Lab Report
Introduction to Deep Learning for Computer Vision

undisclosed authors

February 21, 2018

Abstract

This report sums up our results of the practical in "Introduction to Deep Learning for Computer Vision" at the "Institut für Neuroinformatik" at Ruhr-University Bochum. The course covered a short review of basic image processing methods as well as manual image classification and feature extraction as was common pre 2012 and finally introduced the participants to the construction of deep convolutional neural networks [Krizhevsky et al., 2012] via the Tensorflow [Abadi et al., 2015] library for python3.6 as it is used in state of the art machine learning.

1 Basics of Image Classification

During the first day of the practical all experiments were conducted on grayscale images. The solutions to all tasks of this day can be found in the file 'day1.py'.

As demanded by the given set of tasks, we implemented a function to convolve an image with a filter for arbitrary filter and image sizes. We chose to handle convoluted pixels outside the image borders as if they were of the same color as the nearest image pixel according to L1 norm. Down the road - for reasons of execution time - we used the *filter2D* function provided by the opencv-python library [Bradski, 2000].

util.py

```
def convolve(image, kernel):
    convolutedImage = np.zeros(image.shape)
    for yimage in range(0, image.shape[1]):
        for ximage in range(0, image.shape[0]):
            sum = 0
            for ykernel in range(0, kernel.shape[1]):
                for xkernel in range(0, kernel.shape[0]):
                    yconv = int(np.min( [image.shape[1]-1, np.max([0, yimage - kernel.
    shape[1]/2 + ykernel])]))

                    xconv = int(np.min( [image.shape[0]-1, np.max([0, ximage - kernel.
    shape[0]/2 + xkernel])]))

                    sum += image[xconv, yconv] * kernel[xkernel, ykernel]

            convolutedImage[ximage, yimage] = sum

    return convolutedImage
```

Figure 1: The implemented convolution function

1.1 Image Normalization

We manually selected and extracted a region of interest(ROI) from two instances of easily distinguishable images of speed limitation signs (30 kph and 50 kph) and used those ROIs as convolution filters on two sets of traffic signs containing 30-kph or 50-kph speed limit signs respectively. As we strove to implement a template matching, we started ~~with~~ demeaning the ROI and the image.

Afterwards we made sure both had values with a variance of one, such that the values were normal distributed. This does now allow for better template matching, when the ROI is convoluted with the actual image, as features like global differences in brightness have been eliminated.

In order to map the result of the convolution to $[-1, 1]$ on the one hand and maintain reversibility of the operation on the other, we demeaned and normalized both the ROIs and images by means of normal distribution of all values.[2]

util.py

```
def normalizeImage(img):
    # convert to float point values
    toRet = img.astype(np.float32)
    # calculate the standard deviation
    sigma = np.sqrt(((toRet-toRet.mean())**2).mean())
    # and apply the actual normalization by demeaning and setting the standard deviation
     to 1
    toRet = (toRet-toRet.mean())/sigma

    return toRet
```

Figure 2: The implemented function for normalization

1.2 Classification via Perceptron

Finally we plotted the 2-dim feature vector of each image consisting of the maximum response to the convolution with each of the two ROIs and manually defined a separating hyperplane by which to discern between the two classes. We achieved a failure rate of 34% by means of trial and error while using the classifier. The sets are not linear separable with the chosen feature, as seen in figure [3], which explains the bad classification results, as we try to fit a linear separation.

$$w^T x > b \text{ with } w = \begin{pmatrix} 0.9 \\ -0.9 \end{pmatrix} \text{ and } b = -2$$

