UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Anton Prokopov

High-Value Target Detection

Master's Thesis (30 ECTS)

Supervisor: Assoc. Prof. Gholamreza Anbarjafari

Tartu 2018

Kõrge väärtusega sihtmärkide avastamine

Lühikokkuvõte:

Lõputöö kirjeldab automatiseeritud kõrge väärtusega sihtmärkide tuvastamist, mille eesmärk on inimoperaatori töökoormuse vähendamine analüüsides etteantud videot. Selle magistritöö põhieesmärk on uurida üldise närvivõrkude mehhanismi ning olemasoleva eeltreenitud närvivõrgu tuunimine sobivate andmetega. Edasised närvivõrgu parameetrite korrigeerimised oli vaja teostada, et saavutada paremad tulemused kõrge väärtusega sihtmärkide tuvastamises reaalajas. Seda tüüpi süsteem on suuteline tuvastama erinevaid sihtmärkide klasse olenevalt sellest, mis andmetega süsteem oli treenitud. Uurimistöö oli fokuseeritud tuvsatama kindlaid automarke ja mudeleid, aga tuvastatavaks objektiks saab olla ükskõik mis asi. Süsteemi võib potentsiaalselt rakendada järelvalves, piiripatrullis, loomade jälgimises.

Võtmesõnad:

Närvivõrgud, Autode Klassifitseerimine, Inception, Xception, NASNet, Fine-tuunimine

CERCS: Pilditehnika (T111)

High-value target detection

Abstract:

This work describes an automatic high-value target detection for the purpose of reducing the human workload in analyzing video feed from the given source. The aim of this thesis is to investigate the mechanism of neural networks in general and to fine-tune an existing pre-trained neural network with suitable data. Further adjustment of the parameters was required to achieve better results in performing robust target detection in real-time. This kind of a system can recognize different classes of targets depending on the data it was trained with. The research was focused on detecting particular car marks and models, but target could be defined as any object. The potential application of such system could be found in surveillance systems, border control, monitoring the animals.

Keywords:

Neural Networks, Car Classification, Inception, Xception, NASNet, Fine-tuning

CERCS: Imaging, image processing (T111)

Contents

Chapter 1

Introduction

Video content plays significant role in every person's life. Every minute 300 hours of videos are uploaded to YouTube [1], which means more video content is produced every minute, than a person is able to watch in 2 weeks. YouTube mostly hosts educational or entertaining content, but what about different kinds of surveillance cameras that film non-stop 24 hours a day every week? It is impossible for a person or even thousands of people to watch all the surveillance footage, needles to mention that the attention span of a person is quite short, thus the focus dissolves after first hours of watching. Thankfully, people do not need to watch all the surveillance videos, as mostly nothing happens there. The point of surveillance is to be immediately notified, when a critical issue arises. Taking into account modern image processing techniques, it is possible to define, what in the video sequence is considered to be an object of interest. After that computers can analyze by themselves all the videos they produce and notify the human operator only when needed.

Objects of interest in this thesis are further referred to as high-value targets. If such a target appears in a video sequence of a surveillance camera, the operator definitely has to be notified. The beauty of this approach is that anything could be a target. It is possible to train the model on images of people to make people into a high-value target in a restricted or secured zone, where people are not allowed. Model can be trained on planes to notify the operator, if any plane is in the monitored area of the sky. It is possible to train the model to recognize military vehicles for automated border patrol.

The solution for detecting different high-value targets is essentially the same - neural network trained on appropriate data. The algorithm of training is the same, network architecture stays the same. Depending on the purpose, the network parameters can be fine-tuned to improve the accuracy, but mostly the detection depends on the data the network was trained with. As the final output is data driven, a neural network can cover a wide range of applications, given big enough (hundreds of thousands of pictures) datasets to train with.

In this thesis the research had to be limited to one specific focus area, as collecting huge datasets for neural networks to support particular domains is not a trivial task and requires a lot of time effort. It was decided to train the neural network with open-sourced labeled datasets of cars, so the system is able to automatically classify the cars driving around the city by mark and model. In case a BMW X5 is caught by the video camera, the system should instantly signal to the operator, that BMW X5 is detected. BMW in this scenario is a high-value target. The system will not notify the operator, if any other car is detected. Such scenarios are configurable. For example, it is possible to fire notifications, if BMW or Ford are detected, not depending on the model at all. Another possibility is to fire the notification every time a random car is detected and the confidence threshold is not surpassed. In this case it means that most likely an unknown car was detected. The operator can then manually add a label to such car and launch the retraining of the neural network, so next time the car is recognized.

Chapter 2

Background

This chapter gives an overview of the existing techniques and methods for working with neural networks on the topic of image classification. Existing datasets that contain labeled images of cars are also introduced in this chapter. This data was used to train different architectures of neural networks and obtain the results discussed in the "Experimental Results" chapter. The description of the used neural network architectures is given in here as well.

2.1 Related works

This section describes some state-of-the-art techniques for:

1. building image classification and localization systems,
2. vehicle classification and tracking,
3. multi-label sequential classification,
4. efficient utilization of temporal domain, when processing video sequences.

Each category is described in its own subsection and is represented by one or more papers on the topic.

2.1.1 Object classification and localization

One of the most popular examples of real-time object detection systems is **YOLO - You Only Look Once** - that was proposed in [2]. Further improvements were made to the system to produce YOLOv2 [3], which is a state-of-the-art system on such image classification tasks as Pascal VOC [4] or MS COCO [5]. YOLOv2 can

be configured according to different sizes in order to improve the accuracy (more parameters, bigger size, takes more time) or the speed (less parameters, smaller size, less accuracy).

The performance of YOLO is measured with mAP - Mean Average Precision. To calculate it for Object Detection, it is needed to calculate the average precision for each class in the data based on the model predictions. Average precision is related to the area under the precision-recall curve for a class. Then taking the mean of these average individual-class-precision gives the Mean Average Precision.

YOLOv2 outperforms Faster R-CNN [6], Residual Network (ResNet) [7] and Single Shot Detector (SSD) [8] models, while still running significantly faster. It achieves 76.8 mAP on VOC 2007 with 67 FPS, 78.6 mAP with 40 FPS.

YOLO9000 is a system that jointly trains object detection and classification [3]. It is trained on COCO detection and ImageNet classification datasets. Joint training allows YOLO9000 to predict detections for object classes, that do not have labels. YOLO9000, having data only for 44 out of 200 classes gets 19.7 mAP on ImageNet detection validation set.

Improvements made to YOLO to get YOLOv2.

Better in terms of accuracy:

1. batch normalization - adding it to all convolutional layers improves mAP by 2%, also helps to regularize the model and allows to remove dropout without overfitting.

2. high resolution classifier - YOLO trains the classifier network at 224 224 and increases the resolution to 448 for detection. YOLOv2 trains the classification network at full 448 448 resolution for 10 epochs on ImageNet. Increase by 4% in mAP.

3. convolution with anchor boxes - remove the fully connected layers from YOLO and use anchor boxes to predict bounding boxes for object localization. YOLO only predicts 98 boxes per image, but with anchor boxes the model predicts more than a thousand. Without anchor boxes the intermediate model gets 69.5 mAP with a recall of 81%. With anchor boxes the model gets 69.2 mAP with a recall of 88%.

4. dimension clusters - k-means clustering on the training set bounding boxes to automatically find good priors for the network to better adjust box dimensions.

5. direct location prediction - Using dimension clusters along with directly predicting the bounding box center location improves YOLO by almost 5% over the version with anchor boxes.

6. fine-grained features - add a pass-through layer that brings features from an earlier layer at 26 26 resolution. The pass-through layer concatenates the higher resolution features with the low resolution features by stacking adjacent features into different channels instead of spatial locations. 1% performance increase.

7. multi-scale training - change input dimension on the fly every 10 batches. The model downsamples by a factor of 32, so the dimensions are selected from the following multiples of 32: 320, 352, ..., 608. This means the same network can predict detections at different resolutions.

Better in terms of speed:

1. darknet-19 - 19 convolutional layers and 5 maxpooling layers, see full architecture on *Figure 2.1.* Darknet-19 only requires 5.58 billion operations to process an image yet achieves 72.9% top-1 accuracy and 91.2% top-5 accuracy on ImageNet.

2. training for classification - ImageNet 1000 class classification dataset for 160 epochs using stochastic gradient descent with a starting learning rate of 0.1, polynomial rate decay with a power of 4, weight decay of 0.0005 and momentum of 0.9 using the Darknet. Additional tricks: random crops, rotations and hue, saturation and exposure shifts.

3. training for detection - remove the last convolutional layer and instead add on three 3 3 convolutional layers with 1024 filters, each followed by a final 1 1 convolutional layer with the number of outputs needed for detection. Train the network for 160 epochs with a starting learning rate of 10e3 , dividing it by 10 at 60 and 90 epochs. Weight decay of 0.0005 and momentum of 0.9. Similar data augmentation to YOLO and SSD with random crops, color shifting, etc.

In [9] an improved vertion of Fast R-CNN is introduced. It is called **Faster R-CNN**. It is an object classification system with the following localization similar to YOLO. A Region Proposal Network (RPN) is introduced for guiding the attention of a neural network. It shares full-image convolutional features with the detection network, which enables nearly cost-free region proposals. RPN predicts object bounds at each position. It is further merged with Fast region-based CNN (R-CNN) into a single network, where RPN component tells R-CNN, where to look, and R-CNN then classifies the objects in the proposed attention spans. The model architecture is shown on *Figure 2.2* VGG-16 model runs at 5 frames per second on a GPU, achieving state-of-the-art accuracy on PASCAL VOC 2007, 2012 and MS COCO datasets. Compared with YOLO the model persorms significantly slower.

