Vehicle Identification using Convolutional Neural Networks

Saygın Zeybekoğlu
Department of Computer Science
Yeditepe University
Istanbul, Turkey
saygin.zeybekoglu@std.yeditepe.edu.tr

Mustafa Köse
Department of Computer Science
Yeditepe University
Istanbul, Turkey
mustafa.kose1@std.yeditepe.edu.tr

Görkem Kar
Department of Computer Science
Yeditepe University
Istanbul, Turkey
gkar@cse.yeditepe.edu.tr

***Abstract*—In this research, a vehicle identification neural network will be implemented using a vehicle data set. The training and test data will be collected from Yeditepe Vehicle Dataset Images (2019). The results will be evaluated based on success rate.**

A model has been used based on convolutional neural network. VGGNet is the network. VGGNet is a neural network that implemented very good in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2014.

In this work, neural networks will be implemented for identifying vehicle models, using a vehicle data set. Evaluated by the tests, VGGNet is more suitable for this vehicle data set.

In conclusion, pretrained VGGNet is found that its success rate is about 70% for rank1.

***Index Terms*—Dataset, Artificial Intelligence, Vehicle, Car, Identification, Convolutional, Neural, Network, Yeditepe, CNN, NN, Learning Rate, LR**

I. Introduction

Main of this work is the image classification to create an application that identifies vehicle. It has been observed in many films and series. Have you ever seen the series, Black Mirror, Person of Interest, etc. or the movie, Minority Report, etc.?

The data set, what is named "Yeditepe Vehicle Dataset", is created by the authors. The goal of this data set, is used to be driven any vehicle which is in Turkey by the people.

II. Previous Works

A. CNNs

In last years, there are too much works on image classification and processing. Some of them use Convolutional Neural Network. The strength of CNN is as follows; their capacity for learning.

Machine learning and neural networks have been leading up to this point: Understanding Convolutional Neural Networks (CNNs) and the role they play in deep learning.

In traditional feedforward neural networks, each neuron in the input layer is connected to every output neuron in the next layer – this is called a fullyconnected (FC) layer. However, in CNNs, FC layers aren't used until the very last layer(s) in the network. Thus a CNN can be defined as a neural network that swaps in a specialized "convolutional" layer in place of "fullyconnected" layer for at least one of the layers in the network.

A nonlinear activation function (ReLU, Sigmoid, tanh) is applied to the output of these convolutions and the process of convolution => activation continues (along with a mixture of other layer types to help reduce the width and height of the input volume and help reduce overfitting) until the end of the network is finally reached and applied to one or two FC layers where the final output classifications can be obtained.

Each layer in a CNN applies a different set of filters, typically hundreds or thousands of them, and combines the results, feeding the output into the next layer in the network. During training, a CNN automatically learns the values for these filters.

In the context of image classification, the CNN may learn to:

- Detect edges from raw pixel data in the first layer.
- Use these edges to detect shapes (i.e., "blobs") in the second layer.
- Use these shapes to detect higherlevel features such as facial structures, parts of a car, etc. in the highest layers of the network.

The last layer in a CNN uses these higherlevel features to make predictions regarding the contents of the image. In practice, CNNs give two key benefits: local invariance and compositionality. The concept of local invariance allows to classify an image as containing a particular object regardless of where in the image the object appears.

The second benefit is compositionality is that each filter composes a local patch of lower-level features into a higher-level representation, similar to how a set of mathematical functions can be composed that build on the output of previous functions: f(x(g(x(h(x))) – this composition allows the network to learn more rich features deeper in the network.

B. Learning Rate

Learning rate is a hyper-parameter that controls how much the weights of our network are being adjusted with respect the loss gradient. The lower the value, the slower the travel along the downward slope. While this might be a good idea (using a low learning rate) in terms of making sure that any local minimums are not being missed. The learning rate controls size of the step is indicated.

C. Transfer Learning

Transfer learning proposes a different training paradigm. In context of the proposed challenges above, a CNN would be first trained to recognize cars vs trucks. Then, the same CNN would be used trained on car and truck data to be used to distinguish between airplane classes, even though no airplane data was mixed with the car and truck data.

D. Fine-grained Classification

The FineGrained Image Classification task focuses on differentiating between hard-to-distinguish object classes, such as species of birds, flowers, or animals. However, in our research is identifying vehicle make and model.

