2021

BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

SPECIALIZATION COMPUTER SCIENCE

DIPLOMA THESIS

Traffic Signs Classification

Supervisor

Lect. PhD. Oneț-Marian Zsuzsanna

Author

Ceaușoglu George Eduard

2021

UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA

FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ

SPECIALIZAREA INFORMATICĂ

LUCRARE DE LICENŢĂ

Clasificarea semnelor de circulaţie

Conducător ştiinţific

Lect. Dr. Oneț-Marian Zsuzsanna

Absolvent

Ceaușoglu George Eduard

Abstract

The main drive behind approaching this subject was the up-and-coming rise of fully autonomous vehicles. Numerous tests have been conducted and it has been shown that self-driving cars are actually a lot safer than human-driven ones. Autonomous cars can improve their safety level in two ways. The first one is increasing the percentage of self-driving cars on the roads. The second one is improving the technology in these cars.

The aim of this diploma thesis is to build a highly accurate model capable of classifying traffic signs in order to improve the safety of fully autonomous cars. The program is structured in 2 parts. The first one is the model. It was developed using a convolutional neural network architecture and it obtains a 98.8% accuracy. The second part is the UI/Server app. It was developed with Python, Flask, HTML and JavaScript. The interface is pretty simple and Contains a ‘Choose File’ button through which the user can upload an image. The image can be sent to the server in order to get a prediction through the ‘Predict’ button. On the server, the image is resized and reshaped and then the previously trained model is used to predict what traffic sign is displayed in the image uploaded by the user. The answer is then sent back to the user in the form of a string that represents the name of the traffic sign.

In the scenario of a correctly predicted traffic sign, the user has the option of uploading the image in order to enhance the dataset.

In a real-world scenario, the software of a self-driving car could incorporate this application by using a photo obtained by the car’s hardware as an input for this model. The class id outputted by the model should then be used by the driving decision system of the car in order for the best driving decision to be taken (e.g.: stop at a ‘stop’ sign, or disable overtaking between the ‘overtaking prohibited’ and ‘overtaking not prohibited anymore’ signs).

1. Introduction

Autonomous vehicles are vehicles that can perceive the surrounding environment through various means and can operate without any human involvement.

The concept of autonomous vehicles is not new. The first successful trials of self-driving vehicles took place in 1950, and since then, a lot of research and experimentation has been conducted [29]. Even though autonomous vehicles existed, they were not fully autonomous. In order for a vehicle to be fully autonomous, it has to be able to maneuver through ongoing traffic and take decisions based on the movements of other automobiles, road markings and traffic signs. Significant progress towards fully autonomous cars has been made in the last few years, with innovations from companies such as Tesla, Nio and many others.

The main concern regarding fully autonomous vehicles is their safety and ability to perceive the road conditions and act accordingly.

A very important part of taking driving decisions is traffic signs. They impose the maximum speed allowed, upcoming turns, risks that the upcoming road is subject to, upcoming lanes, and many other indications and information that must be acknowledged by the driver.

The worldwide regulation of automated cars is only a matter of when, not if. Twenty-nine states in the US have enacted legislation regarding autonomous vehicles, while several countries in Europe and Asia have passed legislation regarding assisted driving technology. Laws regarding fully autonomous driving are currently in progress in a large number of countries.

I consider that the classification of traffic signs is a key part of safe autonomous driving, as they contain a copious amount of information vital to the driver. Vehicles are already collecting information and data through sensors and cameras, and are able to isolate portions of such data. The application receives as input a frame (an image) and outputs the type of indicator, in order for the autopilot to take an appropriate driving decision.

In this diploma thesis, I have tackled the task of classifying and recognizing traffic signs (road indicators). The approach used is making use of convolutional neural networks and their native advantage in recognizing visual data.

2. Convolutional Neural Networks

2.1 Artificial Neural Networks

Artificial Intelligence (AI) “is the science and engineering of making intelligent machines, especially intelligent computer programs” [1]. Intelligence is thought of as an ability of acquiring and applying knowledge and skills. It is a trait specific to humans and a few species of animals, and nowadays an argument can be made that intelligence is specific to some machines as well. Rapidly evolving technology - both hardware and software, allows us to develop very powerful algorithms that run on performant machines. For a long time, humans have tried simulating true intelligence on computers, but in the last 70 decades, significant progress has been made. In the 1950s, Alan Turing proposed the Turing test. This test was designed for detecting human-like intelligence in a machine. Since then, many chatbots have passed this test, but the majority of researchers have shifted focus from testing a program with the Turing test to trying to create truly intelligent and rational artificial intelligence [28].

Machine learning (commonly called ML) “is the study of computer algorithms that improve automatically through experience and by the use of data” [3]. Machine learning aims to address the problem of creating machines that learn and evolve through experience [4]. It combines the fields of computer science and statistics and stands at the heart of AI and data science.

In image processing, the two main learning paradigms are supervised and unsupervised learning.

- Supervised learning: Learning with previously labeled inputs that serve as objectives. There is a set of input information comprised of information-target pairs. This set is usually split into an information vector, and a vector containing one or more associated defined output values for each training example [2].
- Unsupervised learning: is distinguished by the absence of labels in the training set. Most of the time, the criterion for success is a network’s ability to increase or decrease a cost function correlated to it [2].

Data harvesting is a crucial part of machine learning. In order for the training process to succeed in supervised learning, the training data must be structured as follows:

- The content - the data representing what the program should be able to predict
- The correct answer (known as target attribute) - the correct identifier for the data that the model should predict/classify. This is what allows the model to acknowledge if the prediction was correct or not.

A model is a program that has been provided with a set of data and an algorithm to interpret that data (a learning algorithm). The desired result is an improved program that can now make predictions on data that is not in the training set, but has similar characteristics. The model artifact developed by the process of training is referred to as an ML model [4]. The learning algorithm searches the training data for patterns that connect the input data attributes to the goal (the result to be predicted), and it outputs a ML model that uses these patterns. The generated model is then used for making predictions in case of new data, for which it does not know the target attributes [4].

Artificial neural networks (commonly called ANNs) are computer systems that are loosely based on the biological networks that make up both the animal and human brain. An ANN is constructed of nodes (often called artificial neurons). These nodes are connected, much like in a biological brain. Each connection can send a signal to other neurons, similar to how things happen in a human brain. One such neuron has the ability to receive a signal, take some action on it and afterward send other signals to neurons with which it has connections. The motivation behind this architecture is the massive amount of computation that can be done by the human brain. This feat of the encephalon allows us to be exceptionally good at recognition and classification tasks. Such tasks are accomplished using a biological neural network that may be mathematically modeled as a weighted and directed graph of highly interconnected vertices [5]. The most significant advantage of ANNs over traditional systems is their high degree of parallelity, as opposed to the traditional sequentially operated system. In the homo sapiens central nervous system, thousands of neural cells die each year and functionality is almost never affected (excepting extreme cases, where cells from key locations die in major numbers). The extreme parallelity of biological brain networks, as opposed to said sequential design of traditional digital computers, accounts for their insensitivity to injury of a few cells

[6]. In common machine learning tasks, these biologically inspired computational models outperform previous types of artificial intelligence by a wide margin [2].

Artificial neural networks are organized in layers such as convolutional layers, pooling layers, fully connected layers, recurrent layers or normalization layers. These layers serve the purpose of receiving data, processing it, and outputting a certain result [34]. A layer's output usually serves as the next layer's input. Every connection between 2 neurons is associated with a numerical value which represents the strength of the connection and is called weight [34]. The most significant aspect in transforming an input into an output is the weights. This is because when the network gets an input at one node, it is transferred to the next node via the connection between them, but only after it has been multiplied with that connection's weight [34].

Activation functions have the purpose of deciding what neurons should be active based on the weighted sum calculated [35]. This in turn defines the output produced by the layer [19]. There are plenty of activation functions, and some of them are better suited in some cases than others. For example: while a recurrent neural network most often uses sigmoid activation or tanh activation, multilayer perceptrons and convolutional neural networks are usually coupled with ReLU activation.

Gradient Descent "is an optimization algorithm for finding a local minimum of a differentiable function" [37]. Gradient Descent is used for identifying the values of the parameters of a function in order to minimize the cost function [37]. This cost function represents the average of all loss functions (loss functions compute the loss of the model at a certain point in training) [37].

Backpropagation is a common approach for the calculation of derivatives inside an ANN and is vital for the process of training a feed-forward network [36]. The gradient of a loss function can be calculated via backpropagation for each of the network's weights. This allows each weight to be updated independently across several training iterations, lowering the loss function [36]. The gradient is computed while moving back through front through the model, from the final layer to the first [36].

2.2 Convolutional Neural Networks

Convolutional neural networks (commonly called CNNs) are some of the most remarkable types of artificial neural networks. They are typically employed for the solving of complex pattern recognition tasks, usually image-related. Their accurate, yet simple architecture makes getting started with ANNs much easier.

Many image-based pattern recognition tasks rely on supervised learning for classification. CNNs were developed with the classic ANNs in mind, therefore, they are composed of nodes that have the ability of self-optimization. Every neuron can take an input and perform an operation on it. The main significant distinction between convolutional neural networks and standard artificial neural networks is the fact that convolutional neural networks are mostly utilized in image pattern recognition. This allows for the encoding of properties associated with images in the architecture, placing the network in a better position for image-driven assignments while decreasing the amount of parameters needed for the setup of the model [2].