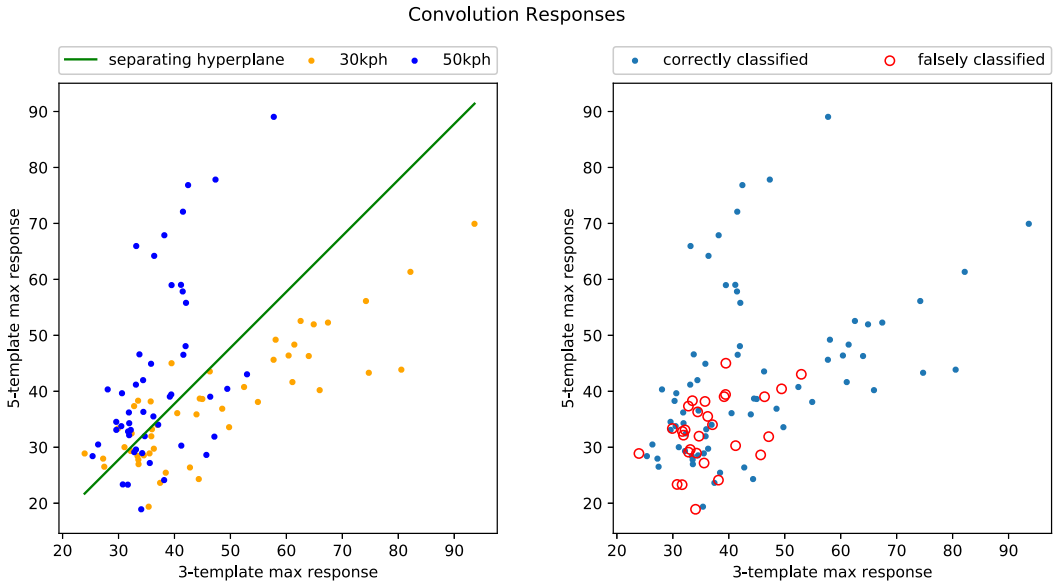

Figure 3: **Left:** Maximum convolution responses for each image and ROI separated by color depending on true labels. **Right:** The same data points separated by color depending on classification success

2 Feature-based Image Classification

Goal of the second day's work was to improve upon the classificator explained above by constructing higher dimensional feature vectors, analyzing the results' principal components and automatizing

the search for good separating hyperplanes by utilization of a support vector machine. All results are condensed in the file 'day2.py'.

2.1 HOG-Feature Extraction

After constructing relatively low-dimensional feature vectors on the first day, we advanced to constructing rather high-dimensional Histogram of Oriented Gradients-Features to improve on classification quality.

We basically partitioned the image into a regular grid of cells, several of which formed blocks and for each cell we computed a histogram of contained gradients consisting of a predetermined number of buckets which in turn were representing orientations. We thereby received a feature vector for each image of dimensionality:

$$\#BlocksPerRow \times \#BlocksPerColumn \times \#CellsPerBlock \times \#Orientations$$

For details consult the original paper [Dalal and Triggs, 2005].

2.2 Principal Component Analysis

Next up was the task to gain additional information about the data on hand by projecting the high dimensional dataset onto a 2-dimensional subspace spanned by it's principal components. We utilized the principal component analysis (PCA)-Class provided by the scikit-learn library to achieve this goal [Pedregosa et al., 2011]. The results for samples from three of the classes can be seen in [4]

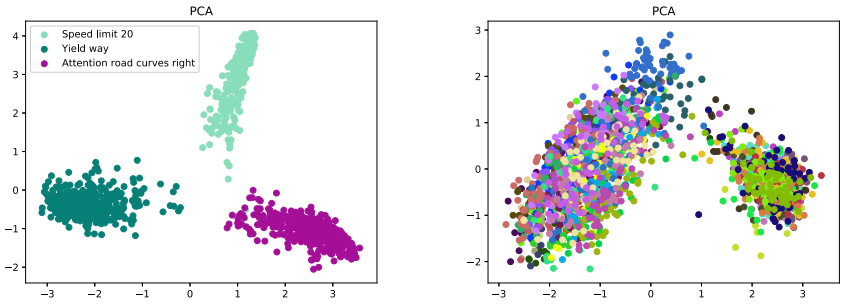

Figure 4: **Left:** The PCA successfully arranged the projected datapoints from three classes (Speed Limit 20, Yield Way, Attention Road Curves Right) in somewhat well separable clusters. **Right:** We can see that the PCA failed produce separable clusters for all classes simultaneously, as the result is a single multicolored cloud, where each color represents a class.

2.3 Support Vector Machines

We used scikit-learn's SVC-Class to realize a support vector machine (SVM) that successfully differentiates between the selected classes of traffic signs. It was initialized to use the "one-vs-one" decision function, such that it tries to separate pairs of labeled point sets, by trying to fit a hyperplane in between which maximizes the margin between these point sets. For details on how support vector machines work turn to the original publication [Cortes and Vapnik, 1995].

We used the HOG feature vector as an input for the SVM and achieved an accuracy of 99.63% and an error rate of 0.37% respectively, for the image classes as chosen in the PCA (Speed Limit 20, Yield Way, Attention Road Curves Right). When using all 43 classes the accuracy dropped to 40.66% with an error rate of 59.34%.

