, thus forcing the network to learn more robust features. Additionally, max pooling layers are employed to reduce the spatial dimensions of the output from preceding layers, effectively lowering the number of parameters and computation required. After flattening the output to convert the 2D feature maps into a 1D feature vector, the model integrates fully connected layers, including a dense layer with 512 units (ReLU activated) and a final softmax layer with 43 units for multi-class classification, indicating the model’s capability to distinguish among 43 different categories. The model is compiled using the Adam optimizer and categorical crossentropy as the loss function, focusing on accuracy as the primary metric. Training is conducted over 15 epochs with a batch size of 32, employing separate training and validation datasets to optimize the weights and assess performance, respectively. Upon completion of training, the model is saved to a file, allowing for future reuse without the necessity for retraining. This CNN model exemplifies a robust approach to tackling image classification tasks, emphasizing the importance of layer architecture, regularization techniques, and efficient optimization

strategies in developing high-performing deep learning models. Observing two different accuracy levels from the execution of ostensibly identical code can be attributed to several factors inherent to the training of deep learning models. Firstly, the initialization of model weights, which are usually set to small random values, can lead to variability in learning outcomes even with the same network architecture and training data. Secondly, the stochastic nature of the optimization algorithm itself, particularly when using methods like stochastic gradient descent (SGD) or variants thereof, introduces variability. These algorithms rely on random subsets (mini-batches) of the data for each update, which can affect the trajectory of convergence. Lastly, slight differences in the computational environment, such as differences in floating-point precision or parallel processing behavior, can also lead to discrepancies in results.

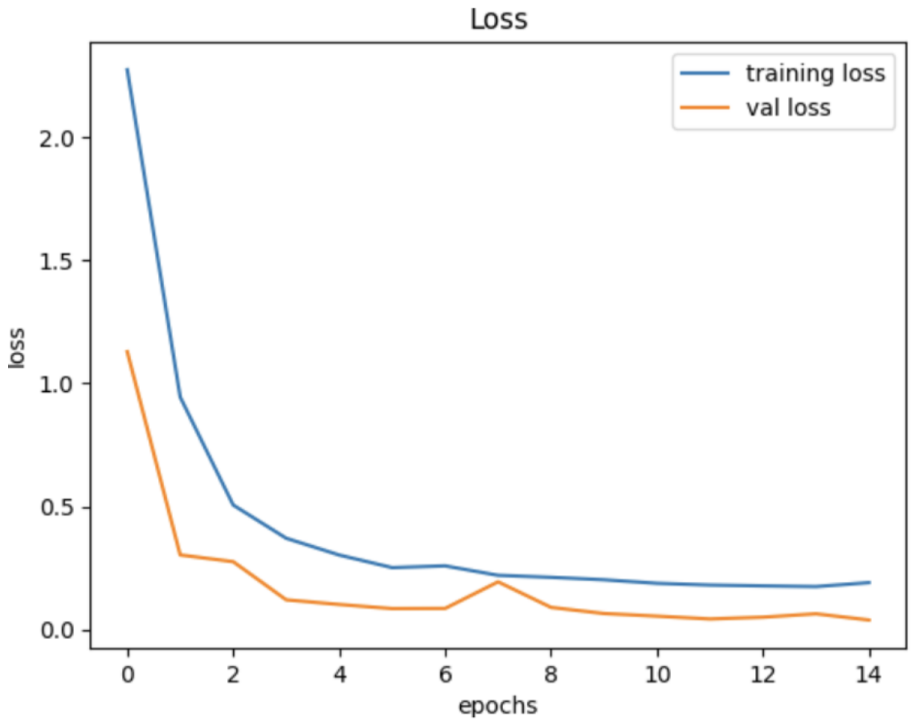
2.3 | Building the model

```
model = keras.models.Sequential()

model.add(keras.layers.Conv2D(32, kernel_size =(3,3), activation="relu", input_shape= data.shape[1:]))
model.add(keras.layers.Conv2D(32, kernel_size=(3,3), activation="relu"))
model.add(keras.layers.MaxPool2D(pool_size=(2, 2)))
model.add(keras.layers.Dropout(rate=0.25))
model.add(keras.layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(keras.layers.Conv2D(filters=128, kernel_size=(3, 3), activation='relu'))
model.add(keras.layers.MaxPool2D(pool_size=(2, 2)))
model.add(keras.layers.Dropout(rate=0.25))
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(512, activation='relu'))
model.add(keras.layers.Dropout(rate=0.5))
model.add(keras.layers.Dense(43, activation='softmax'))

model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])

mdl = model.fit(X_1, y_1, batch_size=32, epochs=15, validation_data=[X_2, y_2])
model.save("my_model.h5")
```



```
Epoch 1/15
981/981 [==============================] - 276s 181ms/step - loss: 2.2735 - accuracy: 0.3915 - val_loss: 1.1288 - val_accuracy: 0.7042
Epoch 2/15
981/981 [==============================] - 178s 182ms/step - loss: 0.9442 - accuracy: 0.7230 - val_loss: 0.3030 - val_accuracy: 0.9186
Epoch 3/15
981/981 [==============================] - 181s 184ms/step - loss: 0.5058 - accuracy: 0.8541 - val_loss: 0.2757 - val_accuracy: 0.9245
Epoch 4/15
981/981 [==============================] - 178s 182ms/step - loss: 0.3708 - accuracy: 0.8967 - val_loss: 0.1208 - val_accuracy: 0.9699
Epoch 5/15
981/981 [==============================] - 178s 181ms/step - loss: 0.3027 - accuracy: 0.9157 - val_loss: 0.1019 - val_accuracy: 0.9736
Epoch 6/15
981/981 [==============================] - 180s 184ms/step - loss: 0.2515 - accuracy: 0.9307 - val_loss: 0.0851 - val_accuracy: 0.9759
Epoch 7/15
981/981 [==============================] - 179s 182ms/step - loss: 0.2586 - accuracy: 0.9322 - val_loss: 0.0856 - val_accuracy: 0.9753
Epoch 8/15
981/981 [==============================] - 179s 183ms/step - loss: 0.2202 - accuracy: 0.9433 - val_loss: 0.1938 - val_accuracy: 0.9447
Epoch 9/15
981/981 [==============================] - 190s 193ms/step - loss: 0.2121 - accuracy: 0.9450 - val_loss: 0.0899 - val_accuracy: 0.9755
Epoch 10/15
981/981 [==============================] - 179s 183ms/step - loss: 0.2022 - accuracy: 0.9477 - val_loss: 0.0651 - val_accuracy: 0.9824
Epoch 11/15
981/981 [==============================] - 181s 185ms/step - loss: 0.1878 - accuracy: 0.9526 - val_loss: 0.0544 - val_accuracy: 0.9851
Epoch 12/15
981/981 [==============================] - 186s 189ms/step - loss: 0.1807 - accuracy: 0.9539 - val_loss: 0.0430 - val_accuracy: 0.9888
Epoch 13/15
981/981 [==============================] - 185s 189ms/step - loss: 0.1773 - accuracy: 0.9558 - val_loss: 0.0502 - val_accuracy: 0.9860
Epoch 14/15
981/981 [==============================] - 180s 184ms/step - loss: 0.1745 - accuracy: 0.9569 - val_loss: 0.0639 - val_accuracy: 0.9829
Epoch 15/15
981/981 [==============================] - 179s 182ms/step - loss: 0.1905 - accuracy: 0.9524 - val_loss: 0.0387 - val_accuracy: 0.9890
```