In order to unify RPN with Fast R-CNN object detection networks, a specific training scheme is proposed. First, fine-tuning of region proposal task is done,

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

Figure 2.1: Darknet-19 network architecture [3]

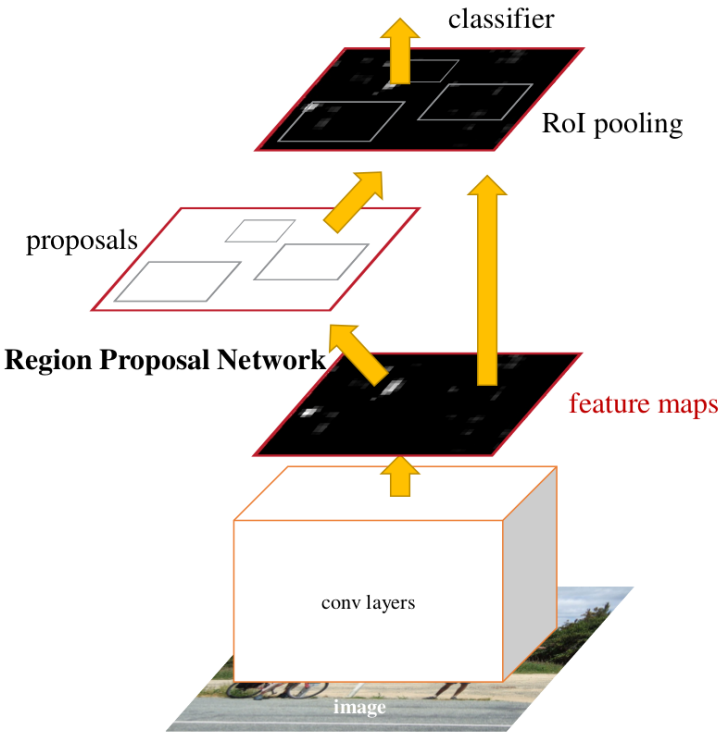

Figure 2.2: Faster R-CNN is a single network for object detection that unifies Region Proposal Network (RPN) with CNN [9].

then fine-tuning for object detection is performed, while keeping the proposals fixed. This scheme converges quickly and produces a network with convolutional features shared between both tasks. The effective running time for proposals is just 10 milliseconds.

Comparison of the Fast R-CNN and Faster R-CNN on MS COCO dataset can be seen on *Figure 2.3*.

			COCO val		COCO test-dev	
method	proposals	training data	mAP@.5	mAP@[.5, .95]	mAP@.5	mAP@[.5, .95]
Fast R-CNN [2]	SS, 2000	COCO train	-	-	35.9	19.7
Fast R-CNN [impl. in this paper]	SS, 2000	COCO train	38.6	18.9	39.3	19.3
Faster R-CNN	RPN, 300	COCO train	41.5	21.2	42.1	21.5
Faster R-CNN	RPN, 300	COCO trainval	-	-	**42.7**	**21.9**

Figure 2.3: Object detection results on MS COCO dataset with base model VGG-16.

2.1.2 Vehicle re-identification

Vehicle detection and classification tasks are relatively popular and during the past years significant progress has been made, but vehicle re-ID problem is hugely neglected. Mostly re-ID is focused on pedestrians [10]. Re-ID aims to identify a target vehicle in different cameras with disjoint views [11], [12], [13].

From a single vehicle image, given a set of gallery images for each vehicle, a global feature is generated, which is considered to be a descriptive representation containing all-view information.

Training pipeline of Adversarial Bi-directional LSTM Network (ABLN) consists of 3 subtasks [14], [15]:

1. CNN to learn features of vehicles' model and viewpoint, then LSTM to infer multi-view features from only 1 input view.
2. adversarial training architecture that treats one LSTM as a feature generator.
3. another LSTM for differentiating the real multi-view features and the inferred ones. The goal is to converge the inferred feature distribution to the target real feature distribution.

As a result, global all-view features are produced.

Extensive experiments showed that the ABLN can achieve promising results and outperform state-of-the-art methods for vehicle re-ID. Following table (*Figure 2.4*) shows the comparison with existing state-of-the-art methods for vehicle re-identification.

	Methods	rank-1	rank-5	rank-20	rank-50	mAP
VeRi	Improved Deep [1]	47.70	62.65	76.05	85.18	17.13
	DGD [37]	50.51	68.86	80.05	87.62	17.96
	LOMO [20]	23.89	40.32	58.61	73.96	9.03
	GoogLeNet [38]	51.98	66.79	78.77	86.37	17.58
	FACT [24]	52.35	67.16	79.97	87.09	18.54
	ABLN-32	58.14	74.41	85.87	92.35	22.91
	ABLN-16	55.51	72.03	84.02	90.97	20.81
	ABLN-Ft-16	**60.49**	**77.33**	**88.27**	**94.08**	**24.92**
VehicleID	Improved Deep [1]	42.73	60.92	77.57	89.09	-
	DGD [37]	44.72	66.68	81.35	90.31	-
	VGG+CCL [23]	43.62	64.84	80.12	89.29	-
	Mixed Diff+CCL [23]	48.93	75.65	88.47	93.37	-
	FACT [24]	39.85	58.47	74.98	86.36	-
	ABLN-32	**52.63**	**80.51**	**91.25**	**95.05**	-

Figure 2.4: Comparison of the proposed ABLN method for vehicle re-identification with the state-of-the-art methods.

2.1.3 Multi-label classification with CNN-RNN

CNNs are proven to be successful, when classifying images with a single label. However, in real world a single image often contains multiple objects in it. In [16] a unified framework for multi-label classification is proposed. A common approach is to utilize CNNs for multi-label classification by treating the problem as multiple single-label classifications. This method deals with every label independently, although multi-label problems exhibit strong label co-occurrence dependencies. The proposed method is to use recurrent neural networks (RNNs) to model the label dependencies. The RNN is designed to adapt the image features based on previous predictions, by encoding the attention models directly in the CNN-RNN architecture. Such design allows the network to recognize dominant objects and then shift to smaller ones.

The proposed RNN has several advantages compared to the state-of-the-art multi-label classification methods:

1. End-to-end model is employed to utilize the semantic redundancy and the co-occurrence dependency.
2. The model of high-order co-occurrence dependency with recurrent neurons is more compact and more powerful than other models solving this task.
3. The attention mechanism adapts the image features to better classify smaller objects that require more context.

RNN with LSTM is an effective model for long-term label dependency.

The proposed model consists of 2 parts:

1. The CNN part that extracts the semantic representations from images.

2. The RNN part that models image-label relationship and label dependency.

2.1.4 Classification in videos with 3DCNN

3D CNN based phantom object removing from mobile laser scanning data [17]. Mapping systems that use mobile laser scanning MLS are fast and able to produce high density point clouds with sufficient accuracy. The problem is that objects that are moving alongside the MLS are produced as long-drawn distorted structures in the resulting point cloud. Such an effect is called phantom effect. It is essential to remove the phantom effect from the data before further processing, as it creates unwanted noise.

It is quite a challenge, as characteristics of phantoms vary significantly. Even the static background regions of the MLS point clouds have a considerably inhomogeneous point density due to occlusions, different distances of the surface points from the scanner's trajectory, varying speed of the scanner platform, and the different laser adsorption/reflection properties of the different surface materials. Moreover moving objects may travel at varying speed, accelerating and breaking during the observation.

Object-based methods that first extract point cloud blobs by a geometric segmentation may produce errors that will affect the classification. Misdetected or erroneously merged object segments can mislead the subsequent categorization module.

Voxel-level methods are not very suitable, as with the increase of the number of voxels, the processing time and storage requirement also increase, making the method heavy.

Proposed solution is vertical column-based data structure for the CNN framework. Columns = pillars are defined in 2D. Neighboring pillars may correspond to the same object or a phantom. The horizontal extension is not restricted. 3D CNN does not need preliminary object extraction and separation. Training data is easy to prepare by assigning appropriate labels to the adjacent columns of the input point cloud.

Assumptions: the phantoms (such as vehicles and pedestrians) can be found on the ground, and have a maximum height of 3 meters. The point cloud is divided into pillars of 3m height, and a narrow base (0.2m $\times$ 0.2m), which are arranged into a dense regular 2D grid on the horizontal plane. 4 semantic classes in the cloud: ground, high objects (such as facades, tree trunks, lamp posts etc.), static

short objects (parking vehicles, street furniture etc), and phantom regions (effects of moving vehicles and pedestrians).

GoogLeNet [18] concept for building the CNN classifier. See *Figure 2.5* [17], which illustrates the architecture of the 3D CNN.

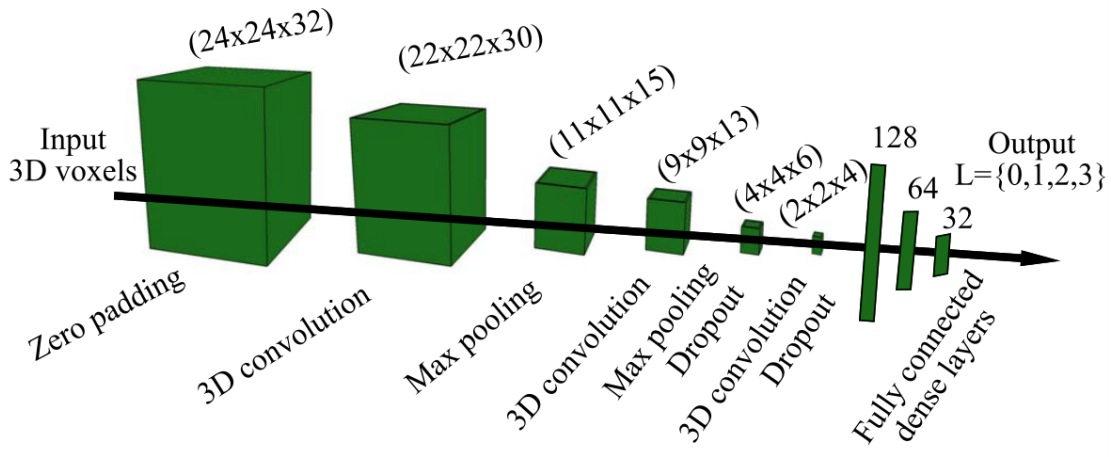

Figure 2.5: Proposed in [17] architecture for 3D CNN

Training took 24 hours using only CPU. Python and Theano for training, C++ and OpenGL for labeling tool. Trained the network with 40000 pillar examples. For each class F-rate above was 89%, while the overall result surpassed 91%. Tests confirmed our initial hypothesis: phantom detection cannot be handled by the density based filters which do not take into account any deeper structural information. On the contrary, the proposed 3D CNN framework can efficiently deal with the problem in this segment as well (93% F-rate).

