Traffic Sign Recognition

Ioannis Ioannidis - Dimitris Patiniotis Spyropoulos

University of Piraeus
NCSR Demokritos

Keywords— Deep Learning, CNN, Multiclass Classification, Traffic Sign Recognition

1. Introduction

Road safety is attracting the attention of many researchers around the world, since it is indispensable in protecting human life. For several years now, systems for the detection, classification, and recognition of road signs have become a very important research topic. The ultimate goal is the construction of a model with the absolute performance, in order to be deployed to driver assistance systems and providing full safety.

In the present report, we describe how we tried to create such a model, and a lighter version of it using Deep Learning by tuning some of its parameters. The structure is as following: Firstly, in the *Data Description* we provide some information about the data we used for the training, validating and testing procedures while in the *Theoretical Background* section we give an intuitive taste of the algorithms we will be comparing. Experimental set-up and the final results are presented in the *Experiments* part. Closing, all the remarks highlighted during the task and the observations we ended up with are quoted in the *Conclusion*.

2. Data Description

The data for this project are collected from "German Traffic Sign Recognition Benchmark", a multi-category classification competition held at IJCNN 2011. It is about a comprehensive, lifelike dataset of more than 40,000 traffic sign images. It reflects the strong variations in visual appearance of signs due to distance, illumination, weather conditions, partial occlusions, and rotations.

Dataset is pre-splitted in train and test sets, including 30.209 and 12.630 images respectively. It comprises 43 classes with unbalanced class frequencies, which is quite logical since some signs like "Keep speed below 30 K-mph" or "A bump ahead" appears more often then signs like "Road under construction ahead".

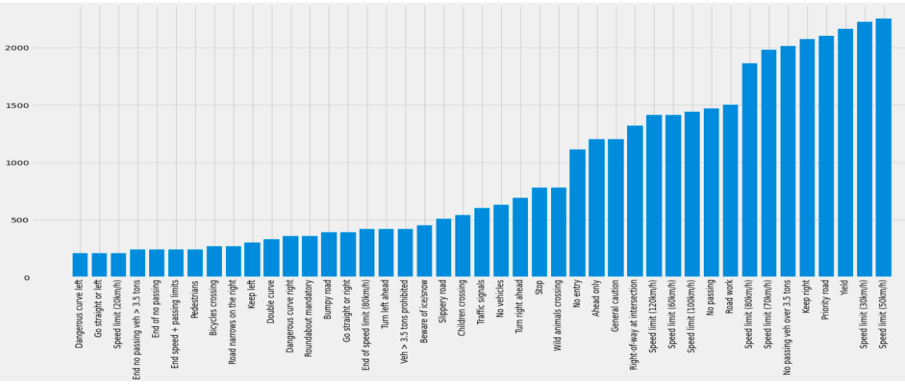


Fig. 1: Number of samples in each class of GTSRB dataset

The *train.csv* and *test.csv* files contain the below information about each image:

- **Width** : width of image in number of pixels
- **Height** : height of image in number of pixels
- **ClassId** : class label of the image. It is an Integer between 0 and 43
- **Path** : path where the image is present in the folder

We need to mention that before passing images through models for training we applied a 30×30 resize.

3. Background

3.1. Related Work

Before the widespread adoption of Convolutional Neural Networks (CNNs), various object detection methods were adapted for traffic-sign classification, e.g. based on SVMs and sparse representations. Recently, convolutional neural network approaches have been shown to outperform such simple classifiers when tested on the GTSRB benchmark. These approaches include using a committee of CNNs, multi-scale CNNs and CNNs with a hinge loss function, the latter achieving a precision rate of 99.65%, better than human performance. However, as noted earlier, these approaches perform classification on already detected signs, which is impractical in real applications.