This drop in accuracy can be explained by the weakness of HOG features in street sign recognition. The three classes in our first test were chosen to be as dissimilar as possible, therefore the good results. As many street signs share similar shape features, the SVM simply fails trying to separate the shape features as for the reason for the shape similarity, which is again inseparable. We can deduce this already from the PCA over all classes, as these show massive overlaps which we can see in [4].

Another tool which supports this interpretation of the observed failure is the confusion matrix. This matrix quantifies how which predictions were misclassified. To make an example, HOG features in street signs do not differ greatly, so the SVM misclassifies the speed signs. This can be seen in the confusion matrix [5], as the upper left block matrix (0-6 to 0-6) represents the confusion within the class of speed signs, because we see many mispredictions in this area.

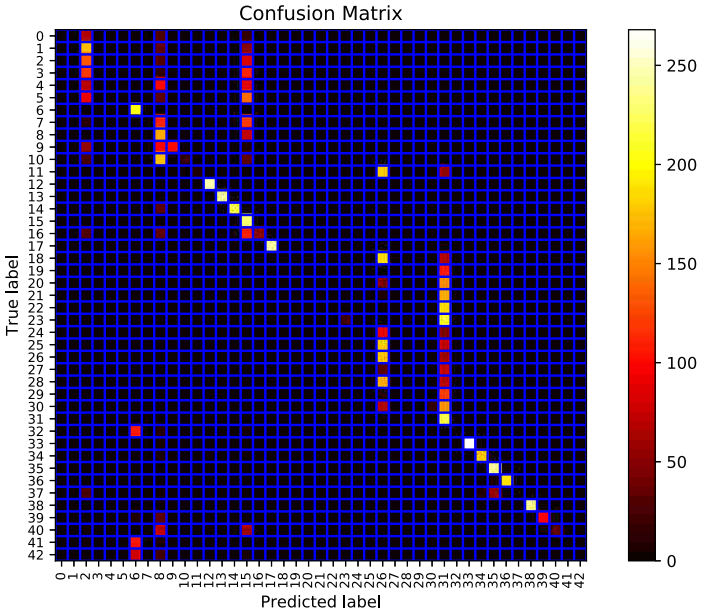

Figure 5: The confusion matrix of all 43 classes from the SVM with HOG feature input.

3 Deep Neural Networks

On the third day we started with the implementation of state of the art methods in image recognition, namely Convolutional Neural Networks (CNN), which had a revolution with AlexNet [Krizhevsky et al., 2012]. We spent the rest of our remaining days in optimizing and introspecting the network internals, to achieve superhuman performance in traffic sign recognition. At the end of the practical we had a network with over 99% accuracy in correctly identifying traffic signs from 43 classes in color images.

3.1 Training Process

For the training of the networks described in the following network we partitioned the complete set of images into three fixed-size, but ~~by~~ label balanced subsets: A training set containing 70% of the images, a testing set with 20% and a validation set with 10%. The training process could be further improved by implementing a non-fixed cross-validation approach like leave-one-out cross-validation [Kohavi et al., 1995]. To speed up the learning process we initialized the variables with normal distributed values, which are clinched to a maximal value of 0.01. For the optimization we have decided to take a state of the art stochastic optimizer called ADAM [Kingma and Ba, 2014].

3.2 Initial Feed-Forward Network

We started off with a feed forward network to classify the image, as demanded by the third day's tasks. We decided to use a standard fully connected network architecture with the RGB image as its input, two hidden layers with 1024 and 128 hidden and bias units in between and 43 output units, each one corresponding to one category from the GTSRB [Houben et al., 2013]. Finally we added a softmax to make the output probabilistic. This architecture is able to achieve about 83% accuracy on the test set. An analysis of the prediction performance can be seen in the figure below. [6]

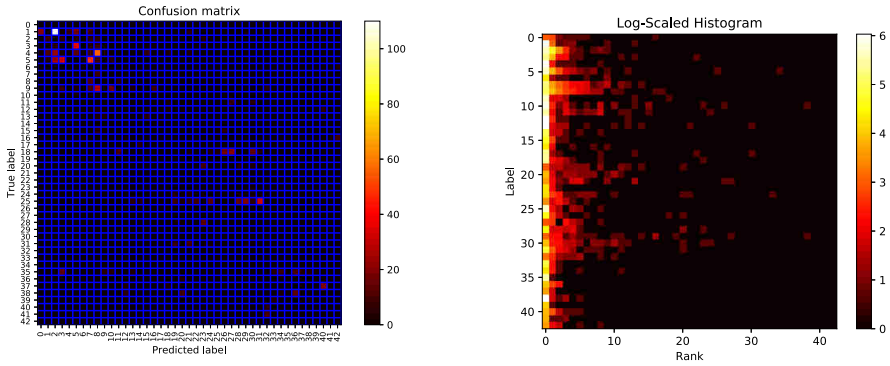

Figure 6: An overview on the overall performance of the feed forward network on the GTSRB.

