Department of Mathematics

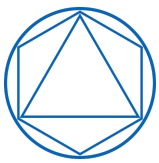
Technische Universität München

TUM Data Innovation Lab

Driver Distraction Detection with Deep Learning

Nikolay Kadrileev	Michael Schuster	Marco Both
nikolay.kadrileev@tum.de	m.schuster@tum.de	marco.both@tum.de

Mentors:	M.Sc. Simone Dari, Dr. Benny Kneissl
Project Lead:	Dr. Ricardo Acevedo Cabra
Supervisor:	Prof. Dr. Massimo Fornasier



Abstract

Driver distraction is one of the main causes of vehicle crashes and therefore detrimental for driving safety. Finding ways to prevent crashes caused by a distracted driver would not only have an impact on lowering vehicle accidents, but also support the development of advanced driver-assistance systems and autonomous driving. In this work we first discuss the decisions that went into creating our dataset using video data provided by BMW AG. This dataset is then used to train a convolutional neural network which is able to classify the head position of the driver. Furthermore, we investigate whether it is feasible to include publicly available videos that contain different camera angles to train the network. The results indicate that consistent camera placement is of significant importance for the performance of our network.

Contents

List of Figures

List of Tables

1 Introduction

One of the major causes for accidents in the driving environment, especially on the highway scenario, is driver distraction. Although the automotive industry is already on its way to make self-driving cars a reality, the human driver will still be the backup vehicle operator in potentially unclear or specific situations. This is valid up until SAE's J3016 level 3 of driving automation, as seen in Figure 1.1, which appoints the human driver as the fallback performance of the dynamic driving task.

SAE level	Name	Narrative Definition	Execution of Steering and Acceleration/ Deceleration	*Monitoring* of Driving Environment	Fallback Performance of *Dynamic Driving Task*	System Capability *(Driving Modes)*
***Human driver* monitors the driving environment**						
0	**No Automation**	the full-time performance by the *human driver* of all aspects of the *dynamic driving task*, even when enhanced by warning or intervention systems	Human driver	Human driver	Human driver	n/a
1	**Driver Assistance**	the *driving mode*-specific execution by a driver assistance system of either steering or acceleration/deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	Human driver and system	Human driver	Human driver	Some driving modes
2	**Partial Automation**	the *driving mode*-specific execution by one or more driver assistance systems of both steering and acceleration/ deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	**System**	Human driver	Human driver	Some driving modes
***Automated driving system* ("system") monitors the driving environment**						
3	**Conditional Automation**	the *driving mode*-specific performance by an *automated driving system* of all aspects of the dynamic driving task with the expectation that the *human driver* will respond appropriately to a *request to intervene*	System	**System**	Human driver	Some driving modes
4	**High Automation**	the *driving mode*-specific performance by an automated driving system of all aspects of the *dynamic driving task*, even if a *human driver* does not respond appropriately to a *request to intervene*	System	System	**System**	Some driving modes
5	**Full Automation**	the full-time performance by an *automated driving system* of all aspects of the *dynamic driving task* under all roadway and environmental conditions that can be managed by a *human driver*	System	System	System	**All driving modes**

Figure 1.1: Summary of SAE International's levels of driving automation for on-road vehicles [1].

In this project, we seek to address a section of a driver's distraction detection utilizing deep learning on video data. This work is a collaboration between the Technische Universität München (TUM), Bayerische Motoren Werke Aktiengesellschaft (BMW AG), and the Leibniz-Rechenzentrum (LRZ).

The remainder of this document is organized as follow: continuing this introduction in Section 1, we will go over facts that motivated this work and discuss the problem statement that outlines the scope of the project. Next, we introduce our approach, explaining the employed procedure, projects decisions and challenges encountered along the way. Lastly,

we present our system's results and wrap up the report with a conclusion and suggestions for future work.

To comply with privacy requirements some information of this work is omitted in this report and proprietary video data was censored.

1.1 Motivation

During the period from 2005 to 2007, the U.S. Department of Transportation conducted a survey to collect information about events leading up to vehicle accidents. The main focus was to collect data about the decisive event preceding the accident and attribute a cause (critical reason) for the crash.

Figure 1.2 shows how the causes of vehicle crashes were distributed between drivers, vehicles, and environment. In 94% of the cases, accidents were originated by human error.

Critical Reason Attributed to	Estimated	
	Number	Percentage* ± 95% conf. limits
Drivers	2,046,000	94% ±2.2%
Vehicles	44,000	2% ±0.7%
Environment	52,000	2% ±1.3%
Unknown Critical Reasons	47,000	2% ±1.4%
Total	2,189,000	100%

*Percentages are based on unrounded estimated frequencies
(Data Source: NMVCCS 2005–2007)

Figure 1.2: Driver, Vehicle, and Environment Related Critical Reasons [2].

To have a better insight on what is prompting driver mistakes, the study further classify driver-related reasons in Figure 1.3. Recognition error, which included driver's inattention, internal and external distractions, and inadequate surveillance, was the most frequently critical reason with 41% of the cases. Non-performance errors can also be attributed as driver distraction from the driving performance, including sleep, coughing, sneezing, etc. This sums up to almost 50% of all accidents caused by human drivers being related to driver distraction.