3.2. Theoretical Background

3.2.1. Support Vector Machines

Support vector machine (SVM) is a method for the classification of both linear and nonlinear data. If the data is linearly separable, the SVM searches for the linear optimal separating hyperplane (the linear kernel), which is a decision boundary that separates data of one class from another. Mathematically, a separating hyperplane can be written as $W \cdot X + b = 0$, where W is a weight vector and $W = w_1, w_2, ..., w_n$, X is a training tuple and b is a scalar. In order to optimize the hyperplane, the problem essentially transforms to the minimization of $|W|$, which is eventually computed as $\sum_{i=1}^{n} a_i y_i x_i$, where a_i are numeric parameters and y_i are labels based on support vectors X_i. That is: if $y_i = 1$, then $\sum_{i=1}^{n} w_i x_i >= 1$; if$y_i = -1$, then $\sum_{i=1}^{n} w_i x_i >= -1$.

If the data is linearly inseparable, the SVM uses nonlinear mapping to transform the data into a higher dimension. It then solve the problem by finding a linear hyperplane. Functions to perform such transformations are called kernel functions. The kernel function selected for this example is the Gaussian Radial Basis Function (RBF): $K(X_i, X_j) = e^{-\gamma ||X_i - X_j||^2/2}$, where X_i are support vectors, X_j are testing tuples, and γ is a free parameter. Figure 2 shows a classification example of SVM based on the linear kernel and the RBF kernel.

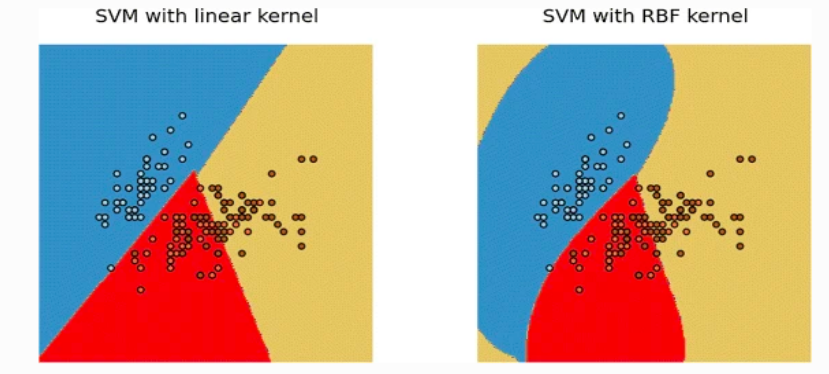


Fig. 2: Classification Example of SVM

3.2.2. Convolutional Neural Networks

Classification with artificial neural networks is a very popular approach to solve pattern recognition problems. A neural network is a mathematical model based on connected via each other neural units – artificial neurons – similarly to biological neural networks. Typically, neurons are organized in layers, and the connections are established between neurons from only adjacent layers. The input low-level feature vector is put into first layer and, moving from layer to layer, is transformed to the high-level features vector. The output layer neurons amount is equal to the number of classifying classes. Thus, the output vector is the vector of probabilities showing the possibility that the input vector belongs to a corresponding class.

An artificial neuron implements the weighted adder, which output is described as follows:

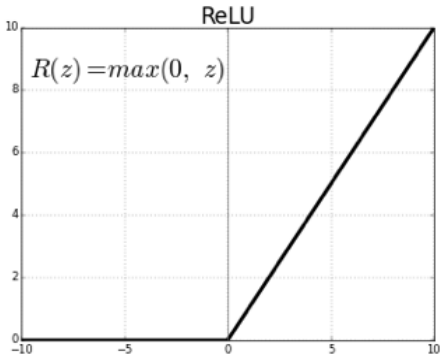
$$a_i^j = \sigma(\sum_k a_k^{i-1} w^j j_k)$$

where a_j^i is the j^{th} neuron in the i^{th} layer, w_k^{ij} stands for weight of a synapse, which connects the j^{th} neuron in the i^{th} layer with the k^{th} neuron in the layer $i-1$. Widely used in regression, the logistic function is applied as an activation function. It is worth noting that the single artificial neuron performs the logistic regression function.

The training process is to minimize the cost function with minimization methods based on the gradient decent also known as backpropagation. In classification problems, the most commonly used cost function is the cross entropy:

$$H(p,q) = -\sum_i Y(i) logy(i)$$

Training networks with large number of layers, also called deep networks, with sigmoid activation is difficult due to vanishing gradient problem. To overcome this problem, the Rectified Linear Unit (ReLU) function is used as an activation function:



Today, classifying with convolutional neural networks is the state of the art pattern recognition method in computer vision. Unlike traditional neural networks, which works with one-dimensional feature vectors, a convolutional neural network takes a two-dimensional image and consequentially processes it with convolutional layers.

