Traffic sign recognition system

Moldovan Horia-Andrei

May 2020

1 Introduction

One of the many projects which must be dealt with in order to close the gap between the present and the sci-fi future we've all been dreaming for, a future in which technology is the central point of every living thing and the motor which puts all the world in motion, is the theme of autonomous driving. Autonomous driving is a project which might be able to improve our day to day lives in a drastic manner. Approximately 1.35 million people die each year as a result of traffic car accidents and many others suffer serious damage. Just think of a society in which all this chaotic traffic is replaced with a well thought car system in which all the previous numbers are drastically reduced near to zero. This might be quite hard to achieve but many results have already proven that the human brain is no match for the nowadays power of computation of a computer, so if we combine the correct logic with the correct hardware the idea of autonomous driving might become a reliable option. A lot of work has already been done and results are already starting to show up.

A small part but a very important one, in this autonomous driving project, is the one of traffic sign recognition. In order for this project to work we have to feed some data to the sensors which are working together for achieving a flawless driving system. This data must first to be captured with the help of several video cameras placed all around the car because a 360 degree view is necessary. In this project we are dealing with the problem of understanding and processing the input data images. The part which is of big interest to us is the one which allows the car to recognize and successfully classify any given traffic sign. Most of the traffic rules are impossible to be followed if we are not able to do this simple thing. Moreover, a false interpretation of the traffic signs may lead to very unpleasant outcomes. This being said, we have to make sure that our autonomous driving car is capable of first capturing images of all the near traffic signs and then interpreting this images, even if the image is damaged by some natural conditions.

The goal of this project is to identify in real time any traffic sign image which is shown to a camera. In order to achieve this we must be able to first detect the traffic sign in the provided image and then to successfully classify it, as one of the existing traffic sign classes.

2 Related work

The subject of traffic sign recognition was already very well researched and therefore a lot of scientific papers were written with the scope of breaking into pieces the working flow of achieving a program of doing such thing. Overall speaking, the problem of traffic sign recognition and classification falls in two different fields of the computer science area: image processing and artificial intelligence. The first field contains all the operations which are done on the input image in order to modify it in such a way the other field can make use of it. This modifications imply scaling, rotating, converting, and many others. After the image is processed in the exact way we want it to be, the AI part begins. Here the "magic" happens and by magic I mean the training of the neural network with the processed images given as input.

Before continuing we have to make sure the concept of neural network is as clear as possible. So what is a neural network? By definition, a neural network is a well thought circuit make of artificial networks. To picture it better we may think of this network as having some layers, the first one being the input layer, the last one the output layer and the one between them being the hidden layers. The main role of the network is to take the data from the input layer and with the help of the logic found in the hidden layers to predict the desired output. Each neuron holds some data and between every neuron of the n'th layer there lies a connection with every neuron of the n-1'th layer. This connections are called weights and they are used in order to determine the importance of the neurons from the n-1'th layer when building the next one.

Now let's see how should we make use of this neural network in order to achieve our goal. Firstly let's examine what each layer of the neuronal network means to us. The input layer is a bunch of neurons and each of them holds the data related to one pixel of the image. So for example if out image is of 10x10 format, then we will have 100 pixels so the neural network will have 100 inputs. Moving over, the output layer will have as many neurons as traffic sign classes we have provided, you can thing of this neurons as having values between 0 and 1. The purpose of the hidden layers is to look at the pixels of the given image through the input layer and from all the neurons of the output layer to set the biggest value to the neuron corresponding to the class in which the traffic sign provided in the input image lies. So let us take a look at the "magic" the network uses in order to achieve the described behaviour. Let's assume that the provided image represents a stop sign and our network has only one hidden layer. The weights between neurons are firstly initialized with random numbers, the bigger the number the more importance the pixel in the input layer has in the building of the next one. So for example, if a pixel has a red colour then the weight starting from this neuron should have a big weight because stop signs are mostly red, but as said, the neuron does not know this because the weights are initialized with random numbers. Following this starting weights the network sets to the output layer in each neuron a number between 0 and 1, meaning that the neuron with the biggest value should correspond to the desired output. But how do we know if this output is the one we were looking for? Well the answer