3.3 Convoltional Neural Network Architecture

Our final CNN architecture consists of three convolution layers, which project into the previously described fully connected feed-forward network. An overview of the network is presented below.[11] Initially we started with a downscaled version of AlexNet, consisting of two convolution layers and two fully connected layers and started to train it on three classes of traffic signs, as selected in day two. At this point a convolution layer consisted of a set of convolutions, which projected into rectified linear units (ReLUs), which in turn projected into maximum pooling units.[10]

As we strove for the goal to maximize the recognition accuracy of our network, we gradually introduced new layers, until we achieved a good trade-off between learning time and accuracy. Then we started fine tuning architectural parameters of our network, to see where our performance ends up. Luckily we already have hit a sweet spot with this trade-off, as we ended on about 95% recognition accuracy, when switching to the full dataset with all 43 classes.

At this point we started integrating tools to communicate with tensorboard, tensorflow's visualization tool, into our program to extract further information about the training process as well as the neural network itself. With this tool we were able to improve our productivity and tune the architecture of the network even further. In a first step we added biases to all layers as well as dropout to the fully connected layers, which resulted in minor improvements for the recognition accuracy. After that we changed the parameters of the convolution layers layer by layer. We iteratively modified the number of convolution masks as well as the size of the pooling masks, until hitting about 99% accuracy at convergence [7], at which point we decided that the prediction was good enough, as the network outperformed human performance, which is around 98.5%.

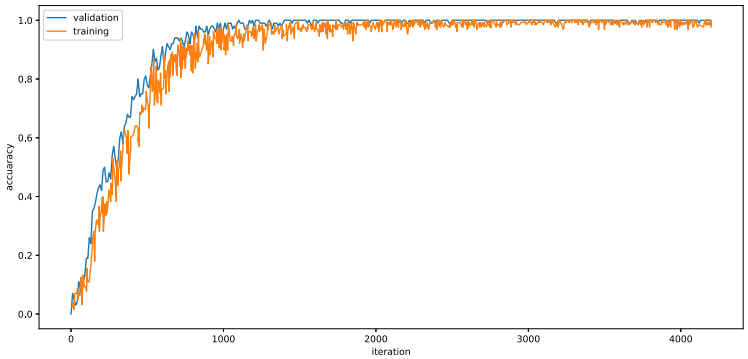

Figure 7: The control trajectories of training and validation accuracies over 20 epochs. At this point we can assume that the learning process converged, as no trend to change closer to 1.0 over the last 500 batches is observable.

Finally we spent our remaining time in introspecting the internals of our network further, trying to find where possible points of failure may reside. We observed that the network was commonly

very confident in its results when failing to predict the correct label. After sighting some masks from the first two convolution layers we assumed that the confidence in wrong predictions could be a form of overfitting in the first layers, since high confidence in wrong results is an indicator for some kind of overfitting occurring.

To verify this assumption we simply added dropout units to the convolution masks of all convolution layers and retrained the network from scratch. With the introduced dropouts we observed an enormous drop in its prediction confidence, when the network failed classifying an input correctly in the test set. While this result is an interesting observation, it yielded just a minor improvement in prediction accuracy. A short analysis of the final output is presented below. [8]

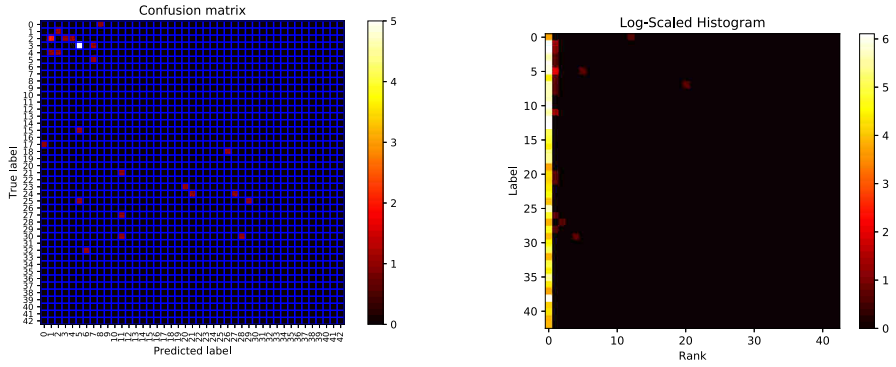

Figure 8: An overview on the overall performance of the final network architecture on the GTSRB.