A CNN is composed of input and output layers and multiple hidden layers, which can be divided into a convolution layer, a pooling layer, a rectified linear unit layer, and a fully connected layer. The architecture of a CNN is shown in Fig. 2.

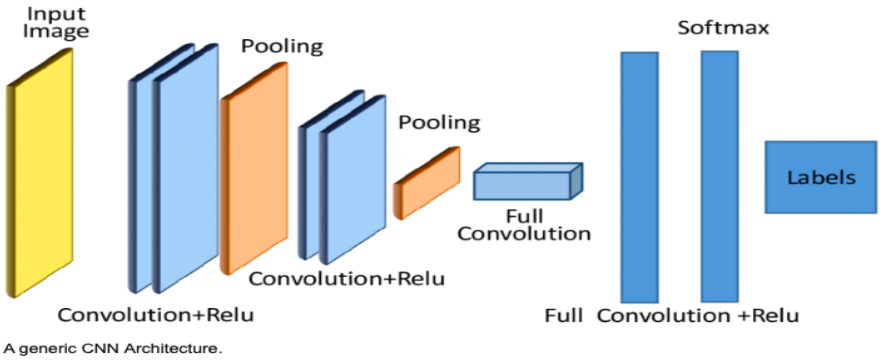


Fig. 3: Convolutional Neural Network Architecture

Each convolutional layer consists of a set of trainable filters and computes dot productions between these filters and layer input to obtain an activation map. These filters are also known as kernels and allow detecting the same features in different locations. For example, Fig. 3 shows the result of applying convolution to an image with 4 kernels.

Compared with a traditional neural network, a CNN has three major characteristics, namely local perception, weight sharing, and a multi-convolution kernel. Based on these characteristics, a CNN has several advantages, including the following:

1. the architecture can well capture the spatial topology of input
2. weight sharing reduces the number of trainable network parameters, thus reducing the network complexity and improving robustness
3. the shared convolution kernel allows no pressure to perform high-dimensional data processing
4. the hardware implementation of a CNN is much simpler than a fully connected neural network of the same input size

Fig. 4: Image Convolution

4. Experiments

4.1. Support Vector Machine as a baseline

Before we begin tuning our model, we first trained an SVM to set a baseline performance. As we mentioned in the related work section, SVMs was a popular choice in the pre-neural network era, and thus was our pick as a representative of the traditional machine learning algorithms. In that way, we achieved an accuracy of 81%.

4.2. Deep Learning Model

4.2.1. Model Architecture

For the Deep Learning part we will be using a CNN network. It consists of four convolutional layers, including batch normalization and max pooling, followed by two fully connected layers. The analytic architecture is shown in Figure 4.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 16)	448
conv2d_1 (Conv2D)	(None, 26, 26, 32)	4640
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
batch_normalization (BatchNormalization)	(None, 13, 13, 32)	128
conv2d_2 (Conv2D)	(None, 11, 11, 64)	18496
conv2d_3 (Conv2D)	(None, 9, 9, 128)	73856
max_pooling2d_1 (MaxPooling2D)	(None, 4, 4, 128)	0
batch_normalization_1 (BatchNormalization)	(None, 4, 4, 128)	512
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 512)	1049088
batch_normalization_2 (BatchNormalization)	(None, 512)	2048
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 43)	22059

Total params: 1,171,275
Trainable params: 1,169,931
Non-trainable params: 1,344

Fig. 5: Initial Model Architecture

4.2.2. Optimizers Comparison

Our first experiment was to compare the performance of three different optimizers: Stohastic Gradient Decent (SGD), Adam and AdaDelta. Beginning with SGD, it the most basic form of gradient decent. SGD subtracts the gradient multiplied by the learning rate from the weights ($\theta_i = \theta_i - \alpha \frac{\partial L}{\partial \theta_i}$). Despite its simplicity, SGD has strong theoretical foundations and is frequently used in training edge Neural Networks.

