Traffic Signs Classification - CNN

Rugerio Karen, ITC A01733228, Tecnológico de Monterrey Campus Puebla

Abstract: This document presents the implementation and explanation of a classification Convolutional Neural Network model made with the Tensorflow Keras library using a dataset of 42 different road traffic signs.

Introduction

Companies focus on making human lives easier. Who is not annoyed by driving from one place to another for many hours? Here is where, in order to make transportation less stressful, several companies try to create self-driving cars with no need for human interaction.

Of course, there are several aspects to consider before these cars can go to market, one of them is the Traffic Sign Recognition the cars need to interpret to make driving safer.

Artificial Intelligence's goal is to make machines act like humans. For example, when a person learns to drive, it also learns the meaning of each sign and applies it when driving. The goal for this project is to help autonomous vehicles to interpret these signs too, so they can make decisions based on the sign recognition.

The algorithm is capable of classifying an image into categories using a convolutional neural network(CNN).

Dataset

The dataset selected is the GTSRB - German Traffic Sign Recognition Benchmark [https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign]

This data set has 3 main folders:

1. **Meta** - Where all the 43 traffic signs are represented in vector images.
2. **Test** - Random images obtained from the training dataset are allocated in this folder
3. **Train** - This folder at the same time contains 43 folders each with several images of the respective enumerated sign. For example, the folder named ‘0’ has ‘speed limit of 20km’ images signs only.

By printing the data contained into the Meta set, we can have an idea of the traffic signs that conform the train and test folders. This is the display:

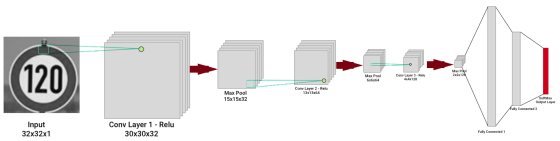
Also this folder includes 3 .csv files that include the image references.

Data Processing

Before uploading the information into the CNN it is important to process it, so we can make an easier process. In this case I decided to resize the original images into a 25x25 pixels format and applied a RGB filter. Finally I shuffled the image index to avoid overfitting. At the end, the images look something like this:

Convolutional Neural Network

Convolutional neural networks are good for image classification and pattern recognition. In the following image we can see how a CNN works.



Input - As the first feature we have the input image. It is the result of the data processing content, which means, is already resized and basically different from the image that was taken.

Convolutional net - Built by convolutional layers, each layer has an input matrix, its kernels and an output feature map.

Max pool - Performs an abstract representation that reduces the image size and sends only the important data to next layers in CNN.

Flatten - This function converts the pooled feature into a single column that is passed to the fully connected layer.

Dense - A dense layer must be performed, this section adds the fully connected layer to the neural network.

Model Applied

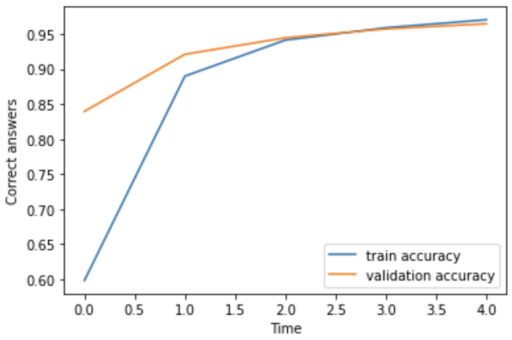
The model selected to make predictions over the dataset previously mentioned, was a sequential convolutional neural network with the following configurations:

- **Input shape** of 25x25 pixel RGB images.
- **2D convolutional** layer with a 3x3 kernel and 32 filters and Softmax activation function.
- A **max pooling layer**, with a 2x2 window, and default strides.
- A **flatten layer**, which unrolls the input into a one-dimensional tensor.
- A **dense hidden output layer** with 43 units and the sigmoid activation function.

Then we have to train the model. To do so, we fit the model using the Training folder. I decided to add another split of data into training and validation. The validation set is like a pre-test made during the training process. In this case I set the configurations to be the 70% of the training dataset used for training and the other 30% used as a validation set.

To compile the model we set the loss as a categorical_crossentropy, set Adam as our optimizer, and we will evaluate the model using the accuracy metric.
Finally we can fit the model with a batch size of 64 during 5 epochs.