Most of the time, a convolutional neural network is comprised of four types of layers:

- Convolutional layers
- Pooling layers
- Fully connected layers
- Dropout layers

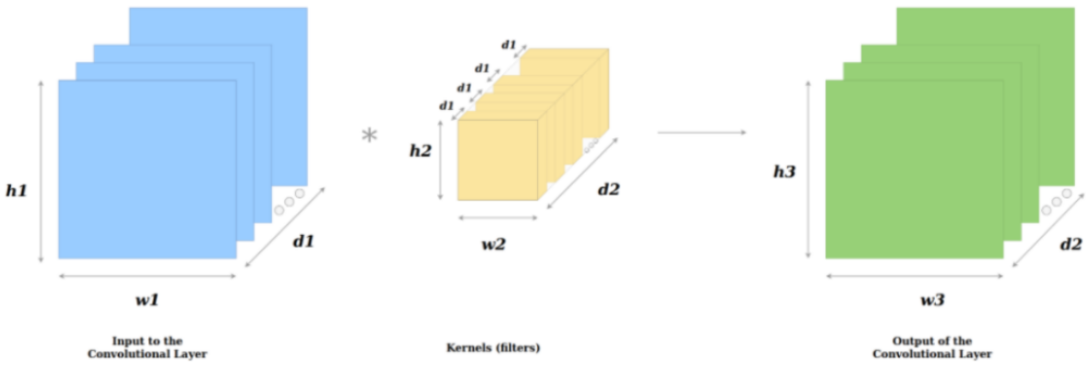

Fig 2.1: Visual representation of a convolutional layer [7]

1. The convolutional layer (See Figure 2.1), as the name suggests, is of critical importance to the operation of CNNs. The parameters of this type of layer focus around learnable kernels (matrices that move over the input data and are used for the computation of dot products with subsets of the input data). The spatial dimensionality of these kernels is usually not high, yet they spread across the full input depth. When data is processed inside a convolutional layer, each filter is convolved by the layer across the spatial dimensions of the input for building a bi-dimensional activation map [2]. These activation maps can be visualized (See Figure 2.2).

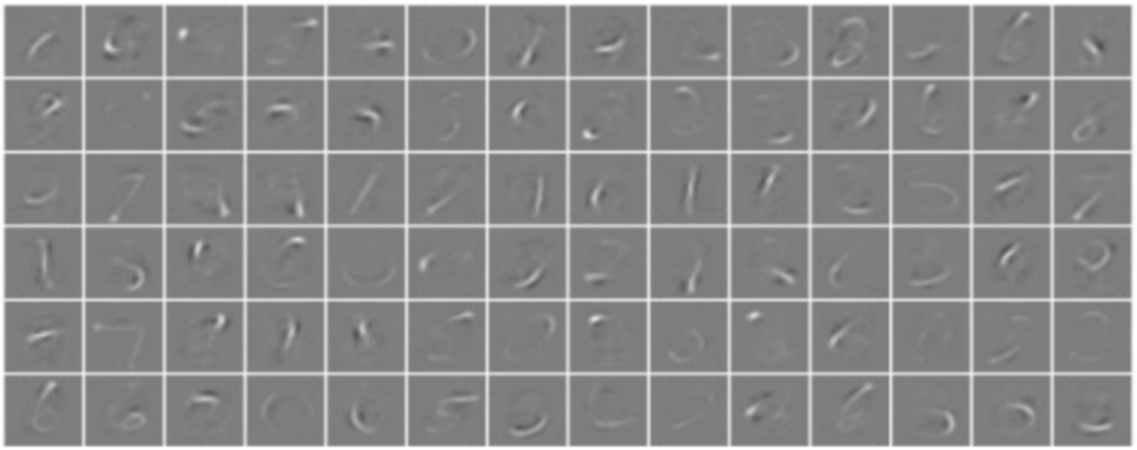

Fig 2.2: Activations from the first convolutional layer of a simple convolutional neural network after it has been trained on the MNIST handwritten digits database. Upon inspection, characteristics unique to certain digits can be observed [2].

The scalar product is computed by the multiplication with a subset of the input data for every value in the current kernel, as we progress through the input (See Figure 2.3). The network will acquire information on kernels that ‘fire’ when they perceive a peculiar feature located at a particular position in the input data. These events are often known as ‘activations’.

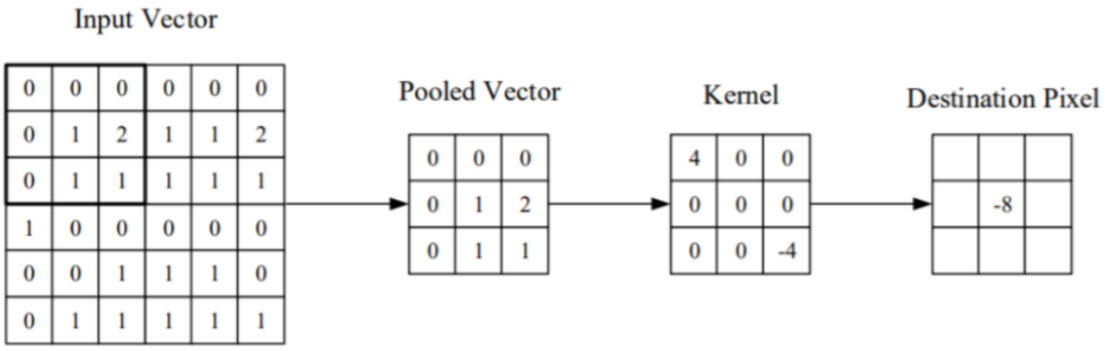

Fig 2.3: Visual representation of a convolutional layer. The centre element of the kernel is placed over the input vector, of which is then calculated and replaced with a weighted sum of itself and any nearby pixels [2].

To every kernel, there will be a corresponding activation map, which will be stacked along the depth dimension in order to generate the convolutional layer's output volume. Using image inputs to train ANNs leads to models that are too large to train properly. This is because of the fully linked nature of traditional ANN neurons. In order to deal with this issue, in a CNN, each neuron is connected solely with a fraction of the total input volume. The size of said region is often called "the receptive field size" [2]. The depth of the input is almost always equal to the degree of connectivity throughout the depth. For example, if a standard network input is 64 x 64 x 3 (an RGB-coloured image with length = 64 and width = 64), and the receptive field size is set to 6 x 6, each neuron from the convolutional layer would have a total of 108 weights (6 x 6 x 3, with 3 being the degree of connectivity across the volume depth). In contrast, a standard neuron in a regular artificial neural network would contain 12288 weights, nearly 114 times as many. By optimizing the output of the model, convolutional layers considerably lower the model's complexity. Three hyperparameters, the depth, the stride, and the zero-padding setting, are used for this optimization.

The number of neurons contained by a layer can be used in order to set the depth of the output volume of the convolutional layers. This may be seen in other types of artificial neural networks, in which all neurons from the hidden layer are directly coupled to each other [2]. The overall number of neurons in the network can be greatly reduced by lowering this hyperparameter. This must be done carefully

because lowering the number of neurons can lower the pattern recognition ability of the model as well.

The stride is used to “set the depth around the spatial dimensionality of the input in order to place the receptive field” [2]. Should the stride be set to 1, there will be a substantially overlapped receptive field containing extraordinarily big activations. If the stride is set to a larger number, the amount of overlapping would be limited and the output size would be smaller.

Zero-padding is represented by the process in which the input border is padded with pixels (usually of value 0). It is a powerful method for managing the output size. Because by using this method, the convolutional layers’ output spatial dimensionality is altered, the following formula is needed in order to compute this [2].

$$\frac{(V-R)+2Z}{S+1}$$

- V represents the input volume size (height x width x depth)
- R represents the size of the receptive field
- Z represents the amount of zero padding that has been set
- S represents the stride

Should the result of this computation not be an integer, then the stride has not been set correctly, and the nodes will not be able to properly fit over the input [2]. This is a problem that can be reduced by using parameter sharing - setting the same weights to every neuron inside a feature map.

Despite using these methods, the models would still be too large to be practical if used for a real-dimensionality image input. The idea behind parameter sharing is: if a region feature is valuable to be computed in one area, it probably is beneficial in another as well. The number of variables forged by this layer can be drastically reduced by constricting each activation map from the output volume to have the same bias and weights [2]. The outcome is that each neuron represents the overall gradient as the backpropagation stage unfolds - only a set of weights is being updated, not each and every one of them.

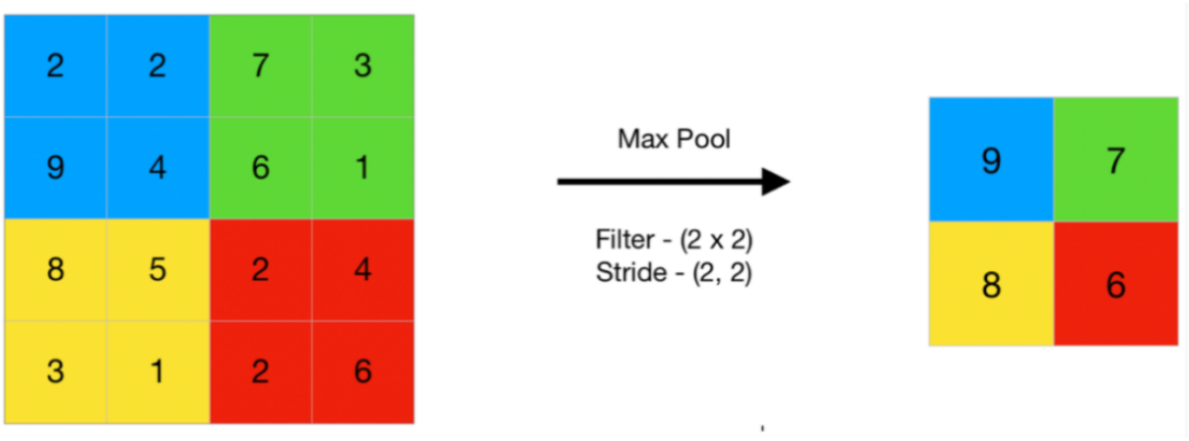

Fig 2.4: Visual representation of a max-pooling layer [8].

2. The pooling layer has the goal of gradually reducing the dimensionality of the representation, lowering parameter number and the computational complexity of the model [2]. The pooling layer adjusts the size of each activation map of the input by the use of the 'MAX' function. In most convolutional neural networks, they are represented by max-pooling layers (See Figure 2.4) with kernels of size 2 x 2 and a stride of 2. In this scenario, this reduces the size of the activation map by 75%, while keeping the depth volume at its original size. Because of the pooling layer's destructive character, there are two approaches to max-pooling that are generally used:

- The stride and the pooling layer's filters are set to 2 x 2. This allows the extension of the layer throughout the input's whole spatial dimensionality.
- Overlapping can be employed furthermore; 2 is chosen as the stride, and 3 for the kernel size. Choosing a kernel of a larger size than this oftentimes considerably decreases the model's efficacy[2].