The 3D CNN model is able to learn spatiotemporal features from time steps of occupancy grids and **predict human motion intentions** with an accuracy of 83% within 60% of the motion performed [19]. It also has real-time performance. The system is useful for human-robot interaction and human-computer interaction applications. It is generalised on other humans, who were not part of training set.

Existing methods: human arm was encoded as a multivariate Gaussian distribution; GMMs to represent classes of human reaching arm motion libraries; Hidden Markov Model (HMM) was used to model human arm motion (using skeleton joint information, depth and RGB images as multimodal input observations); Growing HMMs.

Inputs: markers on human bodies; special gloves or apparels; skeleton models; depth maps and point clouds; multimodal inputs. Point clouds are preferred in this paper.

The spatial information is captured from single 3D inputs while the temporal information from contiguous inputs. Hence, the 3D CNN architecture is able to

learn spatiotemporal relationship in a stream of point cloud data.

Point clouds from 3D sensor are converted to 3D occupancy grids. For each input time step, we have a data sample tensor V_t paired with the label Y_t , the target of a motion intention. Two-Stream convolutional networks for action recognition in videos. For a more recent and extensive introduction to CNNs, see [20]. See *Figure 2.6*, which illustrates the structure of the proposed 3D CNN. Excluded the use of Max pooling layers since we would like to preserve spatial information between layers.

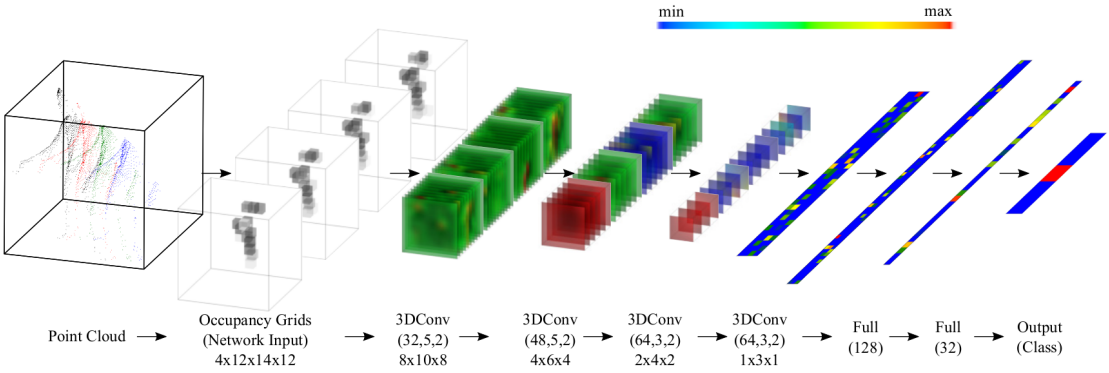

Figure 2.6: Proposed in [20] architecture for 3D CNN

119,102 dataset was used for training and 14,802 for testing. 3-fold cross-validation, Adam optimizer [21]. Dropout of 0.4 on all layers. Training with lookback-window size of 4 time steps takes 26 seconds per epoch on GTX 1080. Keras and Theano are used.

Lookback-window to tweak, 2 is the best. Bigger value gives more errors. Voxel size to tweak. A voxel size of 100 mm gave a more satisfactory result with highest accuracy of 84.37% and low computation time of 0.27 ms. During real-time tests, the whole prediction process could run at 10 fps.

More work is needed to improve the robustness of the model, to better generalise to new data sets and to learn more complex human motions with longer timespan and dynamics.

2.2 Datasets

In total there are 152 912 images in a merged dataset. 4 515 different classes, if taking every combination of mark-model-year as a separate class. There are over 125 distinct marks and over 1224 models, but the final version of the dataset only contains 73 classes and around 1100 models. Before any training has started the initial clean-up of the dataset was performed leaving only 125 classes of car marks and not taking models into account for the first experiments. The amount of

images left in the dataset was split into training and validation data with the ratio of 80/20. Training set contains 106017 images and validation set contains 26562 images. Cleaning the data was required as the dataset was combined from two different open-sourced datasets. Some of the labels were written differently, while representing the same classes. For example, Land-Rover was labeled as "land-rover" in one dataset and as "land_rover" in another one. Also a few typos in the labels were discovered. For example, Lamborghini was separated into two folders: "lamborghini" and "lamorghini". The cleaning of the dataset was performed a few more times after getting unsatisfied results.

2.2.1 A Large-Scale Car Dataset for Fine-Grained Categorization and Verification

Figure 2.7: Example images from CompCars dataset [22]

"A Large-Scale Car Dataset for Fine-Grained Categorization and Verification" [22] paper describes the dataset collected by researchers from Hong Kong and Shenzhen. The dataset is called "Comprehensive Cars" or "CompCars" for short. It contains 208 826 images of 1 716 car models from two scenarios: web-nature and surveillance-nature. Additionally coordinates of bounding boxes around the cars are carefully labeled, so more precise training can be achieved by not considering the background of a car picture. There are 136 727 images capturing the entire cars and 27 618 images capturing the car parts. Some example images are displayed on *Figure 2.7*.

Viewpoints, such as front, rear, left side, are also documented for most of the cars. Each car model is labeled with five attributes, including maximum speed, displacement, number of doors, number of seats, and type of car. Car models are organized into a large tree structure, consisting of three layers , namely car make, car model, and year of manufacture. See *Figure 2.8*.

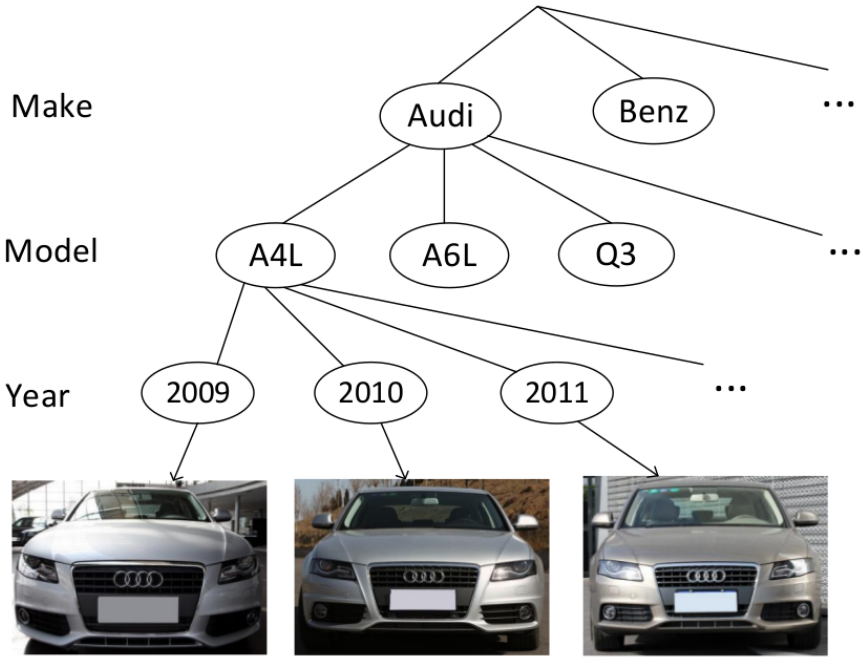

Figure 2.8: The schematic tree structure of the CompCars dataset

2.2.2 3D Object Representations for Fine-Grained Categorization

Figure 2.9: Example images from Stanford's Cars Dataset [23]

Stanford's fine-grained dataset [23]. It consists of BMW-10, a small, ultra-fine-grained set of 10 BMW sedans (512 images) hand-collected by the authors, plus car-197, a large set of 197 car models (16,185 images) covering sedans, SUVs, coupes, convertibles, pickups, hatchbacks, and station wagons. All types of cars made since 1990. When collecting the pictures from the Internet, it was ensured that dataset represents 197 visually distinct classes that were determined by a hashing algorithm, while scanning huge dataset of unlabelled cars from car selling platform. Some example images can be seen on *Figure 2.9*.

2.3 Neural Network Architectures

During the work on the car classificator multiple neural network architectures were tested. The list includes architectures, such as VGG16, VGG19 [24], Inception-v3 [25], Xception [26], DenseNet [27], ResNet50 [7]. ResNet has too many parameters - training and using this network requires lots of resources. Previously seen models with utilizing ResNet architecture could not achieve real-time performance. VGGs are too simple and produce less accurate results. They are used in the initial experiments to prove they are not suitable for the task of car classification. Inception is similar to Xception, but in this thesis it was decided to monitor both of them to ensure in the similarity. DenseNet and NASNet [28] are the two new architectures that were published in 2017 and added to Keras framework this year. There are 2 NASNets - large and mobile. Both NASNet Large and DenseNet produced similar results, so it was decided to use both NASNet architectures and not to use DenseNet.