3 | Conclusion

Having completed the training of our model there were a number of conclusions we could draw. For one our model does at least work to some degree and in fact it had an accuracy of 0.9594 on the test data - truly not a bad result for such a rudimental approach as ours as it conveys that our model is able to correctly identify a road sign correctly approximately 95% of the time. However, with regards to the initial inspiration of the project, namely the development of this deep learning model for a self-driving automated system, unsurprisingly, this result is nowhere near satisfactory. Indeed if a human driver were to incorrectly infer the meaning of a road sign 5% of the time, an accident would occur very quickly. Nevertheless, the construction of such a model was certainly a very informative and educational process. Moreover, it is interesting to reflect on what needs to be done to make our model more accurate on unseen data. For example, a larger dataset with augmented data would be required to increase the diversity of the training set. This data could also be normalised to stabilise and accelerate the training process. One could also increase the depth and width of the CNN (number of layers) to make the model better at capturing complex patterns and features of the inputs. All steps which require vast resources and enormous amounts of computational power the likes of which the companies and research outlets leading this development possess. In a nutshell, our project successfully demonstrates the application of CNNs in the complex task of road sign classification, illustrating both the power and challenges of deep learning methodologies. The observed variability in model performance serves as a reminder of the importance of rigorous evaluation and validation practices in the development of machine learning models. Moving forward, strategies such as averaging results over multiple runs, using ensemble methods, or further tuning the model and training process could be explored to enhance reliability and accuracy, thereby improving the model's applicability in real-world scenarios, such as autonomous driving systems where accurate and reliable road sign recognition is crucial.

4 | References

1. https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign
2. https://www.analyticsvidhya.com/blog/2021/12/traffic-signs-recognition-using-cnn-and-keras-in-python/
3. https://www.analyticsvidhya.com/blog/2021/05/a-comprehensive-tutorial-on-deep-learning-part-1/
4. https://www.analyticsvidhya.com/blog/2016/01/complete-tutorial-learn-data-science-python-scratch-2/
5. https://www.analyticsvidhya.com/blog/2018/12/guide-convolutional-neural-network-cnn/
6. https://www.youtube.com/watch?v=SWaYRyi0TTs