With this configurations we obtain a training accuracy of 0.9704 and a validation accuracy of 0.9646

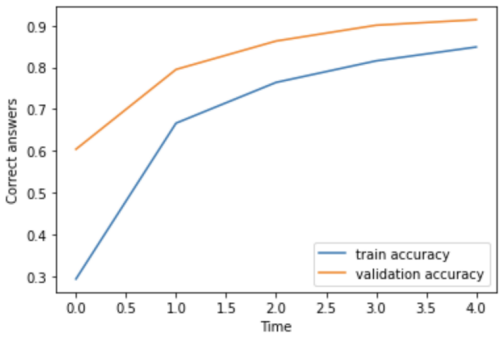


Even when the result appears to be so good, when we get a graphical visualization we can distinguish a clear overfitting, and we can also see how the validation accuracy goes down over time.

Improvement of the model

The model is already so good, but there is a clear overfitting so I wonder if I can make an improvement by adding a regularization technique called dropout. What this function does is randomly setting a fraction rate of input units to 0 at each update during training time, which helps prevent overfitting.

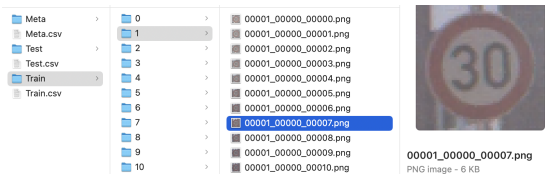
I added a bigger kernel size (7x7) , a dropout regularization technique of 0.3 so each neuron has 30% probability of being off and another dense layer of 300 neurons with a ReLu activation function.



The final accuracy of the model is 0.8488 and the validation accuracy is 0.9141 but now it is not overfitting, so I conclude that this model is better for the following predictions.

Labeling

I already have folders with numbers in the dataset from 0 to 43. Each folder contains related images. So I defined the classes for every labeled image to use them in the prediction process.




```
# Define the labels
classes = { 0:'20km/h Speed limit',
            1:'30km/h Speed limit',
            2:'50km/h Speed limit',
```

In this case for example we can see that in the ‘1’ folder contained in the ‘Train’ dataset there are only numbers ‘30’ so the label we assign to the number 1 is going to be ‘30km/h speed limit’ and we do this with all folders contained inside the Train folder.

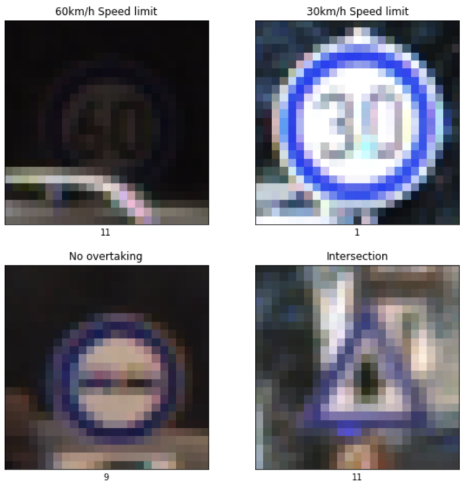
Predictions

The next step is to test the model with the test dataset. The accuracy obtained was around 83%

```
#Calculate prediction accuracy with model2 test
predict_x=modl2.predict(X_test)
classes_x=np.argmax(predict_x,axis=1)
print('Test accuracy: ',accuracy_score(labels, classes_x)*100)
```

```
395/395 [==============================] - 1s 2ms/step
Test accuracy:  83.40459224069676
```

Then I did some plots, so I can see the images the model is testing, the result predictions and the actual labels.



Doing an overview, the results seem to be accurate enough to approve the model.

My Dataset

I decided to do my own test dataset with traffic signs I found in my location, so I ended with a 10 image dataset and I ran predictions with them.

My tes had 2 disadvantages, the first one is that in Mexico most signals have yellow backgrounds, and the second one is that all is spanish labeled, for example, the 'stop' sign in here is an 'alto' sign.

In this scenario, the model was capable of predicting only one of the 10 signs.

Conclusion

When doing machine learning, it is important to have a good understanding of how to build effective models with high accuracy. But it is also important to consider the implementation of the model. As we saw if we use this model in a country that has traffic signs exactly like the test signs, this can be considered a safe model to transport things from one place to another or even people.

But what if we try to use this same model in another country like Mexico? the results for a self-driving car might be catastrophic. It is extremely important to understand how well the model performs, the model capacity and if the model is overfitting or underfitting. To improve the results of this model a hyperparameters configuration change can be considered, and to improve it in other countries, it would be useful to have that country's signs data.

This is one of the reasons why artificial intelligence in self-driving cars has so many areas of improvement I would like to continue exploring and contributing to.

References

Repository: *github.com/KarenRugerioA/TrafficSignClassification*

Dataset: *GTSRB - German Traffic Sign Recognition Benchmark* *https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign*

Video*: https://youtu.be/IfSRu6-e7fg*