Similarly in Europe, with the intention of finding effective countermeasures to traffic crashes, a study was carried out concerning vehicle accidents. Once more, 52% of accidents caused by driver mistakes (referred to in this study as *temporary personal factor*), were a result of distracted and/or inattentive drivers. These facts are shown in Figure 1.4.

These studies characterize driver distraction as a substantial problem concerning driving safety worldwide. The task of detecting this driver distraction emerges as an essential

Critical Reason	Estimated (Based on 94% of the NMVCCS crashes)	
	Number	Percentage* ± 95% conf. limits
Recognition Error	845,000	41% ±2.2%
Decision Error	684,000	33% ±3.7%
Performance Error	210,000	11% ±2.7%
Non-Performance Error (sleep, etc.)	145,000	7% ±1.0%
Other	162,000	8% ±1.9%
Total	2,046,000	100%

*Percentages are based on unrounded estimated frequencies
(Data Source: NMVCCS 2005–2007)

Figure 1.3: Driver-Related Critical Reasons [2].

groundwork to solve the problem.

Temporary personal factor	Total	%
Fear	20	4%
Distraction	182	30%
Fatigue	67	12%
Inattention	123	22%
Under influence	101	17%
Stress	76	15%
Other	1	0%
Total factors	570	100%

Figure 1.4: Types of temporary personal factor [3].

Besides accidents prevention, dealing with driver distraction can also support the development of autonomous driving. SAE's level 3 of automation imposes that the human driver is expected to respond to a *request to intervene* from the automated driving system [2]. This request is a notification to the driver that he/she should promptly begin or resume control over the subject vehicle. Detecting a driver's distraction is important in this case to determine the human's response time, given different forms of distraction, thus allowing requests to be made in a suitable time.

1.2 Problem Statement

The main question raised is how to identify a driver's distraction. There are several factors that could be used as indicators for distraction. For instance, we could track whether the

driver has hands on the wheel, eye tracking could be performed, or smartphone usage could be captured.

To classify an action as a distraction, we also need to bring context to the current situation. Perhaps the usage of a smartphone while the vehicle is parked should not be classified as a distraction. A driver not looking through the front windshield could possibly be glancing over his left window to check if an intersection is clear for a turning maneuver. This translates the problem into a complex task that would need a complete system containing perception of the driver actions and the surrounding driving environment.

During the first phase of this project, the students together with their supervisors refined the scope of this work to only track head movement. We want to know in which direction the driver's head is pointing in a video image. Due to the complexity of the computer vision problem and powerful results of methods that rely on learning data representations, deep learning methods were used in this project.

BMW provided several hours of video data, which the students utilized to build a neural network to detect the driver's head movement. One important objective was to generalize this algorithm in a way that its application could be extended for publicly available video data.

The task was to train and evaluate a deep learning model while finding ways to improve accuracy and performance on unseen data. The final model was used on a completely new test video of BMW to evaluate how well the Deep Learning model generalizes on new data.

We had several ways to improve the results by experimenting with different components of our deep learning system. We tried two preprocessing methods, a couple of dataset splits, our network architecture with different optimizers, and trained our model with multiple hyperparameters. We applied a variety of approaches and we explain our procedure and results in the next sections.

2 Preprocessing

In this section we discuss the labeling and preprocessing techniques that were used to collect and prepare the data for the classification task.

2.1 Data Labeling

To train our neural network, we needed to prepare a dataset containing annotated videos that would serve as a base to training our network in each class. In this case, a class refers to a head direction. As mentioned in Section 1.2, identifying distraction can be a complicated task that depends on context. For this reason, we decided to utilize six classes to describe a set of head positions. A more extensive commentary on this decision and how data labeling influenced our model training is presented in Section 4.1.

To label our video data with class annotations, we used a labeling script. We opted to train our model on still frames. This script would play the subject video and allow the user to press a character in the keyboard (characters were assigned to each class) to label the current video frame. This frame would then be saved as an image inside a dataset folder for its specific class. This procedure allowed us to label over 11 thousand frames, in order to have enough diverse data. The same procedure is executed when the trained model is deployed. Still frames are captured from the video, classified by the network and then put back together as a video.

Along the project we observed that having a strict labeling criterion was crucial to produce a more accurate model. Next, we will describe our labeling principle for each class and provide a frame as an example:

1. **Straight:** For all classes, we are looking for specific references to interpret the driver's head position. For the *straight* class, we can see both cheekbones, but the nose is not pointing in the camera's direction.

2. **Slightly left:** The references we look for in the *slightly left* class are both cheekbones are clearly visible. Besides that, we also often have the front vertical part of the nose facing the camera.

3. **Left:** For the *left* class, we labeled frames where the head is completely turned to the left. Here we use the jawline as a reference. Also, notice that the driver's left cheek is not visible.

4. **Slightly right:** For this class, we check if it is possible to see the left jawline of the driver's face and if the right cheekbone is at least partially covered by the nose pointing slightly to the right.