III. Implementation

This project's main objective is predicting and classifying the cars by making use Yeditepe Vehicle Dataset Images. Firstly, a data set having 1040 images of 18 vehicles of different make and model was gathered. The data set has been generated by gathering pictures from search engines like Google, Bing, etc. where noise free pictures, which are necessary for training two different convolutional models, are provided.

To lighten the time and data constraints, we chose an approach focused around using transfer learning to swiftly create and train neural networks. For instance, we used VGGNet, a deep learning framework, to build, train, and test the networks. The following subsection describes the models used in this project.

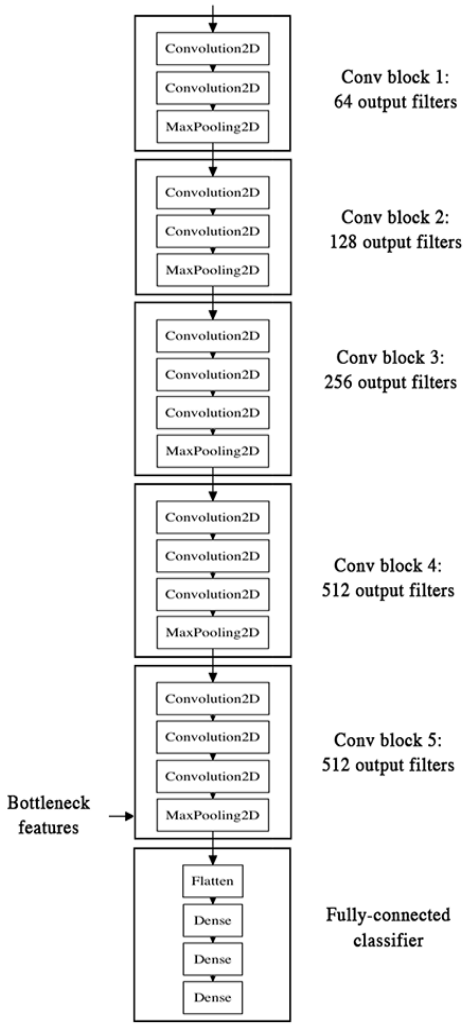

Fig. 1. VGG16 Architecture

A. VGG16

VGG16 is a neural network that performs very well with large information, as it studies with 3x3 convolution filters where the depth is 16 weights layers. Other parameters of the architecture are fixed at first, and then consistently increment the depth of the network by adding more convolutional layers, which is possible because of the use of very small (3×3) convolution filters in all layers. ReLU is using both convolution and fully connected layers. There are about 138 million parameters. As seen in the figure above:

B. Fragments of the Dataset

The dataset, which is named Yeditepe Vehicle Dataset Images, has a config module whose contents will be saved in the file. The dataset has been generated by supplying pictures from search engines like Google, etc. to noise free pictures are given out as output pictures which are necessary for training in executing VGGNET convolutional model. This system has been used to get In the whole of the dataset, 18 models of different cars have been used in which 165 are testing set and 770 are training set, cars such as Sedans, SUVs, Coupes, Convertibles, Hatch-bags and Station Wagons. The data's in Yeditepe Vehicle Dataset has images which have been cropped. This file will include all essential configurations to build the Yeditepe Vehicle Dataset and fine-tune VGG16 on it. The data set has fixed points of cropping box using which we have achieved our desire form of pictures. An example of cropping box is given above:

The build_dataset file will keep the image paths and class labels and then a .lst file for each of the training, validation, testing and traning sets will be generated respectively. Once these .lst files have been created, mxnet's im2rec binary can be applied to construct the record database.

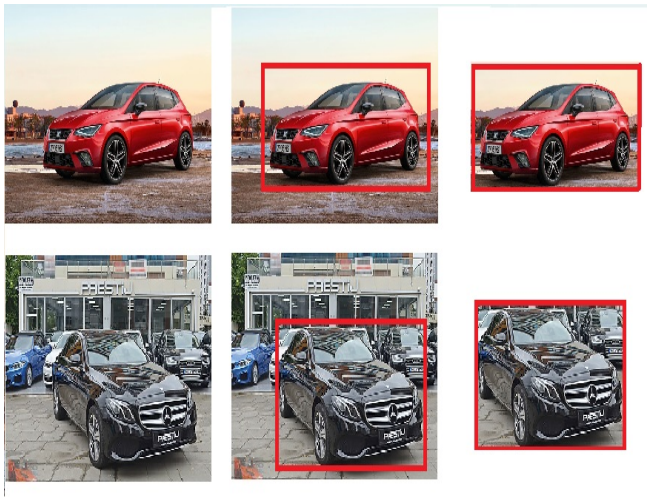

Fig. 2. Cropping box

C. Training the Network

The VGG16 network architecture assumes input images that are 224 × 224 pixels with 3 channels; however, recall from rec.py that .rec files were created with images that are 256 pixels along their shortest dimension.