Inception architecture is illustrated by the *Figure 2.10*. In Inception, the slight separation of image region and its channels can be seen. 1x1 convolutions are used to project the original input into several separate input spaces. From each of those input spaces a different type of filter is used to transform those smaller 3D blocks of data. Xception takes this one step further. Instead of partitioning input data into several compressed chunks, it maps the spatial correlations for each output channel separately, and then performs a 1x1 depth-wise convolution to capture cross-channel correlation.

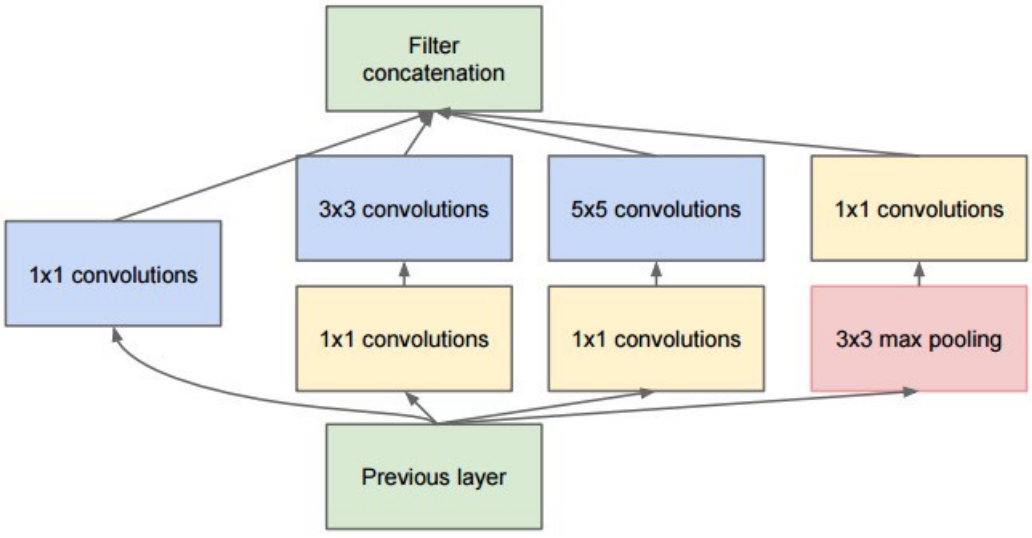

Figure 2.10: Inception architecture (with dimension reductions).

The architecture of Xception blocks is shown on *Figure 2.11*. The whole architecture based entirely on depth-wise separable convolution layers.

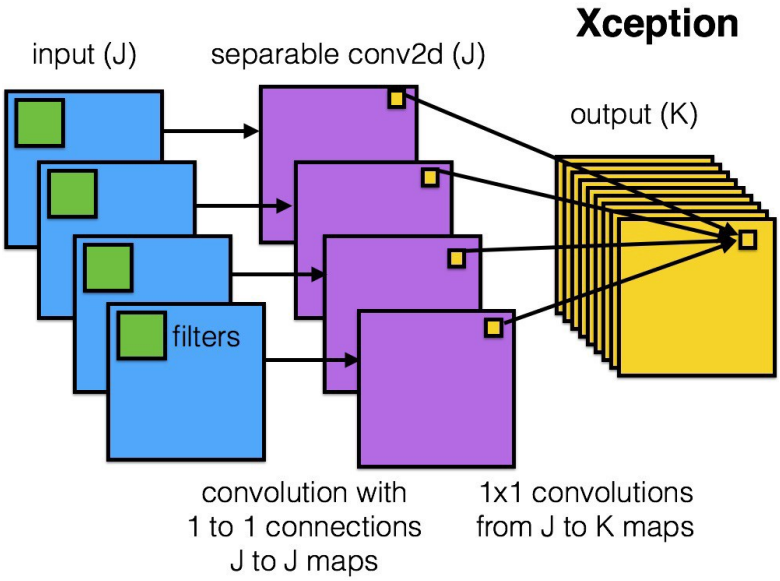

Figure 2.11: Xceptionn architecture

NASNet architecture is visually explained by the *Figure 2.12*.

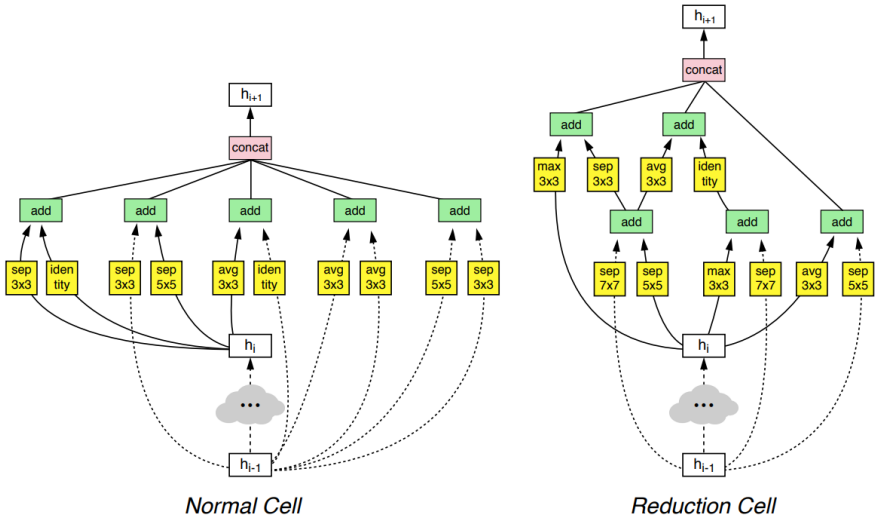

Figure 2.12: Architecture of the best convolutional cells (NASNet-A) with B = 5 blocks identified with CIFAR-10 . The input (white) is the hidden state from previous activations (or input image). The output (pink) is the result of a concatenation operation across all resulting branches. Each convolutional cell is the result of B blocks. A single block is corresponds to two primitive operations (yellow) and a combination operation (green).

Chapter 3

Technologies

This chapter describes the technological choices made to fulfill the car classification task. There are many of machine learning frameworks to choose from nowadays, but learning and trying all of them is not in the scope for this particular thesis. Keras [29] framework was chosen, as it simplifies the code writing, supports python and has lots of built-in network architectures, optimizers and image pre-processing tools. TensorFlow [30] was chosen as a back-end for Keras, as TensorFlow is the most popular [31] deep learning framework with lots of documentation and support articles. In order to speed up the training process, GPU power has to be utilized. Nvidia CUDA toolkit [32] helps with GPU usage and training parallelisation. TensorFlow is adjusted for GPU usage with CUDA toolkit.

To simplify the code writing even more, the Jupyter Notebook [33] was used. It allows to host the notebook on a server with a powerful GPU and then access and run the codebase straight from the browser. Additionally, it is possible to produce visual outputs, such as graphs, image previews, logs, that will be saved in the same file alongside the code.

3.1 Hardware

Multiple GPU resources were involved into training the neural networks described in this thesis. At first **MSI Trident 3 PC** with Nvidia GeForce GTX 1060 (3 GB, 8GB RAM) was used for training on smaller portion of the dataset to test different architectures and parameters. In order to launch the code for long (eg. 50 epochs), the codebase has to be tested and the bugs should be fixed. After the pre-processing of images, building up the models for fine-tuning and the training on a portion of data for 5-10 epochs works well, the training on the whole dataset can be launched. Training the networks on the whole dataset with only MSI Trident is quite slow. For example, training both large and mobile NASNets for 50 epochs each took more than 7 days to finish just to find out the code had some

bugs and the networks did not train properly.

The Rocket cluster of the University of Tartu [34] was considered as a more powerful option to train neural networks. It has 2 Nvidia Tesla P100 GPUs with total of 10 nodes (each with 12-15 GB VRAM). Rocket cluster uses SLURM system for submitting jobs. After submitting a job it might take up to 2.5 days to execute it. It complicates the debugging and bugfixing, so only tested code can be submitted for a job. However, tested code is adjusted for particular GPU, for example batch size for Nvidia GTX 1060 and Nvidia Tesla P100 is not the same. Also finding the right dependencies is not a trivial task. For example Keras is installed only for Python 3.6.0. Though Keras could be installed by a user via pip tool, some dependencies, such as CUDA, cannot be installed or updates without administrative rights. After lots of failed jobs and the time spent for setting up the work with Rocket, it was decided to try something different and less time-consuming. The latest errors were saying that the job is terminated because of memory exhaustion, although the batch size was set to smaller than is used on MSI Trident and the rest of the code was the same.

Paperspace [35] is a Cloud-based infrastructure for machine learning and data science. It is easy to set up and use, as it provides a snapshot of a cloud instance with all the necessary tools pre-installed (same Ubuntu 16.04 with TensorFlow, Keras, CUDA, Jupyter). An instance with Nvidia P4000 (8GB, 30GB RAM) is rented to train NASNet models. It takes about 40-50 minutes for 1 epoch to complete using NASNet Large architecture and the dataset of 106017 training and 26562 validation samples. It was preferred over AWS instace, as it is easier to set up. The template for the instance contained all the necessary tools, so no additional installation were required. Google Cloud was also tried before using Paperspace, but requesting a GPU instance was not successful in any region, as all the GPUs are booked.

Chapter 4

Methodology

In this chapter all the key decisions taken during the work on the thesis are described. Where to get enough training data? Is the data good enough? How to train? What neural network architecture to choose as a base model? What parameters to change? How to improve the accuracy? How to avoid overfitting? All these questions will be answered in the next few sections.