3.4 Error Analysis

As we still got several error, we looked trough all failure cases of the neural network. We could partition these failure cases into three classes.[9] The first class of failure cases has bright or dark spots, which seem to confuse the network. The second class of failure cases is when information has been destroyed in preprocessing. The last class is when multiple signs are visible within the image.

For the failure case of bright spots we tried two things to fix it. The first try was to feed the network images in YUV color space in the hope. YUV spaces provides one channel for brightness and two for channels color information. Our intention was that the network would learn to ignore strong fluctuations in brightness. The result after retraining the network was sobering, as it performed even worse than using RGB images. So we tried harder and applied a histogram equalization to the brightness channel with the intention to smooth out these bright spots. The result of this try was still worse than using RGB information.

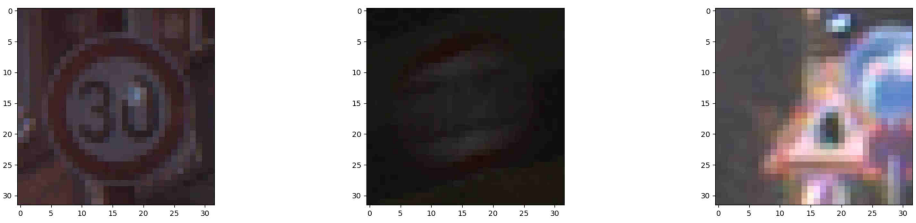

Figure 9: Examples of images which were misclassified. One image out of each failure case. Left to right: 1. Bright/Dark spot. 2. Information destroyed by preprocessing. 3. Multiple Signs.

4 Discussion and Conclusion

Through the practical we went from simple feature extraction and classification methods, to state of the art computer vision techniques, namely Convolutional Neuronal Networks, on the GTSRB

[Houben et al., 2013]. While predictions with manual feature selection can terribly fail, like in the case of a combination of SVM and HOG features, neural networks provide a more sophisticated way to learn image recognition, as the network itself decides which features it needs, depending on the constructed architecture.

We have chosen to take an architecture similar to AlexNet [Krizhevsky et al., 2012] and successfully trained it to superhuman performance in image recognition. The architecture is exhaustively described in the previous section. Out network is able to perform in the GTSRB [Houben et al., 2013] on par with the best methods to presented and only recently has been outperformed by a CNN using spacial transformer networks [Arcos-García et al., 2018].

A novelty we present is the dropout in the convolutional layers to fight overfitting in early layers. While this approach yields no additional accuracy for out network it successfully reduces the networks confidence when predicting wrong labels in many cases. This insight could be useful when combining multiple networks to a committee or when using weighted voting approaches.