Besides max-pooling, convolutional neural networks may have general-pooling. In general-pooling, the neurons of which the layers are made up of can accomplish a variety of tasks such as average pooling or L1 / L2 normalization.

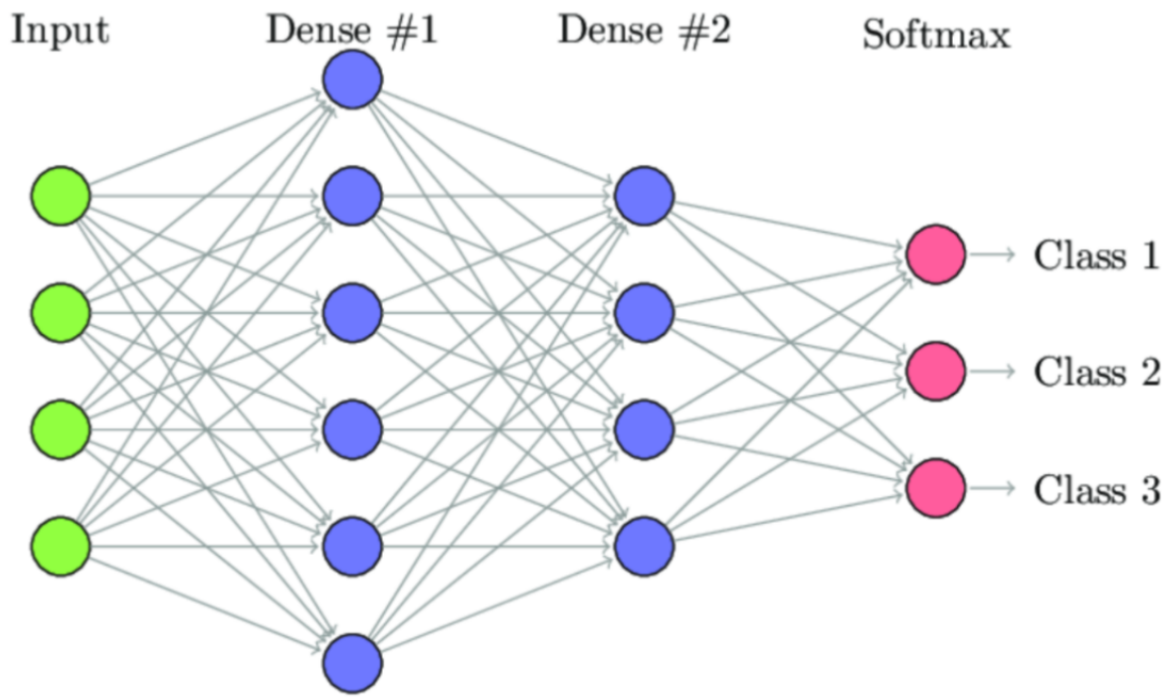

Fig 2.5: Visualization of a fully-connected network [9].

3. The fully connected layer "contains neurons of which are directly connected to the neurons in the two adjacent layers, without being connected to any layers within them" [2]. This is similar to how nodes are placed in standard ANN models. The input for the current layer is represented by the output from the last convolutional or pooling layer. After being flattened (the 3D matrix received as output from the final layer is unrolled into a vector), this output is fed into the fully connected layer [7].

An example of a fully connected layer may be seen in Figure 2.5.

The softmax function is generally used in the last layer of a CNN. The Softmax function "is a normalized exponential function which transforms a D-dimensional original vector with arbitrary real values into a D-dimensional probability vector with real values in the range [0, 1] that add up to 1" [10]. This is a very useful activation function because it allows for the transformation of irregular data into data that can be interpreted as probabilities (values fall within the range 0, 1). For negative or small inputs, the softmax converts them into a small probability, whereas, for large inputs, the results are large probabilities. The softmax formula is as follows [33]:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_i}}, \text{ where}$$

- $\vec{z}$ represents the input vector
- z_i represents the elements of the input vector - can take any real value; negative, positive, or zero
- e^{z_i} represents the standard exponential function and is applied to each element of the output in order to obtain strictly positive values.
- $\sum_{j=1}^{K} e^{z_i}$ represents the formula that ensures that all output values will be in the 0 to 1 range and will finally sum up to 1, thus forming a valid probability distribution
- K represents the number of classes of the classifier

In an ANN, “the activation function is responsible for transforming the summed weighted input from the node into the activation of the node or output for that input” [25]. “The rectified linear activation function or ReLU for short is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero” [25].
The error for the existing form of the model must be calculated periodically in order to optimize the model. This necessitates the selection of an error function, also known as a loss function. The loss function is important in order to minimize the loss in future evaluations [25].

4. The dropout layer is used during training. Its purpose is to set a percentage of the input to zero, thus ignoring some of the features of the data. This process is very important because it helps prevent overfitting by memorization of specific features in the input.

The architecture of convolutional neural networks is quite consistent. There are convolutional - pooling pairs followed by fully connected layers. Another typical CNN architecture is obtained by the stacking of two convolutional layers before each pooling layer. This approach is often preferred because it enables the selection of more intricate properties of the input vector (See Figure 2.6). Splitting sizable convolutional layers, thus forming plenty small ones represents a good strategy for reducing the amount of computational complexity within a layer [2].

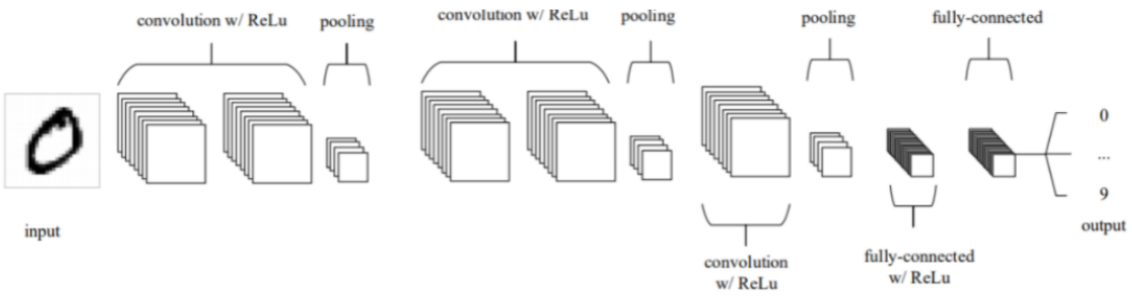

Fig 2.6: In this popular version of CNN, convolutional layers are continually layered between ReLus before passing through the pooling layer and then one or more fully linked layers [2].

CNNs stand apart from other types of artificial neural networks because they use information about a specific sort of input rather than focusing on the full issue domain. As a result, a considerably simpler network design is required.

3. Traffic sign recognition

The idea of autonomous cars is not a new concept. The challenges that a self-driving car poses virtually imply that the vehicle must integrate into its architecture a computer. In order for a car to be fully autonomous, it has to be able to input its own commands (like steering, accelerating, braking, etc). In order for a car to know when it should do these things, it should perceive the road as a human does through sight. Hardware such as sensors and cameras can gather data, and video processing software can transform this data into information processable by the central unit of the car. The data that guides how a car should be driven can be broken up into two categories:

- Direct indicators: road markings painted on the concrete, milestones and traffic signs.
- Indirect indicators: weather conditions, surroundings,

The problem of classifying traffic signs has been tackled before. A great event that sparked interest in this domain is the “German Traffic Sign Recognition Benchmark” - a classification competition organized at The International Joint Conference on Neural Networks in 2011. At this event, participants were provided with the GTSRB dataset and their aim was to build a classification model with as high of an accuracy as they could get. This generated further interest, and with an extensive dataset made publicly available to anyone that wanted to take a shot at the problem, many have tried to obtain the best accuracy possible.

Pierre Sermanet and Yann LeCun (a founding father of the convolutional networks, and the proposer of the LeNet Architecture [31]) have attempted the competition, and in the first phase, their model has achieved the second-best accuracy - 98.97%. At a later try, by enlarging the capacity of their network and converting the images to grayscale, they achieved an improved accuracy of 99.17%. The pair opted for a convolutional networks approach, as opposed to often used vision approaches that make use of features such as histogram oriented gradients (HOG), or scale-invariant feature transform (SIFT) (used for extracting features from data) [30]. Their network was implemented in C++, using the EBLearn package, and they used a two-stage architecture. Their architecture differs from that of a traditional

CNN in that they use connections that skip layers, and pooling layers that use corresponding pooling ratios on the connections that skipped or not [30]. Another difference is the fact that their architecture is not strictly feed-forward. The result generated by the first stage is used as an input for both the second stage and the classifier (See Figure 3.1 for visualization) [30]. The accuracy obtained was higher when they used a double convolution approach, as opposed to when they only used convolution once, on the output of the first stage. While their result is admirable, better accuracy has been obtained by others who attempted this task.