4.1 Data

With the popularity of machine learning and neural networks in particular the amount of available datasets significantly increased. From one side more people are interested in using and analysing different sources of data. From the other side the modern technology allows to collect and store huge amounts of data. The problems start to rise, when a labeled dataset for a specific task is required. There are MS COCO, PASCAL VOC, ImageNet, CIFAR datasets that contain huge amounts of labeled data for general-purpose object detection. However, the problem this thesis is focusing on is classifying cars. There are not so many datasets that contain labeled cars.

Two such datasets were found to satisfy the requirement of car classification: Stanford Cars dataset and Comprehensive Cars. Both contain labels for car mark, model and year. The Stanford dataset was collected and labeled by Stanford's university researchers, so it is populated by the cars used in USA. Some of them are specific for USA market, such as Acura or Plymouth. Another dataset - Comprehensive Cars (CompCars) - is collected by researchers from Chinese Academy of Sciences, Shenzhen, China. There are plenty of cars that are China specific and cannot be seen in European countries, such as Shouwang, Yingzhi or HongQi.

As the datasets were collected and labeled by different people, the structure is different. In order to utilize the two datasets as a whole, when training a car

classifier, it is required to merge both datasets into one that can be simply loaded into Keras. Stanford Cars dataset has all the images in one folder, each image with its own ID. The images are mapped with labels in a separate annotation file. Comprehensive Cars dataset is much bigger. It keeps the data in separated into folders. First level is folders of car marks, each car mark contains folders of models inside. Every model has year folders inside it, only then the actual images of cars can be found. See the example on *Figure 4.1*.

Figure 4.1: Tree structure of directories and images inside the CompCars dataset. This is just a sample, there are more images in each folder.

In order to map the car labels with IDs and merge two datasets into one, Python scripts were written. At first it was decided to try the classification for marks only, without any models or years. The scripts ran through all the files in both datasets and changed the path of images according to this pattern: "data/car_mark/img.jpg". This way a "data" folder was obtained. It contains folders of car marks inside. Each car mark folder already contains corresponding images of cars. *Figure 4.2* illustrates the new structure and *Figure 4.3* shows some image examples from the merged dataset.

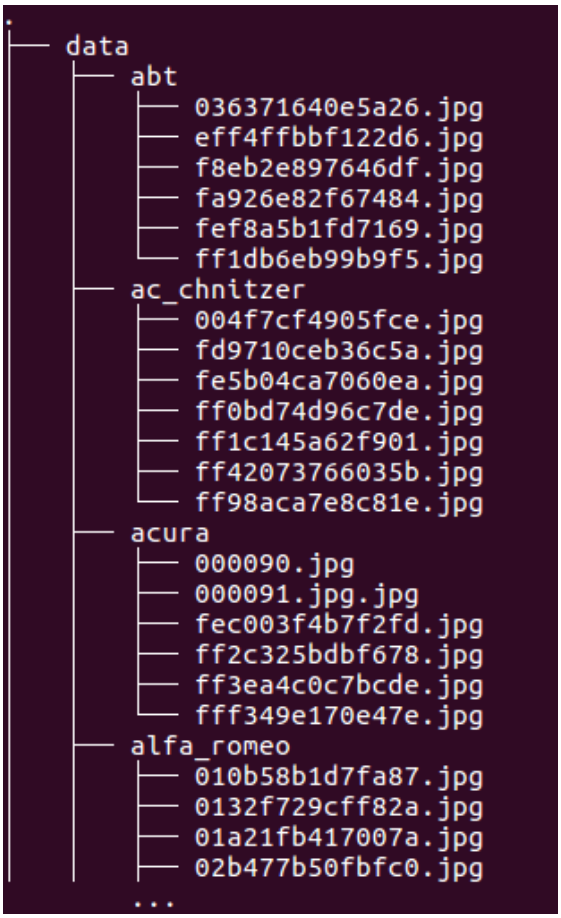

Figure 4.2: Tree structure of directories and images inside the merged dataset. This is just a sample, there are more images in each folder.

After running the Python scripts for merging the datasets, manual re-labeling still had to be done. As the datasets were collected by different people, there exist differences in labeling the same car. For example, in one dataset the label for Land-Rover is "land-rover", but in the other - "land_rover". Due to this minor difference the images of Land-Rover were distributed between two folders instead of being merged into one. One more example is "chevy" vs "chevrolet". Another issue that was discovered is human error. Some typos were found in car labels. For example, in CompCars there is a folder named "bwm" that contains images of BMWs. One more example is "lamorghini" vs "lamborghini". These typos were discovered after multiple training and data cleaning operations.

Before actual data processing all the data should be reviewed and cleaned. The dataset contains too many car marks. There are exotic Lamborghinis and Bugattis, rare and custom built Spykers, non-existent in Europe Shouwangs and Acuras. These sorts of classes were recognized as unnecessary and removed from the dataset for the purpose of better training. In China some car manufacturers produce copies of European cars that look almost exactly the same, but are actually dif-

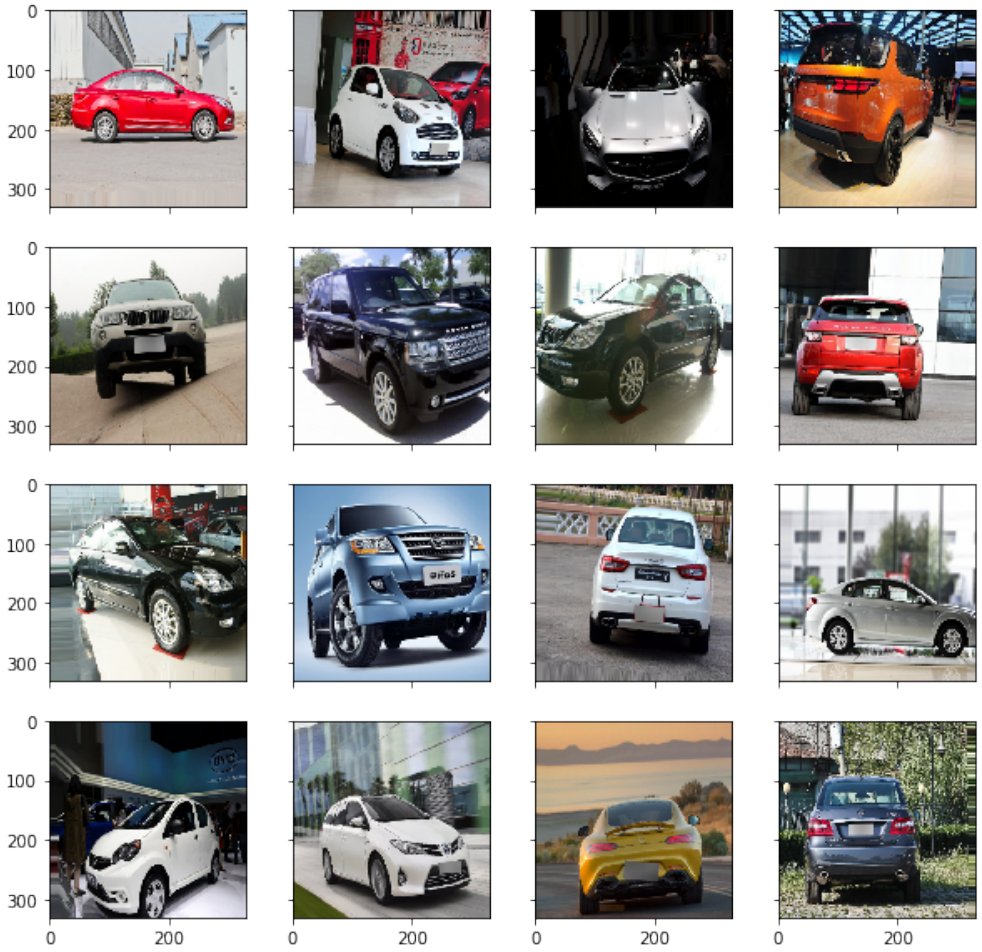

Figure 4.3: Example images from the merged dataset. Images are pre-processed and re-scaled according to the input requirements of a particular network architecture.

ferent marks. The images that look very similar, but should be recognized as different car marks, were also removed from the dataset, as it is highly unlikely the model would have learned the difference. Another criteria for removing images from the dataset was that the amount of images to represent a particular class was insufficient. Car marks containing less than 100 images were removed. Not all the criteria were defined right away. Some were added because of unsuccessful model training.

After all the manipulations with reformatting and cleaning the data are finished, the dataset can be divided into train and validation parts and then loaded into Keras. Another Python script is written to split the data into train and validation sets. 20% of the dataset is the validation part and the rest is for training. Since the data is already formatted, as shown on *Figure 4.2*, the path to the "data" folder can be directly given to the Keras' ImageDataGenerator.

4.2 Selecting Neural Network architecture

Keras framework provides multiple neural network architectures already built-in. The pre-trained on ImageNet weights for every architecture are also available. Initial experiments were done to compare simpler architectures and their performance. Next architectures were tested: VGG16, VGG19, Inception-v3, Xception. ResNet50 architecture has more complex architecture with more parameters and requires more time for training each epoch. This architecture should clearly outperform VGG, Inception and Xception networks, but it has to be confirmed experimentally. The initial tests were launched on the portion of data - only images from Stanford dataset. Moreover, only car marks are considered as classes, no model, year or any other information is given. There are 49 classes in the dataset. 12929 images are in the training set and 3256 images are in validation dataset. The experiment was launched 3 times with epoch numbers of 5, 10 and 30 epochs. All the architectures have the same top layer containing one fully-connected layer and the Softmax activation. The loss is set to Categorical Crossentropy and the optimizer is Adam [21] with learning rate of 0.0001. The initial results are documented in the *Table 4.1* below.