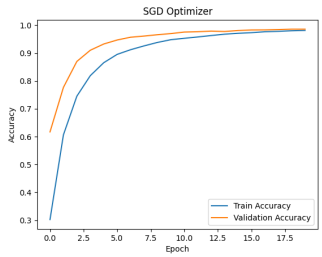
On the other hand, Adam uses the squared gradients to scale the learning rate and it takes advantage of momentum by using moving average of the gradient instead of gradient itself like SGD with momentum. It is an

adaptive learning rate method, which means, it computes individual learning rates for different parameters. Its name is derived from adaptive moment estimation, and the reason it's called that is because Adam uses estimations of first and second moments of gradient to adapt the learning rate for each weight of the neural network. The first moment is mean, and the second moment is uncentered variance (meaning we don't subtract the mean during variance calculation). To estimate the moments, Adam utilizes exponentially moving averages, computed on the gradient evaluated on a current mini-batch.

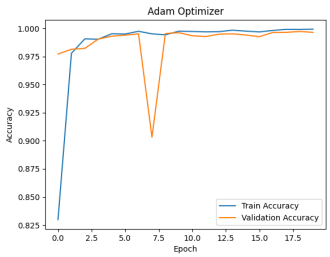
Finally, AdaDelta optimization is a stochastic gradient descent method that is based on adaptive learning rate per dimension to address two drawbacks:

- The continual decay of learning rates throughout training.
- The need for a manually selected global learning rate.

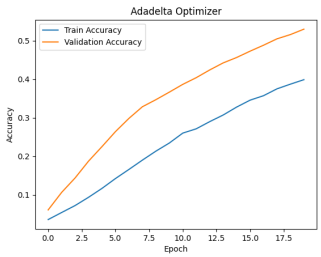
Specifically, it adapts learning rates based on a moving window of gradient updates, instead of accumulating all past gradients. This way, Adadelta continues learning even when many updates have been done. It is important to mention that using AdaDelta we don't have to set an initial learning rate.



(a) SGD



(b) Adam

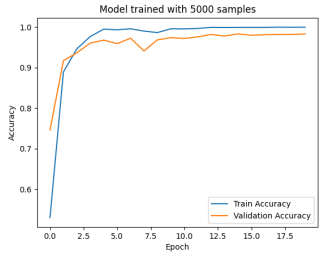


(c) AdaDelta

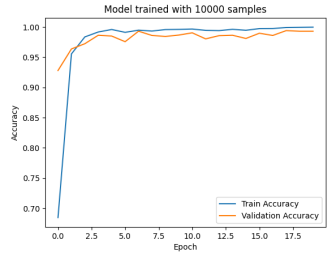
Running the experiments for each optimizer what we observed was that despite the gradual increase of SGD, Adam seems to perform better and faster, while AdaDelta is way too slow.

4.2.3. Training Size Modification

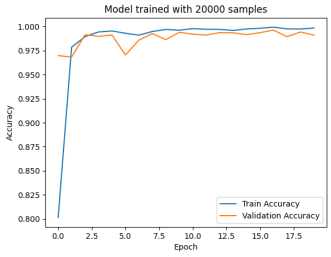
Next we were interesting in examining the the way the accuracy and training time is being affected by the sample size. For that purpose we ran the network's training with 5000, 10000 and 20000 images.



(a) 5000 Images
5s/epoch



(b) 1000 Images
9s/epoch



(c) 20000
20s/epoch

Here it is obvious that while accuracy seems not to be that much affected, training time is significantly reduced when we decrease the train sample.

4.2.4. Reducing Convolution Layers

Finally, we wanted to examine if it would be possible to create a lighter model, without affecting the performance levels. By lighter we mean a model with less computational cost. In order to achieve this, we decreased the number of parameters by almost ten times by applying two convolutions instead of four (Fig 7).

In that way, we achieve quite similar results (Fig 7). However training a lighter model saves a lot of time and computational sources, so we could end up that is a quite sensible trade-off.