is quiet simple, we specify to the network what the output should look like and this output is compared with the one we've got. This comparison is done using a so called loss function. So to put is simpler, let's say that our output layer has 2 neurons, first one representing a stop sign and 2'nd one representing a speed sign. After going through the hidden layer we end up with the output neuron representing the stop sign to 0.3 and the other one to 0.6 for example, but we know that is should look like the first neuron having value 1 and the second one having value 0, because our input is indeed an image representing a stop sign. The loss function will subtract the value of the desired output neurons from the value of the actual output neurons and square it, so it will look like $(0.3\text{ -}1)^2 and (0.6-0)^2$.

We do the arithmetic mean of all the losses and so we get the final loss. The lower the loss value, the more accurate our prediction was, that's way the network desires to make the loss as low as possible using a method called back-propagation. Overall speaking, this method knows that the previous weights have lead to a wrong value of the output neuron and it tries to change this weights in order to achieve a better result. This approach is done starting from the output layer and going up to the first one, this being the reason for its name.

All this above described behaviour is known as training the network. I have exemplified its behaviour on only one image(representing a stop sign), but in order to have a well trained network it is recommended to use as many images as possible. The more images it has seen, the more knowledge it gets. After the training of the network is done, all we have to do is feed images to its input and observe if the output is the one we were looking for.

There are a lot of different types of neural networks and like most of the scientific papers suggest, the best option for implementing our traffic sign problem the most suitable neural network to work with would be the convolutional neural network, also called CNN. This CNN's are said to be one of the best option when dealing with the issue of processing images. The main difference between them and other neural networks is the fact that on the image given as input to the network, a convolution operation is performed. It is said that this operation, to convolve through the image, is being able to recognize some patterns which may be useful to identifying the correct answer and to train the network. The convolution operation works in the following way. We specify the size of an additional filter which will be iterating through the whole input image to give a new image. As example think of a 10x10 input image, each cell having the value of the pixel at that point, and a 2x2(matrix) filter with random values. The first pixel of the output matrix will be computed by applying the 2x2 filter on the first 2x2 pixels of the image and doing the dot product between them. Then next pixel of the output will be given by the dot product between the filter and the next matrix consisting of sliding the filter from the first 2x2 pixels of the image with one position to the right, so cells 0,1 , 0,2 , 1,1 and 1,2. This way a new image is created. Each convolutional hidden layer can have multiple layers or even none. Besides the convolution there is also the pooling operation. This one is used in order to centralize the data and to reduce the image size. This operation works with a filter as well and a stride. The filter goes again

through the whole image but this time each pixel of the output image consists of the mean value of the pixels within the filter. The stride specifies how the filter moves through the input image. So if we have a 2x2 filter and a stride of 2 then in comparison to the convolution, the second iteration of this operation will contain the cells 0,2 0,3 1,2 and 1,3, so it won't go over the same pixel twice. This will result in an output image of a smaller dimension. In our example the input image is 10x10 and the filter 2x2 so the output image will be 5x5. On the other side if the input would have been 9x9 then the 2x2 filter would not have been able to go through all pixels because the one at the border would not have a pair. That's way we have the option to enable zero padding. This option includes and additional border of pixels with value 0 around the image. This way the information from the border pixels won't be lost when using the padding method.