Network	Epochs	Validation accuracy	Validation loss
VGG16	5	10.93%	3.4696
VGG19	5	10.90%	3.4735
Inception	5	11.36%	3.3881
Xception	5	11.85%	3.3795
ResNet50	5	14.97%	3.2165
VGG16	10	11.36%	3.4083
VGG19	10	11.11%	3.4330
Inception	10	13.30%	3.2457
Xception	10	15.03%	3.2508
ResNet50	10	19.66%	2.9797
VGG16	30	12.47%	3.3491
VGG19	30	11.60%	3.3486
Inception	30	19.32%	2.9516
Xception	30	22.90%	2.9139
ResNet50	30	33.55%	2.4502

Table 4.1: Comparison between different architectures on Stanford dataset in order to select few architectures to proceed with.

As it can be seen from the *Table 4.1*, the ResNet50 does, indeed, outperform the rest of the networks, giving the best accuracy score on validation data. VGG network does not seem to improve significantly, but maybe more epochs can improve the situation. Both Inception and Xception networks work relatively good,

but also require more training to learn the car marks. So the same experiment was launched again with 50 epochs for each architecture. Additionally, 2 more network architectures are added to the experiment: DenseNet-121 and NASNet Mobile. These are the networks that were recently added to Keras, as the papers describing them were published in 2017. DenseNet-121 is the simplest DenseNet architecture and NASNet Mobile is the simplified version of NASNet Large architecture. The results for 50 epochs are given in the *Table 4.2*.

Network	Epochs	Validation accuracy	Validation loss
VGG16	50	14.07%	3.2863
VGG19	50	12.84%	3.3278
Inception	50	17.75%	3.5635
Xception	50	18.92%	3.3939
ResNet50	50	1.82%	4.1341
DenseNet121	50	21.54%	3.1317
NASNetMobile	50	13.15%	3.2809

Table 4.2: Comparison between different architectures with 50 epochs each on Stanford dataset in order to select few architectures to proceed with.

The results show that VGG networks still did not improve significantly. Xception and Inception networks gave slightly worse results with additional 20 epochs of training, which means something is wrong with the training process and possibly with data itself. ResNet50 seems to have overfitted, which is expected given the complexity of architecture and the amount of data it is trained with. NASNet and DenseNet both have shown good results. Inception, Xception, DenseNet and NASNet architectures are selected for further experiments.

Since Large version of NASNet architecture was not tested yet, it is now time to compare its performance with Mobile version and DenseNet, which is the best performing model so far. Next table shows the results of that comparison.

Network	Epochs	Validation accuracy	Validation loss
DenseNet121	50	21.54%	3.1317
NASNetLarge	50	22.31%	2.7120
NASNetMobile	50	13.15%	3.2809

Table 4.3: Comparison between DenseNet and NASNet architectures with 50 epochs each on Stanford dataset.

As NASNetLarge and DenseNet are both complex architectures, it is decided to proceed with only one of them and NASNet seemed to produce better results. Now the architectures for further training on the whole dataset and different optimization techniques are selected: Inception, Xception, NASNet Mobile and NASNet

Large. The last one is used just to see, what results are possible, both the rest presumably can be used in real-time video processing.

After the neural network architectures are selected for further experiments it is time to launch the training not only on Stranford's dataset, but on the whole merged dataset. Top layer is still the same with one fully-connected layer and Softmax activation, optimizer is Adam and loss is Categorical Crossentropy. No parameter adjustment is done yet, using the default parameters. Both NASNet architectures and Xception were trained for 50 epochs. Inception is left out of this experiment, as its results are comparable to Xception, and as training at this point is done on MSI Trident's GPU, it allows to save some computational time. In the *Table 4.4* the results are provided.

Network	Epochs	Accuracy	Loss	Validation accuracy	Validation loss
Xception	50	22.46%	3.1852	10.92%	4.3662
NASNetLarge	50	24.92%	2.9847	11.84%	3.6592
NASNetMobile	50	17.14%	3.3620	7.36%	4.1037

Table 4.4: Comparison between Xception and NASNet architectures with 50 epochs each on merged dataset.

The results are not good. Expected validation accuracy is around 60%-70%. At this point, as multiple different architectures are used, it is necessary to adjust the parameters for each architecture. Also at this point the merged dataset was reviewed the second time and all the Chinese brands were removed in order to decrease the number of classes.

4.3 Adjusting parameters

Since the results obtained in the initial experiment both on Stanford's and merged datasets were not satisfying, the ways of improving the results have to be tested. One approach is to look in more details, how the training of each epoch is happening. How does training accuracy converge or not converge to validation accuracy after multiple epochs. The desired graph should look like on *Figure 4.4*.

However, the results of training the car classificator look like on *Figure 4.5*. The model simply does not learn. Current learning rate parameter is 0.0001. Perhaps, this value of learning rate is too small. To ensure the problem is caused by too low learning rate, next training experiments were launched with learning rate of 0.1 and 0.001. First value is too big, so the overfitting should be visible during the first 3-4 epochs. The next value - 0.001 - is actually the default learning rate for all the optimizers in Keras.

Here is an example (*Figure 4.6*) of training with learning rate of 0.1. This value

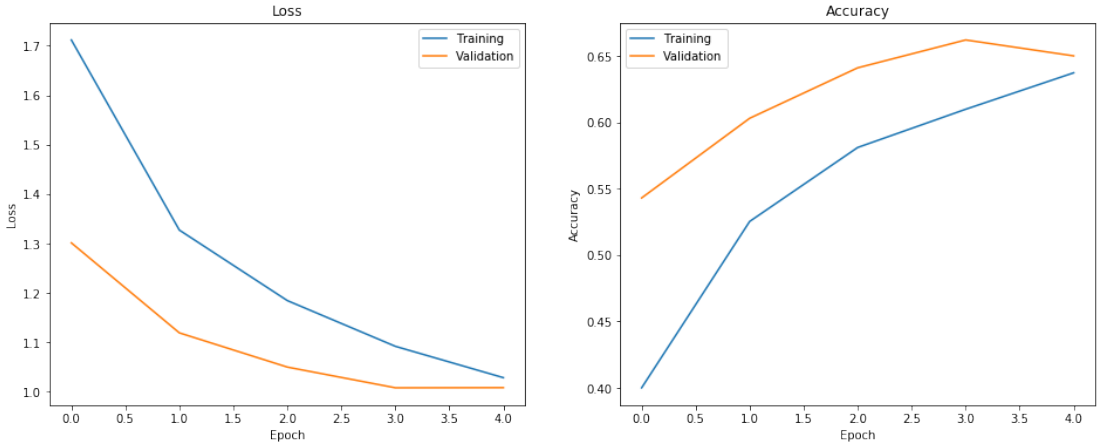

Figure 4.4: An example graph representing good training. After 4 epochs the loss on both training and validation data converges to 1 and the accuracy reaches 63%.

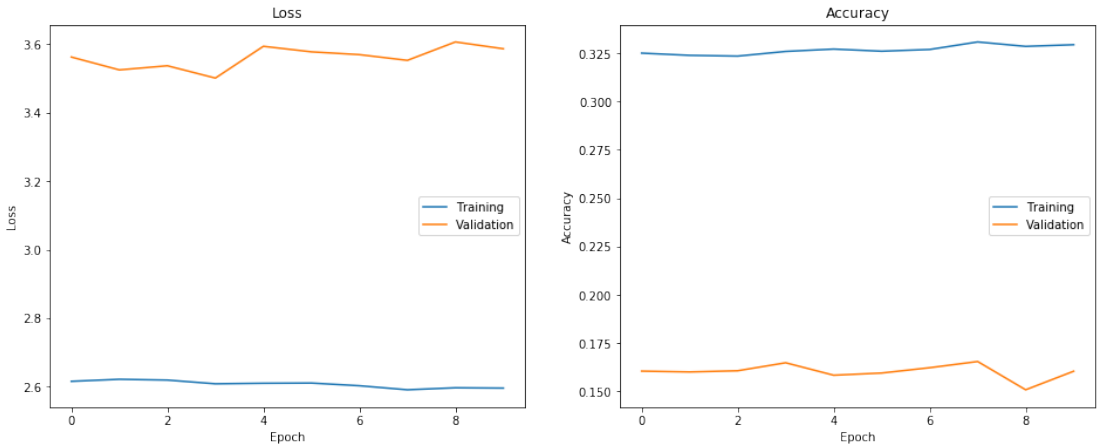

Figure 4.5: Actual results of training the neural network for 10 epochs without significant improvements in accuracy or loss.

is too big to use even on Standford's dataset, as the model tries to learn too fast, resulting in overfitting and sudden jumps in loss and accuracy back and forth. It was assumed that learning rate of 0.1 is too big before, so the training was only done for 4 epochs.

With **learning rate** of 0.001 it was possible to see the accuracy on training set growing significantly faster than with learning rate of 0.0001, but not too fast (jumping to 55% accuracy, while validation accuracy does not grow), as it was with learning rate of 0.1. The learning rate of 0.01 was also tested for a small number of epochs and it produced overfitting as well.