In order to fine-tune VGG16, the SGD optimizer will be used. A smaller learning rates of 5e-4 and 1e-4 will be used initially - Future experiments will help to determine what the optimal initial learning rate is. A momentum of 0.9 and L2 weight regularization term of 0.0005 will also be trained with. A single GPU will be used to fine-tune VGG on the Yeditepe Vehicle Dataset as epochs will only take 18 seconds. Next, the path will be built to output checkpointsPath, as well as initialize the argument parameters, auxiliary parameters, and whether or not "missing" parameters are allowed in the network.

In this context, "missing parameters" are parameters that have not yet been initialized in the network. Normally uninitialized parameters would not be allowed; however, fine-tuning requires to slice off the head of the network and replace it with a new, uninitialized fullyconnected head. Therefore, if epoch zero were be training from, missing parameters will be allowed.

Provided that the pre-trained models are loaded, –model weights from file, allowMissing is also to be updated to be True as the head of the network is being about to replace. Replacing the head of the network is not as straightforward as Keras (as it involves some investigative work in the layer names), but it's still a relatively straightforward process:

The call the fit method in the fine-tuning example is a bit more verbose than the previous ImageNet examples because most of our previous parameters (e.x., arg_params, aux_params, etc.) could be passed in via the FeedForward class. However, since either performing network surgery or loading a specific epoch are being relied on, all these parameters are needed to move into the fit method.

Also, notice how the allow_missing parameter is supplied to mxnet, enabling the library to understand that fine-tuning are being attempted to perform. num_epoch to 55 will be setted maximum – this number may increase or decrease depending on how the training process is going.

D. Testing the Vehicle

VGG16 has been fine-tuned on the Yeditepe Vehicle Dataset, let's move on to evaluating the performance of the network on the testing set. To accomplish this process, test_cars.py file is created.

The car_config (aliased as config) will be needed so the car dataset specific configuration and variables can be accessed. The rank5_accuracy function will be imported so the rank-1 and rank-5 accuracy can be computed on the testing set, respectively. The mxnet library is then imported so the mxnet Python bindings can be accessible.

The list of predictions and ground-truth targets are initialized. The iter_predict function is used to allow to loop over the testIter in batches, yielding both the predictions (preds) and the ground-truth labels (batch). The predictions are grabbed from the network and be converted it to a NumPy array. This array has the shape of (N, 18) where N is the number of data points in the batch and 18 is the total number of class labels. The ground-truth class label is extracted from the testIter, again being converted it to a NumPy array. Given both the preds and ground-truth labels the predictions and targets lists are now able to be updated, respectively. Ensures that both the targets and predictions lists are the same length. This line is a requirement as the iter_predict function will only return batch in sizes of powers of two for efficiency reasons (or it could also be a small bug in the function). Thus, it's nearly always the case that the targets list is longer than the predictions list. The discpreancy can be easily fixed by applying array slicing. The final step is to take the predictions and targets from the testing set and compute our rank-1 and rank-5 accuracies.

E. Predicting the Vehicle

Earlier, the iter_predict function was learned how to use to loop over the data points in an ImageRecordIter and make predictions. The process of preparing an image for classification with mxnet is not unlike the methods for Keras. The cv2 bindings are imported to the OpenCV library. The image pre-processors are imported, initially designed for Keras, but they can be easily reused when working with mxnet as well.

The serialized LabelEncoder loads so the integer class labels which is produced by mxnet and convert them to human readable labels (i.e., the make and model names) can be converted. The contents of the test.lst file is loaded and sampled –sample-size rows from them. Recall that the test.lst file contains the paths to the testing images. Therefore, to visualize the predictions, the rows simply need to sample from this file. The serialized VGG16 or Inception V3 networks can be loaded from disk.