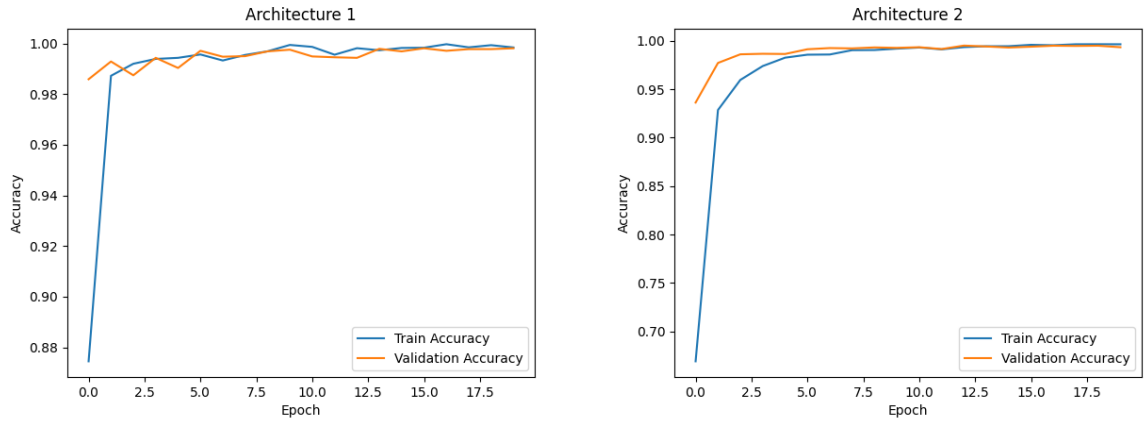
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 16)	448
conv2d_1 (Conv2D)	(None, 26, 26, 32)	4640
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
batch_normalization (BatchNormalization)	(None, 13, 13, 32)	128
conv2d_2 (Conv2D)	(None, 11, 11, 64)	18496
conv2d_3 (Conv2D)	(None, 9, 9, 128)	73856
max_pooling2d_1 (MaxPooling2D)	(None, 4, 4, 128)	0
batch_normalization_1 (BatchNormalization)	(None, 4, 4, 128)	512
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 512)	1049088
batch_normalization_2 (BatchNormalization)	(None, 512)	2048
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 43)	22059

Total params: 1,171,275
Trainable params: 1,169,931
Non-trainable params: 1,344

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 28, 28, 16)	448
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 16)	0
batch_normalization_3 (BatchNormalization)	(None, 14, 14, 16)	64
conv2d_5 (Conv2D)	(None, 12, 12, 32)	4640
max_pooling2d_3 (MaxPooling2D)	(None, 6, 6, 32)	0
batch_normalization_4 (BatchNormalization)	(None, 6, 6, 32)	128
flatten_1 (Flatten)	(None, 1152)	0
dense_2 (Dense)	(None, 128)	147584
batch_normalization_5 (BatchNormalization)	(None, 128)	512
dropout_1 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 43)	5547

Total params: 158,923
Trainable params: 158,571
Non-trainable params: 352

Fig. 8: Model's Summaries



(a) Baseline CNN

(b) Reduced Architecture

Fig. 9: Models Performances

5. Conclusion

In this project we tried to perform some experiments on our way to build lighter models that achieve the same performance of the GTSRB dataset. Our first approach was to examine which optimizer acts faster, in order to save some epochs and time during the training procedure. We compared Stohastic Gradient Decent, Adam and AdaDelta and Adam clearly converged way faster and achieving better performance than the others.

Our second experiment was to try different numbers of training samples. Specifically, we trained a model with 5000, 10000 and 20000 images and measured accuracy and time. What we ended up with was that accuracy wasn't that much affected, while train time almost doubled at each step. Finally, we decreased the convolutions from four to two in order to minimize the parameters. We observed that accuracy remained at the same high levels (0.5 % down).

Concluding, examining the right parameters it is possible to build lighter models, which require less training time and computational cost. As a further work, we could automatize those -and maybe more -experiments, in order to be easily applied on any task. In that way we could optimize the cost for training evaluating and testing the problem.