The best learning rate for Stanford's dataset appears to be 0.001. Following table (*Table 4.5*) illustrates the results for 10 epochs for selected architectures, where

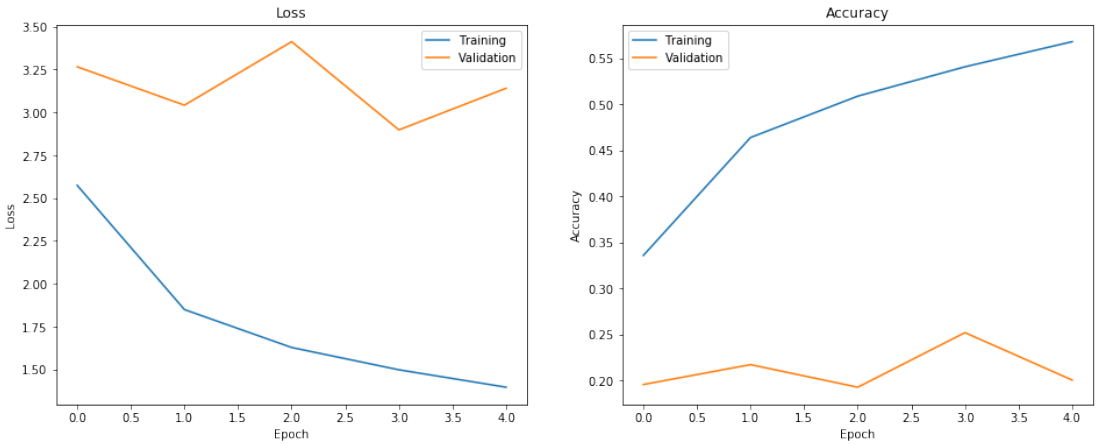

Figure 4.6: Example of training, when learning rate is too big - 0.1.

learning rate equals 0.001. The tests were done on Stanford's dataset to get the results faster and proceed with adjusted parameters on the merged dataset.

Network	Epochs	Accuracy	Loss	Validation accuracy	Validation loss
Inception	10	34.30%	2.3613	10.86%	3.6041
Xception	10	31.72%	2.5583	8.33%	3.7386
NASNetLarge	10	40.43%	2.0882	23.18%	2.6770
NASNetMobile	10	32.37%	2.4072	14.97%	3.3357

Table 4.5: Comparison between DenseNet and NASNet architectures with 50 epochs each on Stanford dataset.

After choosing bigger learning rate of 0.001, compared to previously used 0.0001, the results obtained with only 10 epochs of training are outperforming the results from training for 50 epochs on the same Stanford's dataset. NASNet Large previously had 22.31% accuracy on the validation data after 50 epochs, now after 10 epochs the result is already 23.18%. However, some overfitting seems to still be present, when looking at Inception and Xception results. For now it is not a problem, as the final model will be trained on merged dataset, which has significantly more data to train with.

Batch size was also configured at this point. The initial batch size was set to 16, however the GPUs used for training are powerful enough to process bigger batches at the same time, thus speeding up the process of training. The maximum batch size for Xception, Inception and NASNet Mobile on MSI Trident's GPU that did not cause memory exceptions is 48. For NASNet Large it is slightly smaller - 32 - as this architecture has bigger size of input images - 331 by 331 pixels. The batch size was also configured for Rocket Cluster's GPU, although even with batch size of 32 for Xception and Inception architectures it threw memory exceptions. Paperspace's GPU was able to process Inception, Xception and Mobile

architectures with the batch size of 64 and NASNet Large with the batch size of 48.

With adjusted learning rate and batch size parameters training for selected architectures, starting with NASNet Mobile was launched again on the whole merged dataset for 50 epochs. However, after training 50-th epoch has finished and the results were reviewed, the training process for other models was interrupted. When training on merged dataset of 106017 images in the training set and 26562 images in the validation set (with 125 distinct classes), the NASNet Mobile model achieved validation accuracy of 6.66%. Before that training the same model only on Stanford's dataset gave the validation accuracy of almost 15%. It was expected that given more data, the accuracy will increase, not decrease.

4.4 Input pre-processing

As the models are not training correctly, deep code debugging was performed. Existing open-sourced solutions were investigated and compared to the own solution. It was discovered that the image pre-processing function, which is used on all the images to format them before feeding into neural networks as an input, re-scales the images on its own. However, in the code the separate re-scaling was also performed using ImageDataGenerator, when reading the images from the folders, resulting in double re-scaling. The input to the neural networks was not correct, hence the problems with the training. Moreover, it was discovered that Inception and Xception, having similar architectures, expect the images re-scaled to -1.0 - 1.0 range, instead of 0-255 range. NASNet architecture expects images re-scaled to 0.0 - 1.0 range. The models were built and trained with such input formats in mind, so during model fine-tuning it has to be taken into account. After correcting the inputs to the networks, NASNet models were trained on Stanford's dataset for 5 epochs each to see, if there are any improvements. NASNet Large achieved 42.82% accuracy on training set and 22.95% accuracy on validation. The losses are respectively 1.9081 and 2.7672. NASNet Mobile achieved 29.78% accuracy on trainng set and 15.16% accuracy on validation set. The respective losses are 2.4745 and 3.2770. The results for only 5 epochs with correct inputs are already comparable with the results of training for 50 epochs in *Table 4.3*.

4.5 Solving overfitting problem

When training a neural network it is very important to avoid overfitting as well as underfitting. The overfitting problem was already mentioned in previous sections. It means that the neural network, especially when trained on a small dataset, learns very specific fearures and basically memorizes the training dataset achieving very good accuracy. When later tested with new images, the neural network in

this case cannot predict the correct class, as the generalization of the class was not created correctly.

The simplest way to fix the overfitting problem is by adding more data to the training. Since the overfitting was achieved on Stanford's dataset, which is only small part of the whole merged dataset, adding more data should not be a problem. However, with adding more images of cars to the existing classes, more car mark classes are also added, so the amount of classes grows from 49 to 125 car marks. The data should be balanced properly, otherwise it might cause problems for neural network in learning correct classes.

In order to fix the overfitting issue on Stanford's dataset, different techniques were applied. First thing is to have more data, so the data augmentation was applied. It means that every picture in the training dataset is slightly modified on each epoch by shifting the image by a certain coefficient, rotating it, flipping horizontally. The objects on the pictures stay the same, but the position and view angle slightly changes. It allows neural network to extract slightly different features, while still classifying the image to the same class.

Another thing that could help with overfitting is adding Dropout layers. Dropout with 0.25 parameter was tested on Stanford's dataset. It means the quarter of the features are dropped, when getting to the Dropout layer, so only 75% of features are passed forward. Together with data augmentation the Dropout helped to decrease the the between the accuracy on training set and accuracy on validation set. Previously the worst results on Stanford's dataset were with over 70% accuracy on training, while having less than 20% accuracy on validation. Now the gap is a lot smaller: the accuracy on training set is around 35%, while the accuracy on validation set is over 20%. These experiments were done with Stanford's dataset. Both data augmentation and Dropout were added to the training, when using merge dataset. Dropout was changed to 0.1 not to cause underfitting.

4.6 Further improvements

While investigating and debugging the code, interesting techniques were discovered for fine-tuning the neural networks. The discovered modifications was decided to apply in terms of car classificator as well.

From the logs of training on the merged dataset it was seen before that at some point neural network, for example NASNet Large, achieves 20%-25% validation accuracy, but it does not go further. Previously described modification to prevent overfitting do not allow the accuracy on training set to grow too much, when validation accuracy does not grow, so the training just caps with training accuracy of around 35%-40% and the validation accuracy of 20%-25% depending on the architecture. To solve the issue, more trainable parameters are introduces into

the top level of the model architecture. On top of every architecture additional fully-connected layer of size 128 with ReLU activation followed by Batch Normalization and Dropout are added before the final fully-connected layer with Softmax activation.

Different optimizers for different models are used in order to improve the training performance. Xception and Inception experimentally proved to produce slightly better results, when using Adam optimizer, compared to Stochastic Gradient Descent (SGD) or RMSprop [36]. NASNet architecture, however, uses RMSprop optimizer, as proposed in the original paper.

Moreover, the learning rate is not changed from constant to changing value. For Inception and Xception architectures the learning rate for the first 5 epochs was set to 0.01, then changed to 0.001 and the model was trained for 20 epochs and the last 10 epochs were trained with learning rate of 0.0001. For NASNet architectures the learning rate decay was given to the RMSprop optimizer as a parameter and not changed manually after said amount of epochs. Last few epochs were trained with unfreezing all the layers in the given architectures (only for Inception and Xception) and training all the layers of the network.

The number of epochs was changing from model to model, as the training was performed interactively. The weights after each epoch were compared to the previous weights and the best weights were saved. After training for a certain amount of epochs was finished, the best weights were loaded and the further training for certain amount of epochs was launched. With such approach it was easier to see, if the model is still training, starts to overfit/underfit, if the code contains some errors, if the results make sense. Next chapter gives an overview of the final experiment setup as well as the results achieved by the time of submitting the thesis.

Chapter 5

Experimental Results

The final setup of the experiment includes lost of improvements compared to the initial experiments with the purpose of increasing the validation accuracy. The selected 4 neural network architectures are still the same: Inception-V3, Xception, NASNet Mobile and NASNet Large. Each of the models has following layers on top of the original model: fully-connected layer of size 128 and with ReLU activation, Batch Normalization layer after that, Dropout layer with the dropout amount of 0.2, final fully-connected layer with the Softmax activation.

Learning rate parameter is changing during the training. For Xception and Inception the strategy of changing the learning rate is based on manual overview of the training process. The learning rate of 0.01 is applied for the first 5 epochs only, as then it starts to cause overfitting. Then the main training is done for 20 more epochs with learning rate of 0.001. If the accuracy still grows, then additional 10 epochs are trained with the same learning rate. If the accuracy seems to have reached the plato, then the learning rate is changed to 0.0001 and trained for further 10 epochs. A few times the training was interrupted due to some technical problems, so the weights of the last successful training were loaded and the training process was relaunched. The exact amount of the epochs trained is hard to say because of that reason. Inception and Xception networks finished the training with 3 epochs with all the layers unfreezed.

Learning rate for both NASNet architectures was set up to be changing over time another way. Default parameters of the RMSprop optimizer were changed in order to account for the desired technique. Starting learning rate for the NASNet was also set to 0.01. Learning decay was set to 0.00004, which is applied after each epoch. Rho and Epsilon parameters of the optimizer are both set to 0.9. NASNet architecture is too ambiguous to re-train the weights of all the layers.