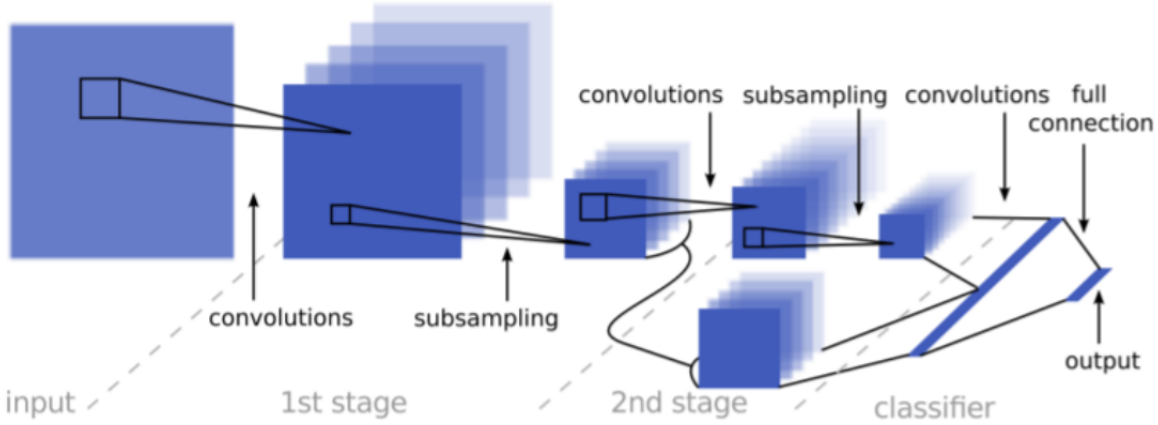

Fig 3.1: Visualization of the model architecture used by LeCun and Sermanet [30].

Zhang et.al. pondered the fact that traffic sign recognition technology is usually used in settings in which resources are not the most abundant hardware-wise (such as cars). The conclusion was that the models should be as lightweight and as easy to use as possible. They have developed a pair of teacher-student networks. Information used during the training of the teacher network is used in the smaller (student) network as well. The student network is composed of 5 convolutional layers and a dense layer, thus enabling facile mobile deployment [32]. The process of transmitting information from a network to another is called knowledge distillation[32]. The Knowledge distillation process has the purpose of training the smaller network by using the softened output obtained by the bigger network on the dataset (Fig 3.2). Both networks use the same input for training. The teacher network is trained first, and the parameters obtained on the

training set have the sole purpose of “guiding the updating of the student network parameters” [32].

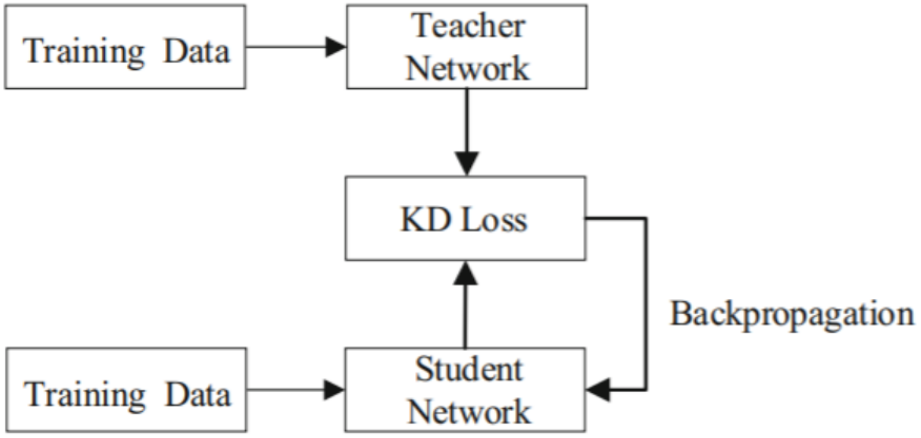

Fig 3.2: The process of knowledge distillation [32].

In order to make the student network as lightweight as possible, after training, the most unimportant channels were identified and then pruned, resulting in a final network consisting of 0.8 million trainable parameters. When testing this network, the result was a remarkable 99.61% accuracy on the GTSRB dataset.

4. The application

4.1 The dataset

In real-world settings, traffic signs have a wide range of visual appearances. Changes in light, shifting weather conditions, or partial occlusions, for example, have an impact on how road signs are perceived. In practice, a large variety of various traffic sign classes must be recognized with great accuracy. Traffic signs have been created to be as easy to read as possible for humans, who display great performance at recognizing them. Computers, however, are not the best at classifying visual information, but both ML algorithms and image processing are constantly being refined in order to improve on this task. The fact that the output category can be of more than two types makes traffic sign recognition a multi-category classification task. It is a difficult real-world computer vision problem with a lot of practical implications, and it has been a research focus for a long time [11]. Human drivers are able to detect and recognize road signs with very high accuracy because these signs have been specially designed in this fashion (to be very easily recognized by humans). Form, color, text and graphics are all used in ways that adhere to defined design principles. These enable a wide range of class variants. Signs that have the same general meaning, such as speed restrictions, have a similar appearance, resulting in subsets of traffic signs that are quite similar. Changes in illumination, motion blur, occlusions, rotations, and climate conditions all add to the spectrum of visual appearance variations that a classifier must deal with [11].

The dataset chosen for this take on traffic sign classification was the “German Traffic Sign Recognition Benchmark” (GTSRB). It is a dataset containing images of german traffic signs, and is split up into 43 classes. The dataset was built using approximately 10 hours of footage captured while driving on various types of roads in Germany during the daytime, in March, October, and November of 2010 [11]. Data collection was performed using the NISYS Advanced Development and Analysis

Framework (ADAF) [11] (See Figure 4.1 for a snapshot from the data harvesting process).

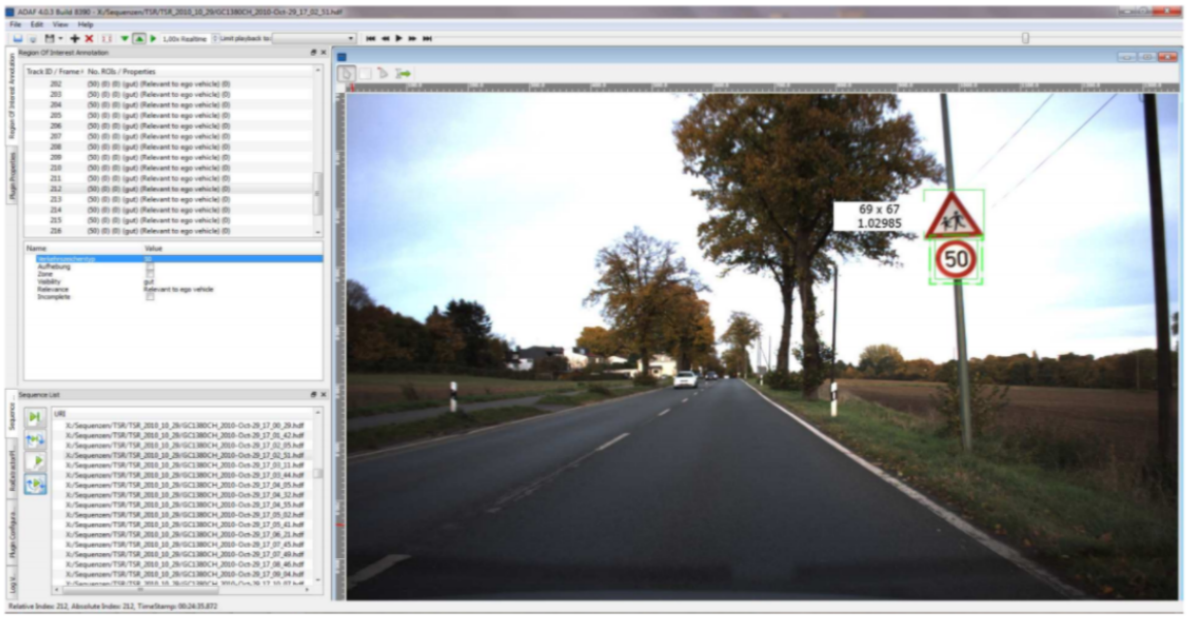

Fig 4.1: Screenshot taken during the manual annotation process using the NISYS software [11].

Several images from a single track are saved because of different lighting conditions and altering of the perspective due to the velocity of the car. Images snapped at a greater distance have lower resolution, while those at shorter distances may have motion blur. The contrast in the images from a single track of a traffic sign can be seen in Figure 4.2 [11].

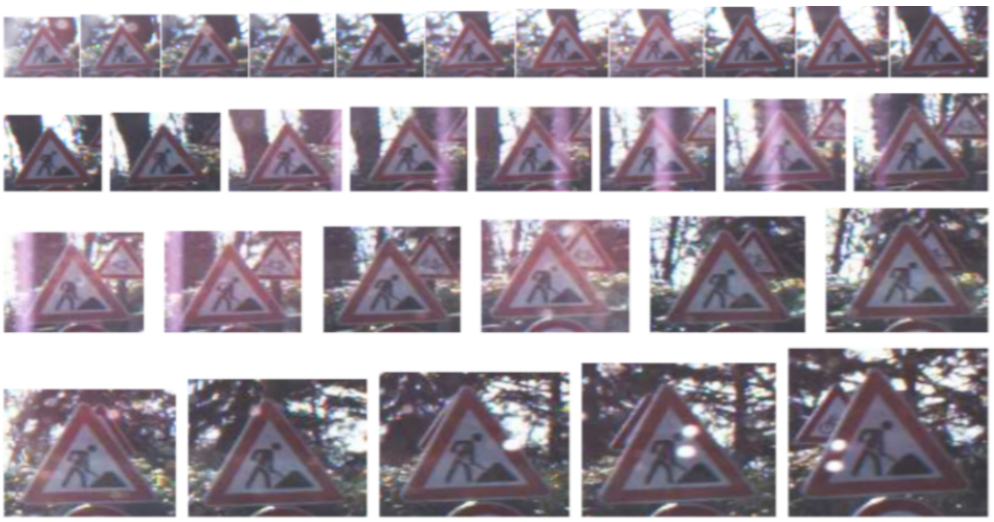

Fig 4.2: Snapshots taken when passing a particular traffic sign instance [11].

After the selection technique was applied, the resulting dataset consisted of 51,840 images split into 43 classes. A single example from each class can be seen in Figure 4.3.

Fig 4.3: A random representative sign from each of the 43 classes of the GTSRB [11].

The data is structured as follows: there is one directory per class, and every directory contains a CSV file containing information about the images in that class. The images contain one sign each and have a border of 10% around the sign (the border is of at least 5 px). The format used for storing the images is portable pixmap (ppm), image sizes span between 15 x 15 and 250 x 250 px. Along with every image, there are 4 coordinates provided. They are used to identify the region of interest (ROI) - the traffic sign inside the image (See Figure 4.4 for visualization) [11].