3 Proposed solution

3.1 Solution description and implementation flow

As mentioned in the previous statements, the best option for creating this project, at least from my point of view, would be the one involving convolutional neural networks. The flow of the implementation should look in the following way. The first part is to import in out project a database of images which will be helping to train the network. This bunch of images should be classified on traffic sign categories, so each image of a traffic sign should correspond to a category. After successfully reading and classifying the data the next step would be to process it in the way we want to send id to the neural network. After doing all the desired transformations on every and each image we have to create the CNN. After creating and initializing the network we have to train it in order to be able to recognize any traffic sign image with an accuracy as high as possible. The training and other implementation methods will be explained in detail in the following section. After we finish training the model we save it as a pickle file. In an another project we create the environment for testing if the created model actually works. Here we first do the necessary settings in order to capture image frames from the video camera. This image will be then scaled and prepared in the way the network wants its input to be and then given to the CNN in order to make a prediction. This prediction will be the class which the network thinks the image corresponds to. So if we show for example an image with a red hexagon on which in the middle the word STOP with white color is written, the network will classify it as a stop sign. One important thing to mention is that when working with CNN model the debugging gets really hard and in order to prevent unwanted behaviour a good solution would be to constantly plot some image samples or other particularities like pixel values or vector lengths and many more in order to make sure that everything looks the way we want it to be. When coming across creating the CNN I've thought of using two different methods in order to analyse which one performs the best,

but well discuss about this when we get to that point. The platform I use is the Scientific Python Development Environment Spyder, which is an integrated development environment(IDE) that is included with Anaconda. The images I'm working with are collected from the german traffic sign benchmark(GTSRB).

3.2 Code analysis

3.2.1 Parameters

The first and very important thing is to create some global variables which will hold data which be changed in order to gain different outputs from the CNN. The most important parameters are the batch size will specifies how many images to be processed together at a time, so if we set it to 50 then the neural network will work in parallel with 50 images at a time. Then we have the steps per epoch, an epoch meaning one iteration over all the training data. So if we have 50,000 images and a batch size of 50 then the number of steps in an epoch would usally be 5,000, but we can preset it to a fixed value. The image dimension parameter is initialized to the format our image will have. So if it's 32,32,3 it mean that an image will have 32*32 size and 3 color channels(RGB). It also means that it holds 32*32*3 pixels. Following are the test ratio and validation ratio parameters. This are very important because with them we specify how much from out data we want to use for training, how much for testing and how much for validating. This means that from the 43 classes holding a total of 34,799 images we don't want to use them all for training so we split them in 3 parts. The test and validation ratios specify how much data we want to use for training respectively for validating so for example if we have 1000 images and the test ratio is 0.2 then 200 images will be used for training, following if the validation ratio is 0.2 then from the remaining 800 images 160 will be used for validation. But before splitting we have to save this images in some variables.

3.2.2 Reading the data

The images are organized in 43 different folders, numerated from 0 to 42, each corresponding to a traffic sign class. Each folder contains a different number of images, each one corresponding to the folder class it has been placed into. After iterating through all the folders we get two numpy arrays: images and classNo. The first one is a collection of all the images and the second one is an array which at each position holds the number of the class, the image in the image array at the same position, corresponds to. So classNo[x] = class number of images[x]. After the is collected, it is split in the training, testing and validation parts, as mentioned above.

3.2.3 Processing the data

Some processing of the data is necessary in order to achieve a better performance. Some of the changes the image will go through are the following. Firstly we were supposed to scale the image to the desired dimension but the images are

already of 32x32 format so we skip this step. The first change is modifying the image from color to grayscale, so instead of holding 3 different values between 0 and 255 for each pixel of the image, now we will hold only one. Then we will be use the equalize histogram function in order to add contrast to the photo. Lastly we normalize the value of each image pixel by dividing its value to 255 in order to have only values between 0 and 1. We then add a depth of 1 in order to go back to the initial format, so switching from (34.799, 32, 32) to (34.799, 32, 32, 1). Following we do a procedure called augmentation. Through this process we modify the image in order to make it more generic. Rotation, scaling and other processes are done. This step is very important because in this way we could train the network with images from a wider range. This process is used also when a traffic sign class lacks on image quantity. The last modification we do is to modify the classNo array into a matrix because this is the data the CNN expects the labels to be.