The AspectAwarePreprocessor which will resize images to 224 × 224 pixels is instantiated. The MeanPreprocessor will

perform mean subtraction using the RGB averages from the Simonyan and Zisserman paper – the means are not computed from the training set as fine-tuning is being done and the averages must be used computed over the ImageNet dataset. Finally, the ImageToArrayPreprocessor was initialized. Originally, this class was used to convert raw images to Keras-compatible arrays based on using “channels last” or “channels first” the keras.json configuration file. However, since mxnet always represents images in channels first ordering, the dataFormat="channels_first" parameter is needed to supply to the class to ensure the channels are ordered properly. It’s time to loop over the sample images, classify them, and display the results to the screen.

The ground-truth target and imagePath from the input row are extracted, followed by converting the target to an integer. Respectively,

- Load our input image.
- Clone it so we can draw the output class label visualization on it.
- Start the pre-processing stage by resizing the image to have a maximum width of 500 pixels.
- Apply all three of our image pre-processors.
- Expand the dimensions of the array so the image can be passed through the network.

Classifying an image using a pre-trained mxnet network is as simple as calling the predict method of model.

The predictions are returned for each of the 164 class labels. The indexes of these labels then are sorted according to their probability (from largest to smallest), keeping the top-5 predictions. It displays the human readable class name of the ground-truth label, while the number 1 predicted label is drawn to image, including the probability of the prediction. The last code block handles printing the top-5 predictions to terminal and displaying output image.

IV. Test and Results

A. Experiment-1

In the first VGG16 fine-tuning experiment, the learning rates are decided to split and start training with a 5e-4 learning rate. SGD optimizer, momentum, and regularization terms were kept as momentum=0.9 and wd=0.0005.

The first 50 epochs are studied. The training loss/accuracy keeping pace with the validation can be seen, and eventually overtaking the validation loss/accuracy. The network was allowed from 46 epoch to continue training until epoch 56. And then, the learning rate was reduced from 5e-4 to 5e-5 and trained for another ten epochs.

While there is certainly a gap between the training and validation loss, the validation loss is not increasing. Looking at the output of the 56th epoch, the network was reaching 70.00% for rank-1 and 96.36% for rank-5 accuracy on the validation set.

Between 67th epoch and 100th epoch is better and slowly increasing predicting rank-1 test vehicles. The algorithm is not hurting although learning rate is 5e-6.

Epoch	Testing	LR	Prediction	Training	Validation
20th	61.33%	5E-4	83	90.40%	60.67%
46th	65.33%	5E-4	94	99.04%	70.00%
56th	64.00%	5E-5	95	99.31%	70.00%
67th	64.00%	5E-6	94	99.31%	71.33%
85th	68.00%	5E-6	96	99.45%	72.67%
100th	67.33%	5E-6	97	99.59%	74.00%

TABLE I
Experiment-1 Accuracies Table for Different Epochs

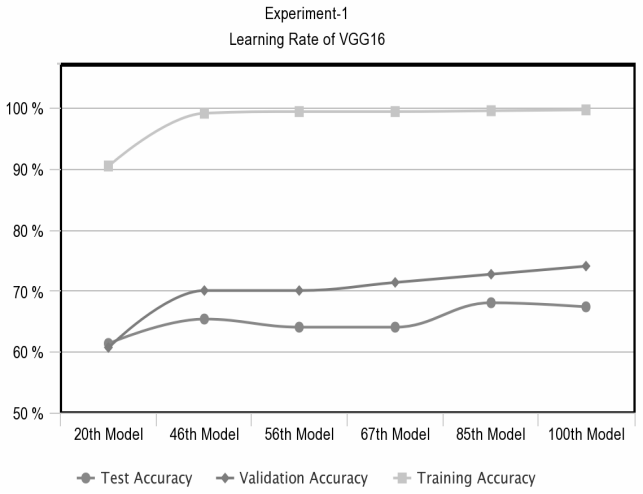

Fig. 3. Experiment-1

These accuracies are certainly better than the accuracies obtained in the other experiments. I would also argue that these accuracies are more desirable than the experiment as the network was not overfitted and risk the ability of the model to generalize. At this point, the experiment was felt comfortable stopping altogether and moving on to the evaluation stage.