For the training of my model, I have chosen a subset of the GTSRB. This subset consists of 39,209 images. The whole set of images could not have been used because not all images were labeled. A plot of the dataset as traffic sign frequency by Class ID can be observed in Figure 4.5. Each image was cropped according to its own ROI indicators and then resized to a size common to all images. The resizing of the images was done using cv2.resize from the cv2 library. In deep learning, resizing

images is an important preprocessing step because models train faster on smaller inputs. The dataset was split in a 60 - 20 - 20 ratio. 60% of the images were used for the training of the model, 20% for testing, and 20% for validation. Before being split, the dataset is randomized because images are loaded in the program in blocks formed by class id. The configuration of the data after the split is saved in .npy files for use in other sections of the program.

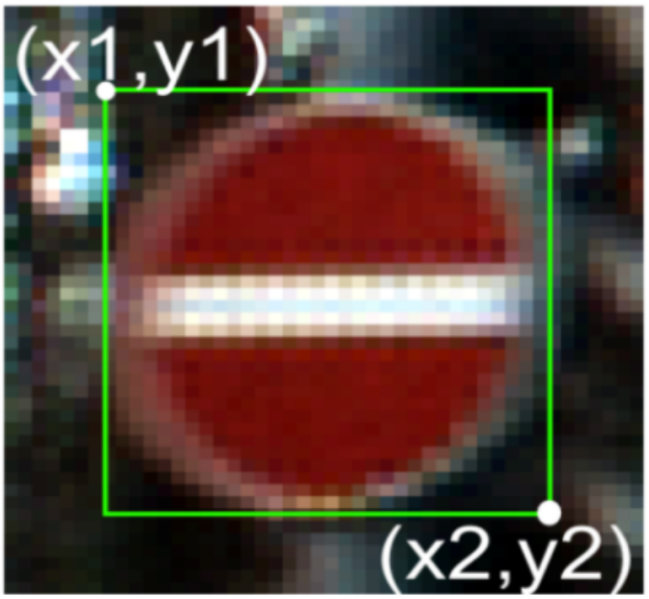

Fig 4.4: Visualization of the ROI [11].

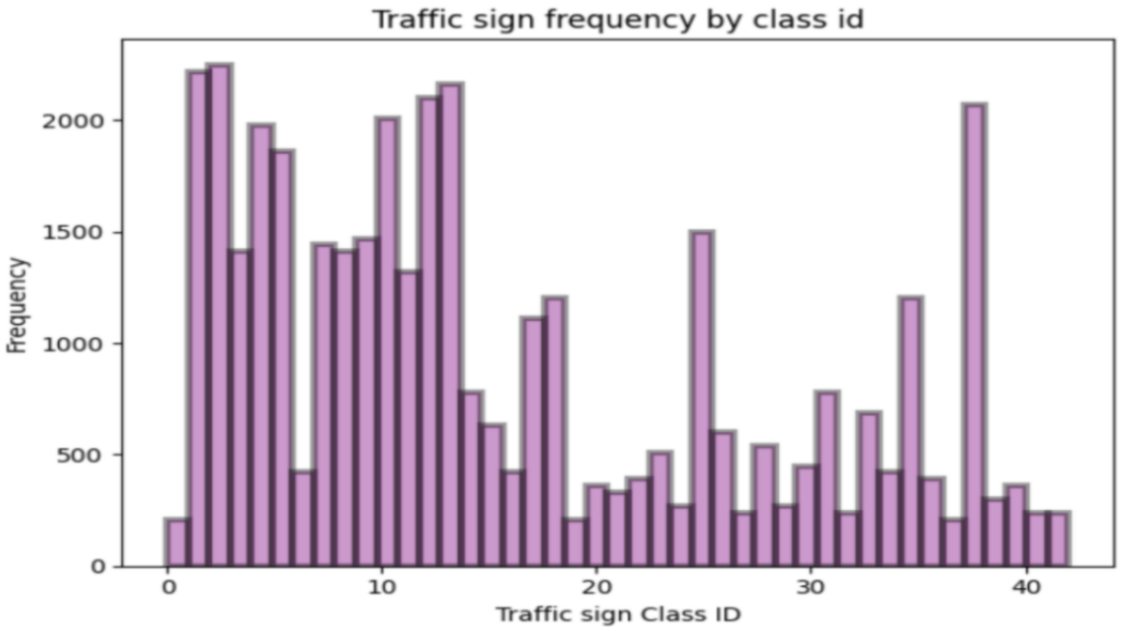

Fig 4.5: Plot of traffic sign frequency by class id. Source: Model.plotDataset() method.

4.2 The Architecture of the Model

Keras is “a library that provides highly powerful and abstract building blocks to build deep learning networks” [12] and it supports both CPU and GPU computation. Theano and Keras are used to create the building blocks provided by Keras [12]. The Sequential construct is used to define a model, allowing the user to add and customize layers. A user can create a network by adding one or more layers. There are six types of layers that comprise the model developed for this application:

- Conv2D
- BatchNormalization
- MaxPooling2D
- Dropout
- Flatten
- Dense

The Conv2D layer is a two-dimensional convolution layer, often used for spatial convolution over images [13]. Convolution is the process of applying a filter to an input in order to produce an activation (See Figure 4.6). When the same filter is applied to an input multiple times, a feature map is created. This feature map displays the positions and the strength of a recognized feature in the input. In CNNs, similar to ANNs, the convolution is the linear process involving the multiplication of a set of weights with the input. This operation is done between an array of input data and a 2D array of weights (typically named a filter or a kernel), because the approach was created for bi-dimensional input. The repeated application of the same filter across the same image is a powerful idea. If the filter’s aim is to detect a specific feature in the input, regardless of the placement inside of the input, this process allows for the discovery of the feature anywhere in the image. This capacity is known as translation invariance, which refers to “the general interest in whether the feature is present rather than where it was present” [19].

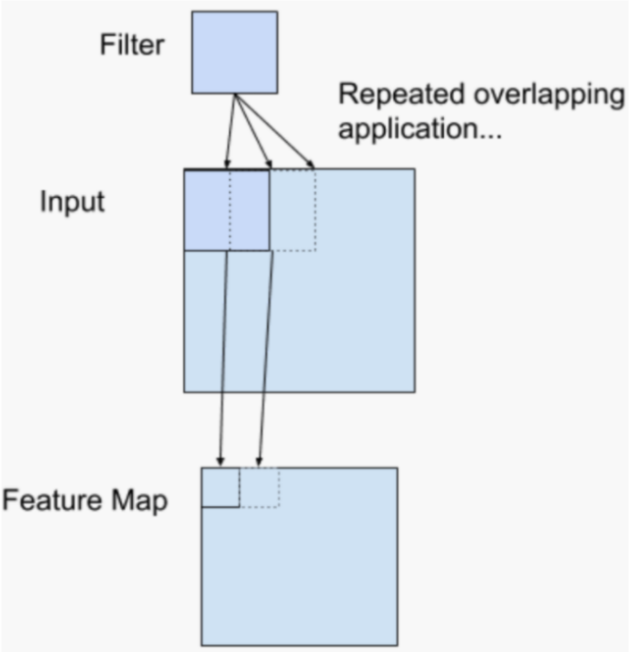

Fig 4.6: An example of an application of a filter on a 2D input. Source: https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/

The BatchNormalization layer makes use of a transformation in order to keep the mean output value near 0 and the output standard deviation near 1 [14]. If this layer is used in the training phase, (model.fit()) the current batch of inputs' mean and standard deviation are used to normalize the layer's output. For every channel that is being normalized, the value returned by the layer is:

$$gamma * \frac{batch - mean(batch)}{\sqrt{var(batch) + epsilon}} + beta, \text{ where:}$$

- gamma represents a learned scaling factor with the initial value of 1
- epsilon represents a small constant
- beta represents a learned offset factor with the initial value of 0

If the layer is used during the inference phase (model.evaluate() or model.predict()), the normalization of the layer's output is done using the moving average of the mean and the standard deviation of the batches encountered during training [14]. The return value is [14]:

$$gamma * \frac{batch - self.moving\_mean}{\sqrt{self.moving\_var + epsilon}} + beta,$$ where:

self.moving_mean and self.moving_var variables of the non-trainable type, and every time the layer is called in training mode, get updated by the following formulas [14]:

- moving_mean = moving_mean * momentum + mean(batch) * (1 - momentum)
- moving_var = moving_var * momentum + var(batch) * (1 - momentum)

As a consequence, only after being trained on data with comparable statistics to the inference data, the layer will normalize its inputs during inference. The input can be of any shape, and the output shape matches the one in the input.

The MaxPooling2D layer handles the max pooling operation over bi-dimensional spatial data. The process consists of downsampling the input together with its spatial dimensions (width and height). This is achieved by selecting the largest value for each channel of the input over an input window (windows size is determined by pool size) [15]. The placement of the window is shifted along both dimensions by predetermined strides. The impact of the stride size choice over the volume of the output can be seen in Figure 4.7. The input is a 4D tensor, while the output shape depends on the padding option used [15]:

- padding = ‘same’ => output_shape = math.floor((input_shape - pool_size) / strides) + 1 (when input_shape >= pool_size)
- Padding = ‘valid’ => output_shape = math.floor ((input_shape - 1) / strides) + 1

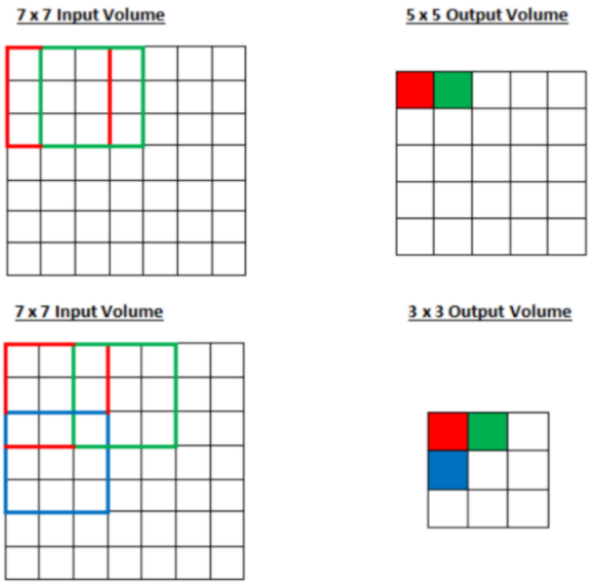

Fig 4.7: Max pooling using a stride of 1 (above) and a stride of 2 (below). Source: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/

The Dropout layer, whose primary purpose is to minimize overfitting, changes input units to 0 at random with a set frequency (called rate) at each step during training time [16]. Dropout refers to the practice of disregarding units (neurons) during the training phase of a particular set of neurons, basically resulting in a smaller network [20]. See Figure 4.8 for visualization. Inputs that aren't set to 0 are scaled up by $\frac{1}{1-rate}$, so that the total sum of the inputs remains the same [16]. This layer is only applied when the model is being trained (when the training parameter is set to True). This is done in order to not lose any data during the inference process. The training parameter is True by default when model.fit() is being used [16].