3.2.4 Creating and using the CNN

In order to create a neural network we have to make use of tensorflow. Tensorflow is an open source intelligence library, using data-flow graphs to build models. Within the tensorflow library there is something called Keras. Keras is a neural network library which provides high level API's. For this project I chose to implement neural networks both using keras but also using tensorflow without it in order to see how they perform. We will focus more on the CNN created by using Keras library functions. The first step when building the CNN is to specify the number of filters and their size in order to feed them when building the hidden layers of the network. Besides covolution we want our network to be able to do some pooling as well so we have to specify the size of the pooling filter as well. The covolution and the pooling methods have already been described in the previous section. We then have to specify the type of our model, sequential or functional. The functional approach is easier to be used and to be understand because it permits to each layer to communicate only with the previous and the next layer, so I have chosen this one. Then we are ready to start creating our layers by calling the Conv2D function with the appropriate parameters. This will take care of the convolution and the MaxPooling2D one will take care of the pooling. In order to make sure that only positive values are assigned to the neurons we make use of the rectified linear unit (relu) function which work as a filter that allows only positive values to pass, the negative ones being converted to 0. Lastly we call the compile method which initializes the graph and in which we have to specify the optimizer, the loss function and the metrics used by our network model. We will use Adam optimizer which is a stochastic gradient descend method, used to find local minimums. The loss function we use is called categorical crossentropy, which is used when an injective function between the input and the output, this being applied to our case, because the input image can have as output only one traffic sign class. Lastly we have to set the metrics, a metric being a function which is used to judge the performance of the model. A metric function is similar to a loss func-

tion, except that the results from evaluating a metric are not used for training the model. We will use an accuracy metric which compute the accuracy rate across all predictions which the network has made. After the model is created we begin to train it by calling the fit generator method with the corresponding parameters. After the training is done we could ask for its accuracy in order to see how well trained it actually is, by calling the evaluate function on some test data which at the time being it has newer been seen by the model. Lastly we save the trained model as a pickle file, which allows the data to be serialized and written into the file as a character stream.

3.2.5 Runnign the program

All there is left to do is to set up the camera and to use our model in order to do the predictions. We first import in this new project the already trained model we've saved before. Then we capture images from the screen and we process them in order to have the same properties, as the ones we have worked with before had. From the csv file which comes along with the german database we assign to each class label (from 0 to 42) its corresponding traffic sign class name. Lastly, we use the predict function on the processed image in order to get the class name of the image. We also have access to the success probability so we can choose to not show the class name only if this rate is above a certain level.

4 Results

All there is left to do is to set up the camera and to use our model in order to do the predictions. We first import in this new project the already trained model we've saved before. Then we capture images from the screen and we process them in order to have the same properties, as the ones we have worked with before had. From the csv file which comes along with the german database we assign to each class label (from 0 to 42) its corresponding traffic sign class name. Lastly, we use the predict function on the processed image in order to get the class name of the image. We also have access to the success probability so we can choose to not show the class name only if this rate is above a certain level. I have also tried to test the same image database with a tensorflow created CNN, this time working with a graph not a model. The results were quite the same when working with fewer than 6000 images per training, testing or validation set because if the working set would been larger, then the compiler would exceed the allocated memory and an error would appear. Other than that, both implementations bring similar results. I've also tried to use a normal neural network without hidden layers but the accuracy would be smaller with at least 20

5 Conclusion

The project was a total success but there are always ways of improvements. This is only a small part when talking a bigger project. An interesting task would be to make the program recognize more than one traffic sign in a single image. But for this very project, I am very pleased with the results.

6 References

https://towardsdatascience.com/recognizing-traffic-signs-with-over-98-accuracy-using-deep-l

https://hackernoon.com/traffic-sign-recognition-using-convolutional-neural-network-8a1f9

https://www.datacamp.com/community/tutorials/cnn-tensorflow-python

https://www.datacamp.com/community/tutorials/convolutional-neural-networks-python

https://www.youtube.com/watch?v=QHra6Xf6Mew

https://www.youtube.com/watch?v=YRhxdVk_sIs

https://www.youtube.com/watch?v=ZjM_XQa5s6s

https://www.youtube.com/watch?v=qSTv_m-KFk0&t=206s

https://www.youtube.com/watch?v=Ilg3gGewQ5U&t=417s

https://www.youtube.com/watch?v=aircAruvnKk&t=991s

https://www.youtube.com/watch?v=ILsA4nyG7I0&t=1182s

https://www.pyimagesearch.com/2019/11/04/traffic-sign-classification-with-keras-and-deep