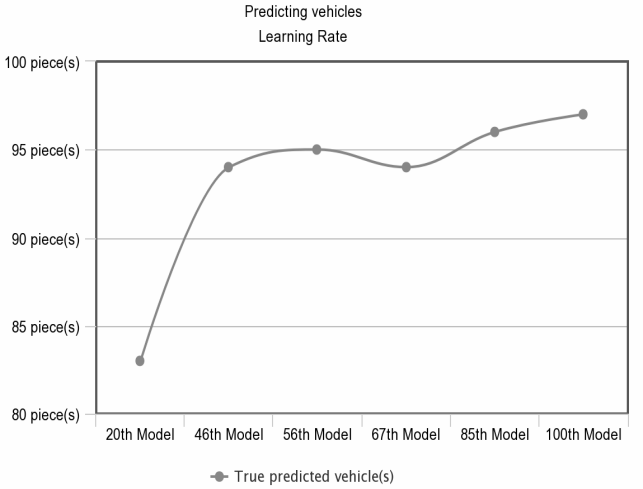

Fig. 4. Vehicle prediction

Epoch	Testing	LR	Prediction	Training	Validation
43rd	60.00%	1E-4	79	96.16%	66.67%
44th	66.00%	1E-4	84	95.75%	66.00%
45th	63.33%	1E-4	85	98.35%	68.00%
62nd	64.67%	1E-4	95	99.04%	70.67%
86th	67.33%	1E-4	94	99.45%	72.00%
92nd	66.00%	1E-4	96	99.18%	71.33%
97th	68.67%	1E-4	98	98.63%	66.00%

TABLE II
EXPERIMENT-2 ACCURACIES TABLE FOR DIFFERENT EPOCHS

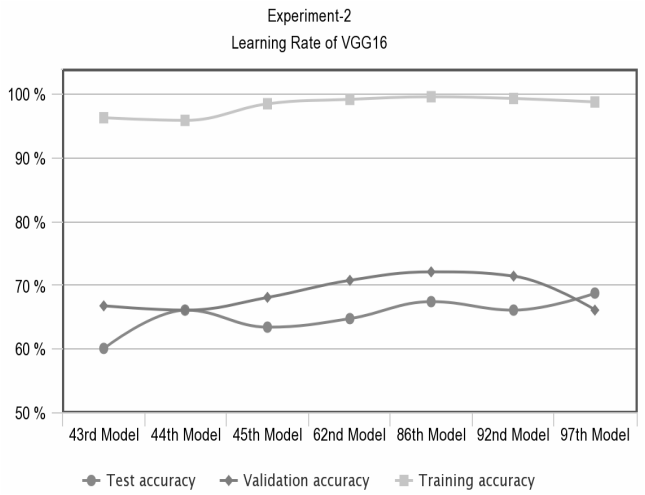

Fig. 5. Experiment-2

B. Experiment-2

In the last VGG16 fine-tuning experiment, same SGD momentum and weight decay were kept. Only the learning rate was 1e-4.

The 100 epochs are studied. The training loss/accuracy is done perfectly. However the network could be done as validation loss/accuracy which stays close to the same in a different way. Anyway, the learning rate was 1e-4 that is all around the network.

While 45th epoch is going to 62nd epoch, predicting vehicles' column is suddenly increased. Although 62th model of network is pleasing, it is rising steadily towards the hundredth model.

97th epoch is better and however slowly increasing predicting rank-1 test cars. The algorithm is not hurted because of the learning rate which is 1e-4.

V. CONCLUSION

In summary, the experiments which are with various learning rates are evaluating the success rates. Second experiment is a little well than the first experiment, however nearly same. When the tables and figures are looked at, it is seen that 97 and 98 vehicles are close to the prediction vehicles, 67% and 68% on testing accuracies same too, respectively experiments.

Varying learning rates are superior than constant learning rate. By adjusting the learning rate on an epoch-to-epoch basis, loss can be reduced, increase accuracy, and even in certain