The Flatten layer takes a pooled feature map and turns it into a single column which can be supplied to a fully connected layer (dense) [17]. This process does not modify the size of the batch. See Figure 4.9 for the visualization of this process.

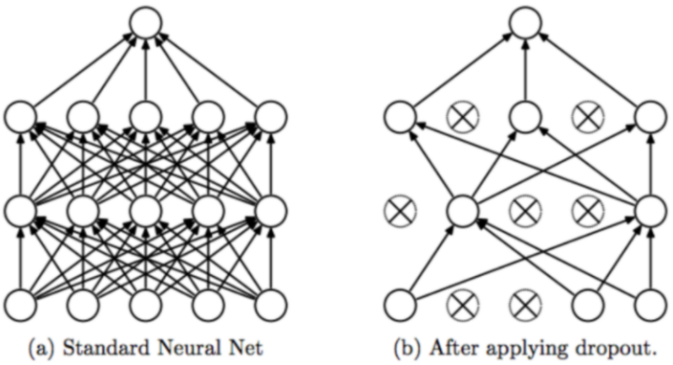

Fig 4.8: Visual representation of the effect of dropout on an ANN [20].

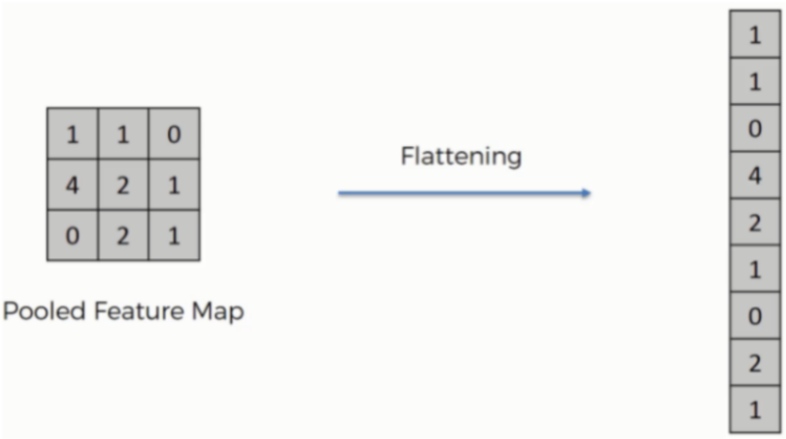

Fig 4.9: Visual representation of the flattening process. Source: https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-3-flattening

The dense layer is a deeply connected neural network layer, meaning that all neurons from this layer receive input from every neuron in the previous layer. A matrix to vector multiplication is done in the background by the dense layer. Backpropagation may be used to train and update the values in the matrix (the parameters) [18]. The dense layer is most often the last layer in a CNN model's architecture.

The model that I have developed is comprised of three (convolution, convolution, max-pool) tuples, followed by fully connected layers and the final output layer. Batch normalization layers are used after each convolution for the purpose of standardizing the inputs for the next layers. Dropout layers are applied after each max-pooling and dense layers, with the purpose of omitting some neurons in order to

prevent the overfitting of the network. The flatten layer is used for converting the max-pooled features into a single vector which is used as input by the fully connected layer. The total number of trainable params of the network is 3,489,595.

4.3 The Design

The application is organized into two parts.

The first part is the model. The model was implemented in python, and the Sequential Model from Keras was used. The model is implemented in the ‘Model’ class and implements methods for loading the data, randomizing the data, splitting the dataset into training / testing / validation subsets, saving the distribution in separate files, building and training the model, saving the model, etc. An object of type model is being instantiated in the controller, where the paths for loading the data and saving the model are provided. The necessary methods are called here in order for the model to be trained and saved.

The second part is the server. The server has been implemented in python as well, employing the help of the Flask framework. An object of type Server is instantiated in the controller. The ‘Server’ class inherits the FlaskView class and implements methods for generating the user interface for getting the prediction for the user-uploaded image, as well as displaying the prediction to the user. If the prediction is deemed correct by the user, the image is saved in the ‘/images/classID’ folder for enhancing the dataset.

The ‘Controller’ contains the methods for generating and training a model, creating a server, and running it.

An overview of the classes’ attributes and methods and the relationships between them can be seen in the class diagram (Figure 4.10).

An overview of the flow of the application can be observed in the sequence diagram from Figure 4.11.

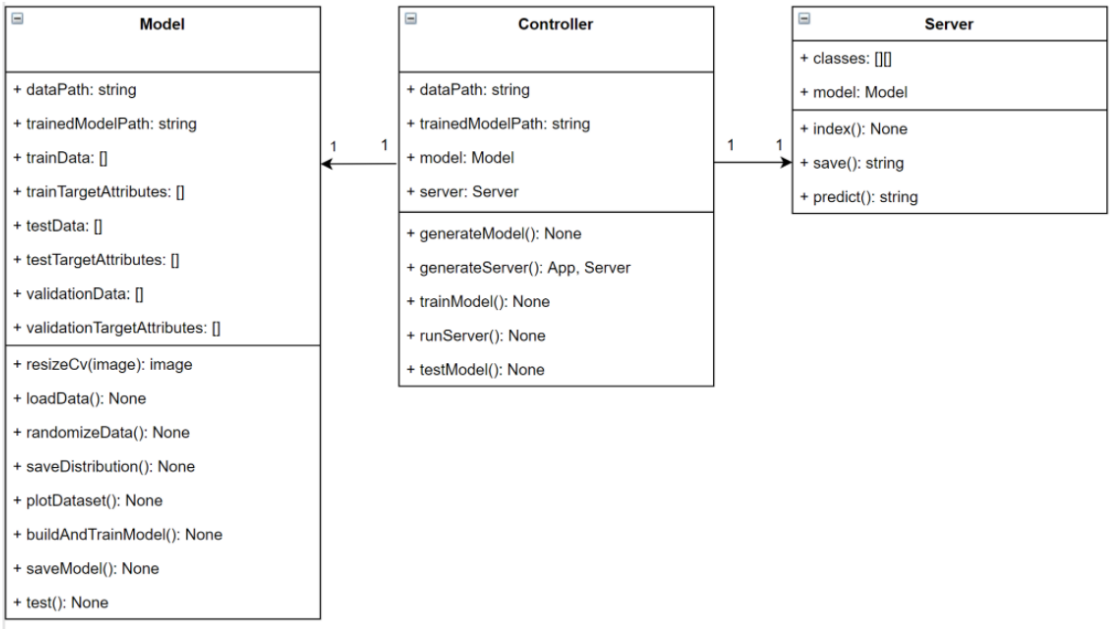

Fig 4.10: Class diagram representing the structure of the program, the relationships between classes, and an overview of the attributes and methods for each class.

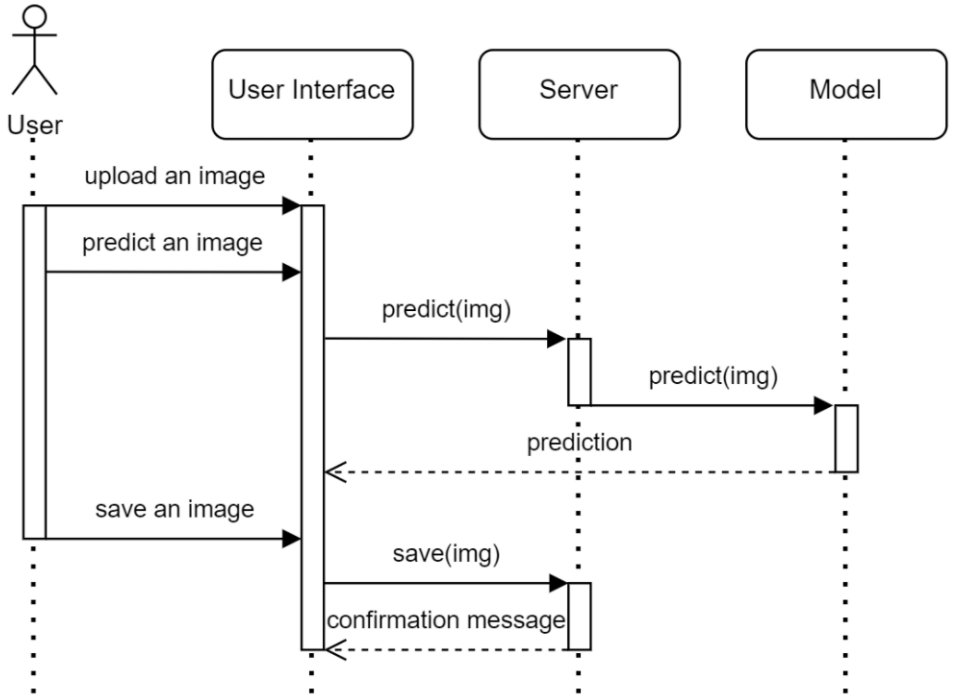

Fig 4.11: Sequence diagram

4.4 Implementation. Technologies

For developing this app, a considerable number of technologies have been employed, ranging from libraries used to render plots of data distribution, to frameworks used in building artificial neural networks.