The dataset for training and validation is the merged dataset that contains data from both Stanford and CompCars datasets. Cleaning of the dataset was already performed a few times in order to reduce the amount of classes and to fix the

issues with incorrect or inconsistent labels. The merged dataset only contains car marks as classes, the information about models and years is not processed at this point of research. Moreover, the information about car models requires heavy modifications, as there are lots of model names that contain mark name and include some metadata, such as car type (eg. hatchback or sedan) or car submodel (eg. standard, performance, hybrid). For processing the models as well it is needed to modify most of the labels. The merged dataset is containing 125 car mark classes and has 106017 images in the training set and 26562 images in the validation set. The split ratio between training and validation set is 80/20.

The best results achieved with every architecture after training for 40-50 epochs are described in the *Table 5.1.* It can be seen from the table that the accuracy of Inception and Xception networks is very similar, as it was expected from the beginning. Although, it was expected for Inception to slightly outperform Inception, but it happened vice versa. NASNet was expected to show a lot better results. Due to the amount of layers in NASNet architecture it was not possible to save weights of the model after each epoch as checkpoints the same way it was done with Inception and Xception. The weights have to be saved manually, thus some of the weights were lost, if some kind of error appeared before the saving method was called.

Network	Loss	Top 1 Accuracy	Top 5 Accuracy
Inception	3.5352	17.97%	46.35%
Xception	3.5351	17.16%	44.59%
NASNetLarge	3.9664	9.68%	35.60%
NASNetMobile	4.7133	1.33%	12.16%

Table 5.1: Final results of Inception, Xception, NASNet Large and NASNet Mobile network architectures after improved training on the whole merged dataset.

The results for Xception and Inception architectures have definitely improved from the initial experiment (*Table 4.4*). The Xception model's accuracy on validation set was 10.92%, now it increased to 17.16% with top 5 accuracy of 44.59%. The results of NASNet architectures became significantly worse, but most likely it is because of the code error or missing weights. The model was not re-trained due to the lack of time.

However, the accuracy of 60%-70% was still not achieved. There are 2 possible explanations: the data is not suitable or the car classification based on marks cannot be learned that easily, as a single car mark contains too many different car models and types inside - really hard to find general unique features to distinguish marks.

After the final results were obtained, the data was reviewed again. The number of classes was reduced from 125 to 73, as all the Chinese cars were removed, as most

of them are just copies of European cars that look almost the same, but have to be labeled differently. Cars with less than 100 images were also removed. A few more labels were merged together into one label, as there was a typo or the car makers were labeled differently in 2 datasets (eg. Chevy vs. Chevrolet). From the classes some models were removed, as they were very rare or unique cars, sometimes only concepts that were never produced (eg. Audi Urban). New dataset contains 91692 images in training set and 22960 in test set. The training after final cleaning of the dataset was not performed due to the lack of time. With further training it will be possible to say, if the messy data was affecting the training results that much.

Another problem is more interesting. Perhaps, it is not a trivial task for a neural network to learn the features of particular car marks. The variety of cars inside one mark is huge. There are different models and submodels, depending on the year of manufacturing cars look different. There are sedans, hatchbacks, convertibles, SUVs, pickups - all of which can belong to a single car mark. The assumption to start with car marks first might be wrong. It may be possible that a neural network can learn different car models much faster and from that predict the car marks. Every car model has more distinguishable features than every mark.

Chapter 6

Conclusion

The thesis was focused on the high-value target detection, where high-value targets were defined as different cars. For example, it was one of the goals to recognize BMW car mark and its model X5 from a real-life video footage to signal an alarm to the human operator. The classification algorithm implementation includes training a neural network. Unfortunately, the results of training were not good to be actually applicable to the task described. Further work on modeling the classificator is required.

Despite the poor classification accuracy, a lot of work was done. Multiple network architectures were tested, the parameters and techniques for fine-tuning a neural network were researched and experimentally tested. The working methods are preserved and documented, the methods that produced worse results are ruled out. Questions such as "what network to use as a basis?", "what pre-processing methods to use?", "how to better clean up the data to achieve meaningful results?" have their detailed answers. Some of the modifications, such as cleaning up the dataset, are also fulfilled already. The lack of powerful enough GPU resources throughout the testing period has significantly slowed down the training, hence the mismatch between expected and actual results was discovered too late to conduct more training experiments.

A special comment deserves the data the neural networks were trained with. The dataset was not good, required multiple clean up procedures to finally get something working and may still not satisfy the requirements for good training. Now it is clear that there is no better dataset, than the one prepared by own hands and tailored for specific needs. Open-sourced datasets are not big enough, require filtering, restructuring and cleaning. It is a good idea to make an automated system for collecting more data in a desired format, so the balanced dataset with all the classes labeled properly and consistently is available.

Next steps would be to:

1. Train the data with the last version of merged dataset (79 car mark classes).
2. Train the networks using car models as classes - requires a lot of re-labeling in the current dataset or collecting a new dataset.
3. Investigate, if the model trained to distinguish models can also classify car marks.
4. Try CNN-RNN architecture or hierarchical softmax function to combine classification of car marks and models into a single network architecture.

Bibliography

[1] Youtube statistics for 2018. `https://fortunelords.com/youtube-statistics/`.

[2] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788, 2016.

[3] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. *arXiv preprint arXiv:1612.08242*, 2016.

[4] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, January 2015.

[5] Serge Belongie Lubomir Bourdev Ross Girshick James Hays Pietro Perona Deva Ramanan C. Lawrence Zitnick Piotr Dollár Tsung-Yi Lin, Michael Maire. Microsoft coco: Common objects in context. *arXiv arXiv:1405.0312*, 2014.

[6] Ross Girshick. Fast r-cnn. *arXiv preprint arXiv:1504.08083*, 2015.

[7] Sasha Targ, Diogo Almeida, and Kevin Lyman. Resnet in resnet: generalizing residual architectures. *arXiv preprint arXiv:1603.08029*, 2016.

[8] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[9] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[10] Ejaz Ahmed, Michael Jones, and Tim K Marks. An improved deep learning architecture for person re-identification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3908–3916, 2015.

[11] RT Lee, KC Hung, and HS Wang. Real time vehicle license plate recognition based on 2d haar discrete wavelet transform. *International Journal of Scientific & Engineering Research*, 3(4):1–6, 2012.

[12] Hongye Liu, Yonghong Tian, Yaowei Yang, Lu Pang, and Tiejun Huang. Deep relative distance learning: Tell the difference between similar vehicles. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2167–2175, 2016.

[13] Florian Chabot, Mohamed Chaouch, Jaonary Rabarisoa, Céline Teulière, and Thierry Chateau. Deep manta: A coarse-to-fine many-task network for joint 2d and 3d vehicle analysis from monocular image. In *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.(CVPR)*, pages 2040–2049, 2017.

[14] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2172–2180, 2016.

[15] Xiaodan Liang, Liang Lin, Xiaohui Shen, Jiashi Feng, Shuicheng Yan, and Eric P Xing. Interpretable structure-evolving lstm. In *Proc. CVPR*, pages 2175–2184, 2017.

[16] Jiang Wang, Yi Yang, Junhua Mao, Zhiheng Huang, Chang Huang, and Wei Xu. Cnn-rnn: A unified framework for multi-label image classification. In *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, pages 2285–2294. IEEE, 2016.

[17] Balázs Nagy and Csaba Benedek. 3d cnn based phantom object removing from mobile laser scanning data. In *Neural Networks (IJCNN), 2017 International Joint Conference on*, pages 4429–4435. IEEE, 2017.

[18] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[19] Joshua Owoyemi and Koichi Hashimoto. Learning human motion intention with 3d convolutional neural network. In *Mechatronics and Automation (ICMA), 2017 IEEE International Conference on*, pages 1810–1815. IEEE, 2017.

[20] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 922–928. IEEE, 2015.

[21] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[22] Linjie Yang, Ping Luo, Chen Change Loy, and Xiaoou Tang. A large-scale car dataset for fine-grained categorization and verification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3973–3981, 2015.

[23] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 554–561, 2013.

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

[25] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[26] François Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint*, 2016.

[27] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[28] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2017.

[29] Keras documentation page. `https://keras.io`.

[30] Tensorflow homepage. `https://www.tensorflow.org`.

[31] Ranking of deep learning libraries 2017. ://blog.thedataincubator.com/2017/10/ranking-popular-deep-learning-libraries-for-data-science/.

[32] Cuda documentation page. `https://docs.nvidia.com/cuda/`.

[33] Jupyter homepage. `http://jupyter.org`.

[34] Rocket cluster homepage. `https://www.hpc.ut.ee/en_US/web/guest/rocket-cluster`.

[35] Paperspace homepage. `https://www.paperspace.com`.

[36] T Tieleman and G Hinton. Divide the gradient by a running average of its recent magnitude. coursera: Neural networks for machine learning. Technical report, Technical Report. Available online: https://zh. coursera. org/learn/neuralnetworks/lecture/YQHki/rmsprop-divide-the-gradient-by-a-running-average-of-its-recent-magnitude (accessed on 21 April 2017).

All the web references were valid on 21.05.2018.

License

Non-exclusive license to reproduce thesis and make thesis public

I, Anton Prokopov (date of birth: 14.03.1993),

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

   1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

   1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

   "High-value target detection", supervised by Gholamreza Anbarjafari,

2. am aware of the fact that the author retains these rights.

3. certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, 21.05.2018

Source Code

As this thesis is a project requested by Mooncascade, source code is not publicly available. To receive the source code, please contact the author via the following e-mail: prokopov@hotmail.com