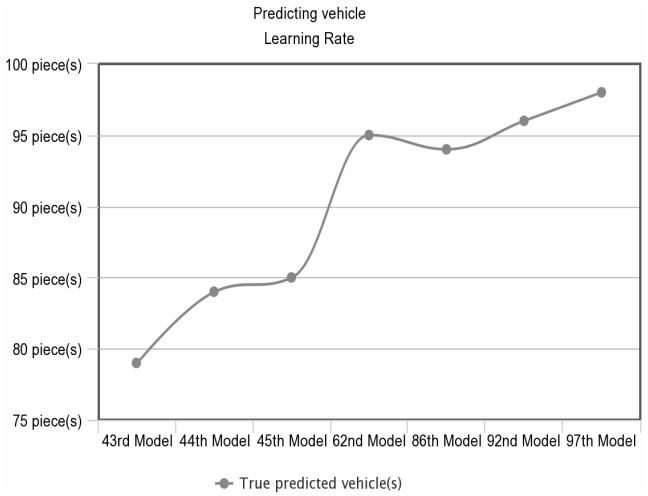

Fig. 6. Vehicle prediction

situations reduce the total amount of time it takes to train a network.

Therefore, view the process of learning rate scheduling as:

1) Finding a set of reasonably "good" weights early in the training process with a higher learning rate
2) Tuning these weights later in the process to find more optimal weights using a smaller learning rate.

There are two primary types of learning rate schedulers that these will be likely encountered:

1) Learning rate schedulers that decrease gradually based on the epoch number (like a linear, polynomial, or exponential function (for using this)).
2) Learning rate schedulers that drop based on specific epoch (such as a piecewise function).

However the dataset is the most important after all.

VI. FUTURE WORK

Our future goal is to increase the prediction rates.For that we would need train our system with more data.Further our system focused on detecting a single vehicle.In the future CNN could be modied to detect multiple vehicles in images.

REFERENCE AND BIBLIOGRAPHIES

[1] Krause, Jonathan, et al. "3d object representations for fine-grained categorization." Proceedings of the IEEE International Conference on Computer Vision Workshops. 2013.

[2] Szegedy, Christian, et al. "Going deeper with convolutions." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.

[3] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

[4] Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." European Conference on Computer Vision. Springer International Publishing, 2014.

[5] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

[6] Zagoruyko, Sergey, and Nikos Komodakis. "Learning to compare image patches via convolutional neural networks." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.

[7] Liu, Derrick, and Yushi Wang. "Monza: Image Classification of Vehicle Make and Model Using Convolutional Neural Networks and Transfer Learning."

[8] Jia, Y. Q. C. "An Open Source Convolutional Architecture for Fast Feature Embedding." (2013).

[9] Krause, Jonathan, et al. "Learning Features and Parts for Fine-Grained Recognition." ICPR. Vol. 2. No. 7. 2014.

[10] Ramnath, Krishnan, et al. "Car make and model recognition using 3D curve alignment." IEEE Winter Conference on Applications of Computer Vision. IEEE, 2014.

[11] Mojumder, Uttam, Sarker Toqi Tahamid, Monika Gulnahar Mahbub, Ratul Nurul Amin. “Vehicle Model Identification using Neural Network Approaches”, Braac University (2016)

[12] Adrian Rosebrock, “vgg16”, “ImageNet”, “Deep Learning for Computer Vision with Python”, pyimagesearch [Online] Available: https://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/

[13] M. Stark, J. Krause, B. Pepik, D. Meger, J. J. Little,B. Schiele, and D. Koller. Fine-grained categorization for 3d scene understanding. In BMVC, 2012.

[14] TFLearn: Deep learning library featuring a higher-level API for TensorFlow.

[15] Sermanet, Pierre, et al. "Overfeat: Integrated recognition, localization and detection using convolutional networks." arXiv preprint arXiv:1312.6229 (2013).

[16] Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." Journal of Machine Learning Research 15.1 (2014): 1929-1958.

[17] Mxnet, [Online], Available: http://data.mxnet.io/models/imagenet/vgg/vgg16-symbol.json, http://data.mxnet.io/models/imagenet/vgg/vgg16-0000.params.

[18] F. Chollet et al., “Keras.” https://keras.io, 2015

[19] Plemakova, Victoria, Vehicle Detection based on Convolutional Neural Networks, Tartu University (2018)

[20] Rosebrock, Adrian, Deep Learning for Computer Vision with Python: Starter Bundle. 2nd ed., vol. 1 3, PyImageSearch, 2017.

[21] Rosebrock, Adrian, Deep Learning for Computer Vision with Python: Practitioner Bundle. 2nd ed., vol. 2 3, PyImageSearch, 2017.

[22] Rosebrock, Adrian, Deep Learning for Computer Vision with Python: Imagenet Bundle. 2nd ed., vol. 3 3, PyImageSearch, 2017.