An integrated development environment (IDE) “is a software suite that consolidates basic tools required to write and test software” [21]. It provides extensive software development capabilities for programmers. An IDE usually consists of a source code editor, a debugger, and build automation tools [22]. The integrated development environment chosen for this project is PyCharm - the python IDE developed by Jetbrains.

The front-end of the application has been developed through the use of HTML and JavaScript. HTML stands for HyperText Markup Language, and “it represents the standard markup language for texts that are planned to be accessed through a web browser”. JavaScript “is a web programming language” and allows for the dynamization of web pages, and for the interaction of the user with the client-side script [24].

In this application, HTML was used for the creation of the buttons seen by the user, while JavaScript allowed for the possibility of the user interacting with these buttons.

The Flask framework for building the web application and connecting the buttons in the front-end to the methods in the back-end. Flask is a python API (application programming interface) used for the building of web applications. The methods used in the back-end for getting a prediction from the trained model or saving the correctly predicted image in order to enhance the dataset were developed in python, and use libraries such as SciPy, NumPy, or OpenCV. The ‘imread’ method from the SciPy library was used for reading the images sent from the user interface to the back-end for getting predictions or saving. The method ‘argmax’ from the NumPy library has been used for getting the id of the class predicted by the model. The model was loaded using the load_model function from Keras. The ‘os’ module from python is used for building the paths used for saving the images that were correctly predicted.

The model used for training was developed in python, employing the help of the Sequential model from Keras. The images from the dataset were processed

using libraries such as Pandas, SciPy and OpenCV, in order to obtain images of a standard size from the varying size images stored in the '.ppm' files.

The data distribution plot and the confusion matrix have been obtained by using two of the most useful libraries when it comes to data visualization and graphic plotting, Seaborn and Matplotlib.

4.5 Testing

Given that the application consists of a model and a minimalistic user interface destined for interacting with the trained model and getting predictions in a visual manner, the testing of the app was preponderantly done through hands-on testing. I have interacted with the user interface in order to make sure that the correct messages are displayed to the user, the program functions properly overall and presents no unusual behavior. Assertions have been used in order to test if the correctly predicted images have been saved, and to test if the 'Model' class functions properly. The method testModel() inside the class Controller has been used to assess functionalities of the model, such as correctly saving paths, correctly distributing data into training / testing / validation subsets, or saving the trained model for further use.

4.6 User Manual

In preparation for running the app for the first time, the user should make sure that the file 'Application/images/imageCoutnter.txt' contains only a zero ('0'). The number in this file will be incremented by one every time a user saves a correctly predicted image, thus assuring a unique filename each time an image is saved (the filename is the number found in imageCounter.txt + '.jpg')

For running the app, the user only has to work with the 'main.py' file. See Figure 4.12 to see the structure of 'main.py'.

```
from Application.Controller import Controller


dataPath = 'C:/Users/eduar/Desktop/Bachelors Thesis/GTSRB/Final_Training/Images'
trainedModelPath = 'C:/Users/eduar/Desktop/Bachelors Thesis/Trained Model/Trained'

controller = Controller(dataPath, trainedModelPath)

controller.testModel()      #1
controller.trainModel()     #2
controller.runServer()      #3
```

Fig 4.12: Snapshot of the main file

The user must provide two paths. In the variable dataPath, the path to the dataset must be stored. In the above example, 'C:/Users/eduar/Desktop/Bachelors Thesis/GTSRB/Final_Training/Images'. In the variable trainedModelPath, the path to where the model should be stored after training. In the above example, 'C:/Users/eduar/Desktop/Bachelors Thesis/Trained Model/Trained'.

The following three instructions after the instantiation of the controller (denoted with #1, #2, #3) should be used depending on the desired effect. Instruction #1 tests the Model class, impacting nothing in the application. Instruction #2 generates and trains a model on the input provided. Instruction #3 runs the server. The server can be accessed through an internet browser such as Google Chrome or Microsoft Edge, by pasting the following address into the address bar: 'http://127.0.0.1:5000/'.

After accessing the aforementioned address, the user is presented with the interface (See Figure 4.13)

Fig 4.13: user interface

By pressing the Choose File button, the user is presented with a pop-up window within which they have to select an image from their device (See Figure 4.14).

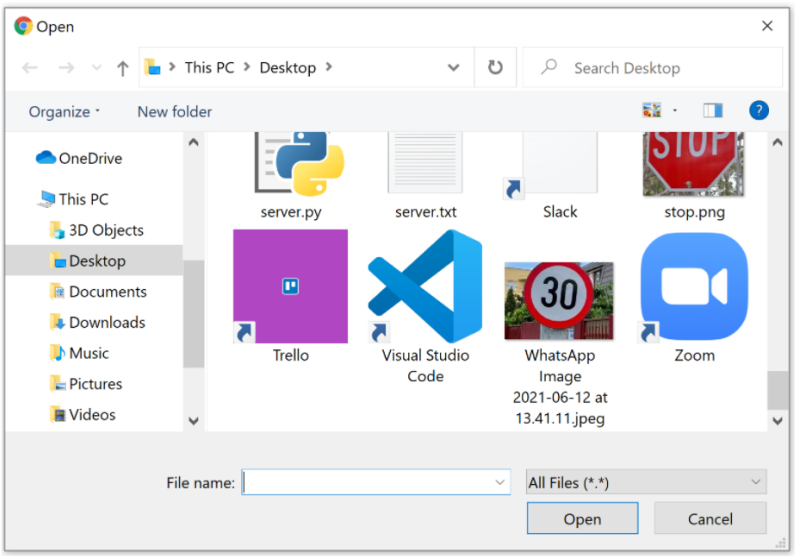

Fig 4.14: user interface

If the file selected by the user is not an image, after pressing either the 'Predict' or 'Save' button, the user will be prompted with a warning message (See Figure 4.15).

Fig 4.15: user interface

If the file selected by the user is of an appropriate type, the image is displayed in the browser, and if a warning message previously existed, that message gets deleted (See Figure 4.16).

Fig 4.16: user interface

After pressing the ‘Predict’ button, the user will be presented with the name of the predicted traffic sign, and is prompted to save the image (hit the ‘Save’ button) if the prediction was correct (See Figure 4.17).

Fig 4.17: user interface

After saving the image, the user receives a thankful message and the process can be restarted by uploading a new image, getting a prediction for that image, and so on (See Figure 4.18).

Fig 4.18: user interface

4.7 Experimental Evaluation

For the training of my model, I have chosen a subset of the GTSRB. This subset consists of 39,209 images. The whole set of images could not have been used because not all images were labeled. A plot of the dataset as traffic sign frequency by Class ID can be observed in Figure 4.5. Each image was cropped according to its own ROI indicators and then resized to a size common to all images. The resizing of the images was done using cv2.resize from the cv2 library. In deep learning, resizing images is an important preprocessing step because models train faster on smaller inputs. The dataset was split in a 60 - 20 - 20 ratio. 60% of the images were used for the training of the model, 20% for testing, and 20% for validation. Before being split, the dataset is randomized because images are loaded in the program in blocks formed by class id. The configuration of the data after the split is saved in .npy files for use in other sections of the program.

While building the model I have tried two types of structures. The first was using three pairs of convolutional - max-pooling layers before the fully connected layers. The second was stacking two convolutional layers prior to each max-pooling layer. This allowed for the model to pick up more intricate features of the input images, thus resulting in better accuracy than the one obtained with the first approach.

The activation function used for each layer in the model is ReLU (except for the output layer). This activation function is often preferred in neural network architecture

because the models that use it are trained more easily and tend to achieve an increased accuracy [25]. The hyperparameter padding was set to ‘same’ in the convolutional layers, which means that all images in the input are padded with zeros evenly, in order for the output to be of the same dimensions as the input. This was done to make sure that all areas in the input are covered by the pool size - stride pair. The pool size and the stride have been both set to (2, 2) inside of the max-pooling layers. The dropout rate selected for the fully connected layers was 0.3, whereas, for the other ones, it was 0.2. This decision was taken by trial and error processes. Although the most recommended configuration for CNNs used 0.5 for the dropout rate in the fully connected layers, in practice, using the 0.3 rate yielded better accuracy.

The loss function of choice for the model is categorical cross-entropy because it is regarded as the default error function for multi-class classification. “Cross-entropy will calculate a score that summarizes the average difference between the actual and predicted probability distributions for all classes in the problem” [25]. The most desirable cross-entropy value is 0. In order for the ‘categorical_crossentropy’ to be used as the loss function. A requirement for this function is that the final layer of the CNN is configured with a neuron for every class id, and uses the softmax activation function. In order for this to work, the target attributes must be one-hot encoded. One-hot encoding is a method of quantifying data. This method outputs a vector of size equal to the number of classes, and for every input, the vector is filled with zeros (‘0’), except for the position of the class associated with the input, to which ‘1’ is attributed [26]. The optimizer of choice for this CNN model is ‘adam’. “The Adaptive Momentum (Adam) technique estimates the adaptive learning rate for all parameters involved in the training of gradients” [27]. Adam is frequently used in machine learning when large sets of data are involved. It minimizes computational cost and necessitates less memory for implementation [27]. The learning rate for the optimizer is set to ‘1e-4’, which is a scientific notation for 0.0001.

The number of epochs chosen for training the model is 7. This was also determined by a trial and error process. When the model was trained for more than 7 epochs, it began overfitting. “Overfitting refers to a model that models the training data too well” [25]. This process occurs “ when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data” [25].

A confusion matrix is a method of measuring the accuracy of a model. On one axis it has the predicted target attributes, and on the other the correct target attributes. Its design makes it a great tool for observing a model's performance, assessing the number of true positives, false positives, false negatives and true negatives. The confusion matrix of the model can be observed in Figure 4.19.