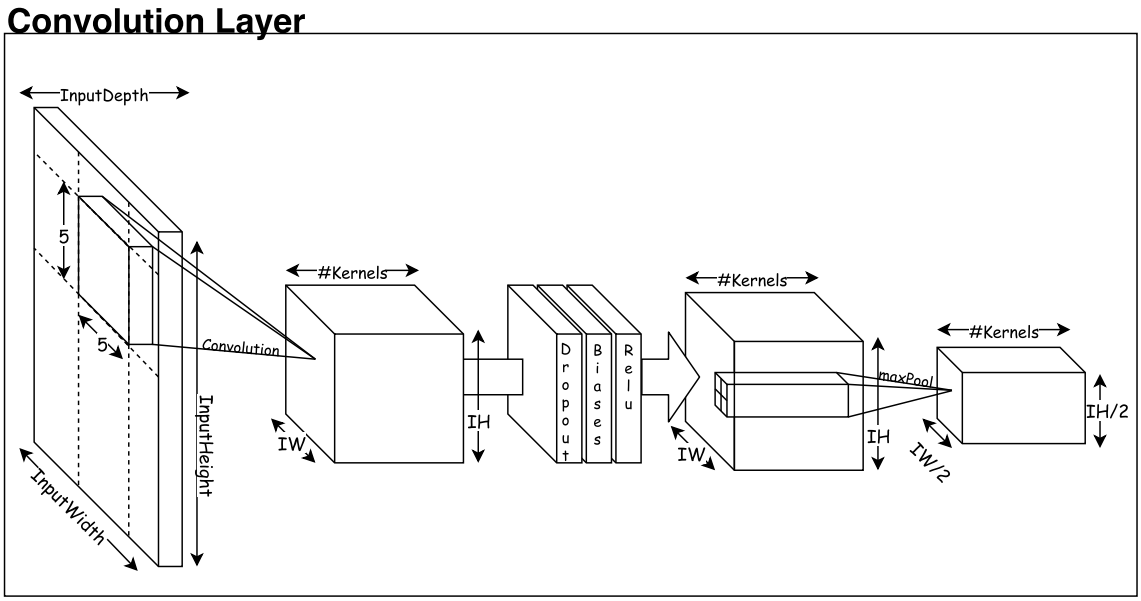

Figure 10: The convolution layers are special, as they include dropouts before the rectified linear unit to reduce the confidence in obviously wrong results.

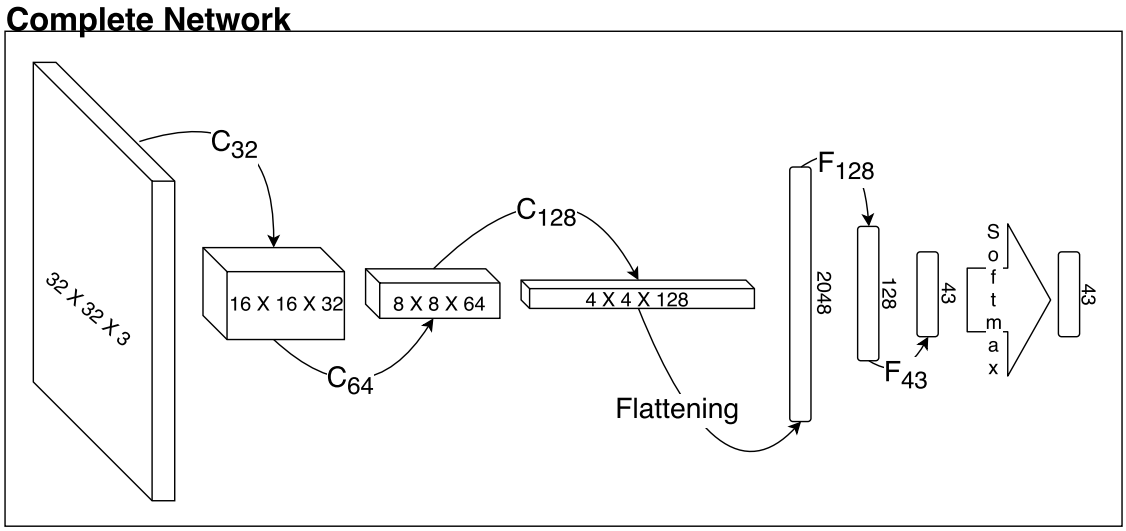

Figure 11: The final architecture of our convolution neural network for traffic sign recognition. We took AlexNet [Krizhevsky et al., 2012] as our paragon and therefore derived an architecture consisting of three convolution (C) and two fully connected layers (F). **Legend:** $C_x :=$ Convolutional Layer with $\#Kernels = x; F_x :=$ Fully Connected Layer with $\#Outputs = x$ and a bias vector the size of the input.

References

[Abadi et al., 2015] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[Arcos-García et al., 2018] Arcos-García, Á., Álvarez-García, J. A., and Soria-Morillo, L. M. (2018). Deep neural network for traffic sign recognition systems: An analysis of spatial transformers and stochastic optimisation methods. *Neural Networks*.

[Bradski, 2000] Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

[Cortes and Vapnik, 1995] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

[Dalal and Triggs, 2005] Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE.

[Houben et al., 2013] Houben, S., Stallkamp, J., Salmen, J., Schlipsing, M., and Igel, C. (2013). Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. In *International Joint Conference on Neural Networks*, number 1288.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

[Kohavi et al., 1995] Kohavi, R. et al. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Montreal, Canada.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.