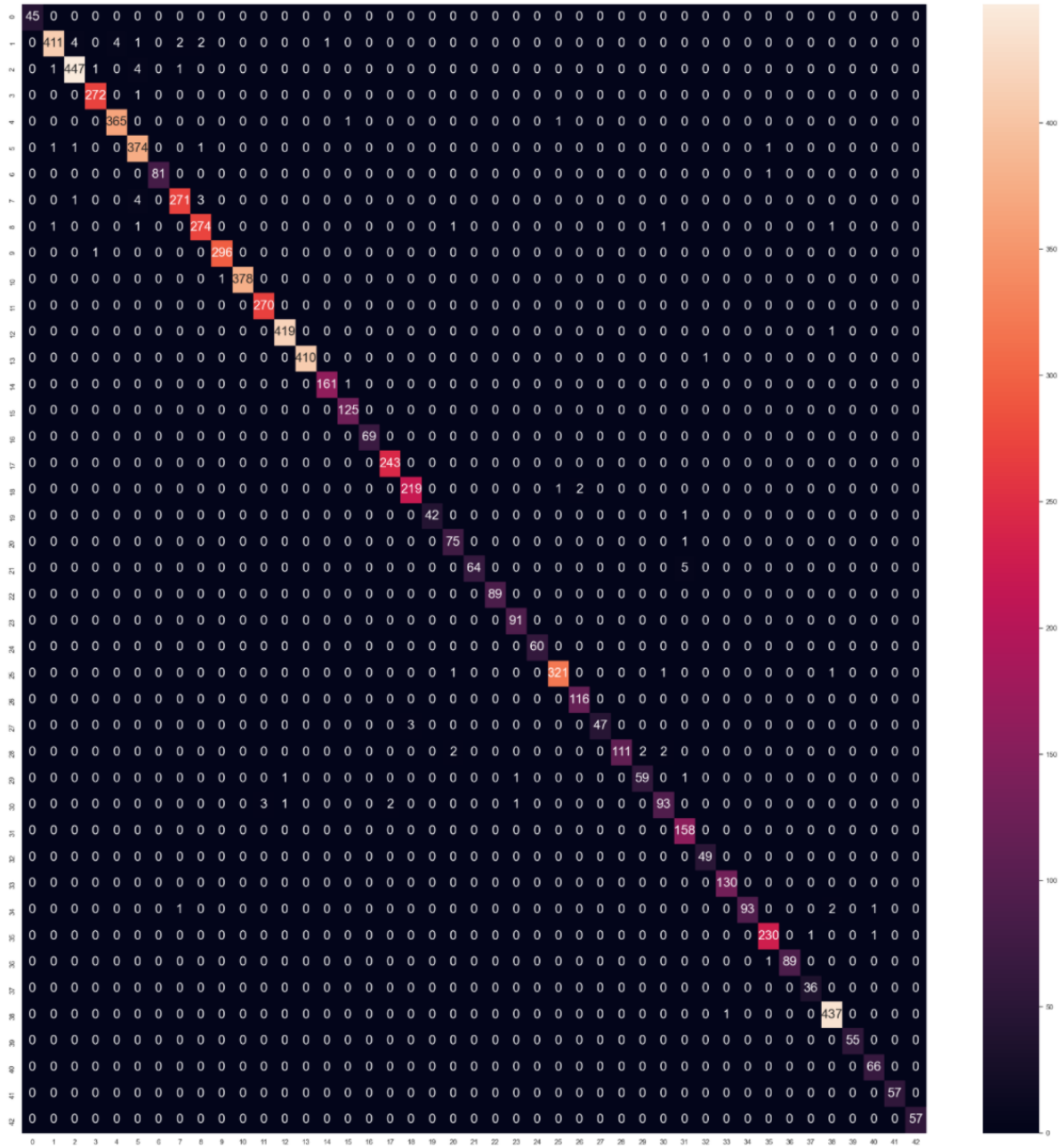

Fig 4.19: The confusion matrix.

Accuracy:

- Precision score = 0.9883023857450784
- Recall score = 0.989627280323195
- F1 score = 0.9888574017784761

5. Conclusions

Traffic sign classification is a feature vital to any fully autonomous vehicle. Many lives depend on the ability of a car to drive itself, and this ability is heavily impacted by a car's ability to correctly recognize traffic signs.

In this diploma thesis, I have attempted to develop a convolutional neural network with the purpose of classifying traffic sign images. The model was developed in python, by employing the help of the 'Sequential' model from the Keras library. The model was trained on the 'German Traffic Sign Recognition Benchmark' dataset and managed to achieve an accuracy of 98.8% on the test data.

Future work should investigate the impact that color has on the model's accuracy. Different approaches, using raw color, normalized color channels, or grayscale images should be tested.

This model represents only a part of the software mechanism of traffic sign recognition that a fully autonomous car should be equipped with. Another model should extract the image containing the traffic sign from a live video feed, and that image should be used as the input for getting a prediction from the model developed for this thesis. The class id outputted by the model should then be used by the driving decision system of the car in order for the best driving decision to be taken (e.g.: stop at a 'stop' sign, or disable overtaking between the 'overtaking prohibited' and 'overtaking not prohibited anymore' signs).

References

[1] McCarthy, J. (2004). What is artificial intelligence? *Artificial Intelligence,* 1-14.

[2] O'Shea, K., & Nash, R. (2015). An Introduction To Convolutional Neural Networks. arXiv, 1-11.

[3] https://en.wikipedia.org/wiki/Machine_learning - as of 30.05.2021

[4] Jordan, M., & Mitchell, T. (2015). Machine learning: Trends, perspectives, and prospects. *Science, 349,* 255-260

[5] Hassoun, M., (1996). Transactions on information theory. *IEEE*, *42,* 1322-1324

[6] Graupe, D., (2013). Principles of Artificial Neural Networks 3rd edition. *Advanced Series in Circuits and Systems*, *7,* 1-13

[7] https://towardsdatascience.com/convolutional-neural-network-17fb77e76c05 - as of 05.06.2021

[8] https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/ - as of 05.06.2021

[9] https://www.researchgate.net/figure/Example-of-fully-connected-neural-network_fig2_331525817 as of 05.06.2021

[10] Bishop, C., (2006). Pattern Recognition and Machine Learning *Springer.*

[11] Stallkamp, J., Schlipsing, M., Salmen, J., & Igel, C., (2012). Man vs. Computer: Benchmarking Machine Learning Algorithms for Traffic Sign Recognition. *Neural Networks, 32,* 323-332

[12] Ketkar, N., (2017). Introduction to Keras. *Deep Learning with Python.*

[13] The Keras API manual - the Conv2D layer. https://keras.io/api/layers/convolution_layers/convolution2d/ - as of 08.06.2021

[14] The Keras API manual - the BatchNormalization layer. https://keras.io/api/layers/normalization_layers/batch_normalization/ - as of 08.06.2021

[15] The Keras API manual - the MaxPooling2D layer. https://keras.io/api/layers/pooling_layers/max_pooling2d/ - as of 08.06.2021

[16] The Keras API manual - the Dropout layer. https://keras.io/api/layers/regularization_layers/dropout/ - as of 08.06.2021

[17] The Keras API manual - the Flatten layer. https://keras.io/api/layers/reshaping_layers/flatten/ - as of 08.06.2021

[18] The Keras API manual - the Dense layer. https://keras.io/api/layers/core_layers/dense/ - as of 08.06.2021

[19] Brownlee, J., (2020), Deep Learning for Computer Vision: Image Classification, Object Detection and Face Recognition in Python. 1-544

[20] Budhiraja, A., (2016). Dropout in (Deep) Machine Learning. https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5 - as of 10.06.2020

[21] Gillis, A., & Silverthorne, V., Integrated development environment (IDE). https://searchsoftwarequality.techtarget.com/definition/integrated-development-environment - as of 12.06.2021

[22] https://en.wikipedia.org/wiki/Integrated_development_environment - as of 12.06.2021

[23] Musciano, C., & Kennedy, B. (2002). HTML & XHTML: The definitive Guide. Fifth edition.

[24] Wilton, P., (2004). Beginning JavaScript. Second edition.

[25] Brownlee, J., (2020) Better Deep Learning: Train Faster, Reduce Overfitting, and Make Better Predictions.

[26] One Hot Encoding https://deepai.org/machine-learning-glossary-and-terms/one-hot-encoding - as of 14.06.2021

[27] Yaqub, M., Feng, J, & Mehmood, A. (2020). State-of-the-Art CNN Optimizer for Brain Tumor Segmentation in Magnetic Resonance Images.

[28] The Turing test https://en.wikipedia.org/wiki/Turing_test - as of 14.06.2021

[29] The history of self-driving cars https://en.wikipedia.org/wiki/History_of_self-driving_cars - as of 14.06.2021

[30] Sermanet, P, & LeCun, Y. (2011). Traffic sign recognition with multi-scale Convolutional Networks. The 2011 International Joint Conference on Neural Networks, 2809-2813.

[31] Yann LeCun https://en.wikipedia.org/wiki/Yann_LeCun - as of 15.06.2021

[32] Zhang, J., Wang, W., Lu, C., Wang, J. & Sangaiah, A.K. (2020). Lightweight deep network for traffic sign classification. *Annales Des Télécommunications*, 75(7-8), 369-379.

[33] Wood, T. What is the Softmax Function?. https://deepai.org/machine-learning-glossary-and-terms/softmax-layer - as of 15.06.2021

[34] Machine Learning & Deep Learning Fundamentals https://deeplizard.com/learn/video/FK77zZxaBoI - as of 15.06.2021

[35] Activation functions in Neural Networks https://www.geeksforgeeks.org/activation-functions-neural-networks/ - as of 15.06.2021

[36] Wood, T. What is Backpropagation? https://deepai.org/machine-learning-glossary-and-terms/backpropagation - as of 15.06.2021

[37] Donges, N. (2019). Gradient Descent: an introduction to 1 of machine learning's most popular algorithms. https://builtin.com/data-science/gradient-descent - as of 15.06.2021