Figure 2.1: Example for *straight* class.

Figure 2.2: Example for *slightly left* class.

Figure 2.3: Example for *left* class.

5. **Right:** This class is very similar to the *left* class. The references are the well-defined jawline and the absence of the driver's right cheek.

6. **Distracted:** Here distracted means any head position where the driver is not looking through the windshield or any side window. In Figure 2.6 the driver has the head pointing downwards. This can be easily seen by the direction where the nose and chin are pointing. Other common cases for this class would be looking completely down (in any side) or having the head turned to the back seat.

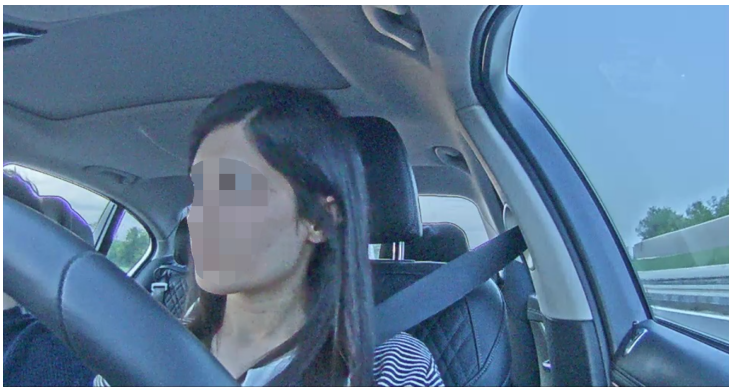


Figure 2.4: Example for *slightly right* class.

Figure 2.5: Example for *right* class.

Figure 2.6: Example for *Distracted* class.

2.2 Image Preprocessing

The script **preprocess_image.py** gives users a possibility to preprocess a single image. There are three flags that determine the preprocessing procedure that will be discussed throughout this section.

2.2.1 General preprocessing for the network

The *subtract_imagenet_mean* flag tells the program to subtract the ImageNet mean to replicate the preprocessing technique used during the training of the VGG16 network. Subtracting these mean values is crucial for the performance because we do not start the training from scratch but use the pretrained weights. Unfortunately, the mean values are not integer numbers, which our classification network expects as inputs. Therefore, we decided to perform the mean subtraction step inside of the neural network and set this flag to **False**.

The VGG16 image input size is 224x224 pixels. Therefore, we resize a face region as the final step of our image preprocessing algorithm. Moreover, this face region is always a square. In the following subsections, we will explain how to get these square areas. Of course, we could rescale and resize a rectangular area, but then we should always use the same scaling for the width and length of every image to preserve the proportions of a face, which is more difficult in the end.

2.2.2 Cropping

The *youtube_video* flag tells the network if the cropping of the top left part should be performed. This step should be used for the BMW videos because they consist of four camera views, namely, two interior and two exterior views. Due to privacy and data protection policy, we show only interior views in the report.

2.2.3 Finding face region

Using the *find_driver_face* flag, one chooses which method will be used to find a driver's face, and namely, cropping from the middle or finding a face using a face detector.

Here, we want to give a small comparison of both methods that assist in finding a face region. But why do not we take just the whole image for classification? The answer is that the input image contains too much information, which we actually do not need, e.g. background, or another person sitting in a passenger seat. Therefore, we want to help our network to focus on those features that really matter in finding a distracted driver.

Cropping from the middle

Cropping from the middle is a simple preprocessing method that returns a center part of an image, based on the smallest dimension between height and width. This results in a square image with the driver's head. An example of using this technique is presented in Figure 2.7 and Figure 2.8. The main drawback of cropping is its sensitivity to the camera position, e.g. for YouTube videos a driver almost never appears in the middle of the frame. Therefore, this method is only valid for one specific camera position. Another

problem is that we still have some background in the image. To tackle these problems, we created the second preprocessing method based on neural networks.

Figure 2.7: Input image.

Figure 2.8: Image after cropping top left and cropping from the middle.

Face detector

As we have already mentioned above, *Face detection* is another preprocessing method implemented in our program that returns a region of interest (ROI) containing a driver's face. This approach is based on the Single Short Detector Framework (SSD) with a ResNet base network. Weights and neural network architecture files in Caffe format are available on GitHub of OpenCV. After downloading these files, one have to put them into the preprocessing folder. The newest release of OpenCV makes it possible to read Caffe models and perform classification, detection or other tasks depending on the network functionality. The output of the face detection network is an array with rectangular regions and confidence levels for each region. Confidence level is a value between 0 and 1 showing the probability that a returned rectangle contains a face (Figure 2.9). Using a threshold, we can get rid of detections with low confidence levels. In our program, we ended up with a threshold value of 0.4, which is a good trade-off between false positive and false negative detections. This parameter can be changed in the **config.py** file. The returned rectangular region is further postprocessed, and namely, its smallest dimension is extended to be equal to the largest one. Therefore, an output is given as coordinates of the top left and bottom right corners of a square (Figure 2.10).

Figure 2.9: Rectangular regions containing faces with confidence levels of 98.88% and 98.26% for the left and right images respectively.

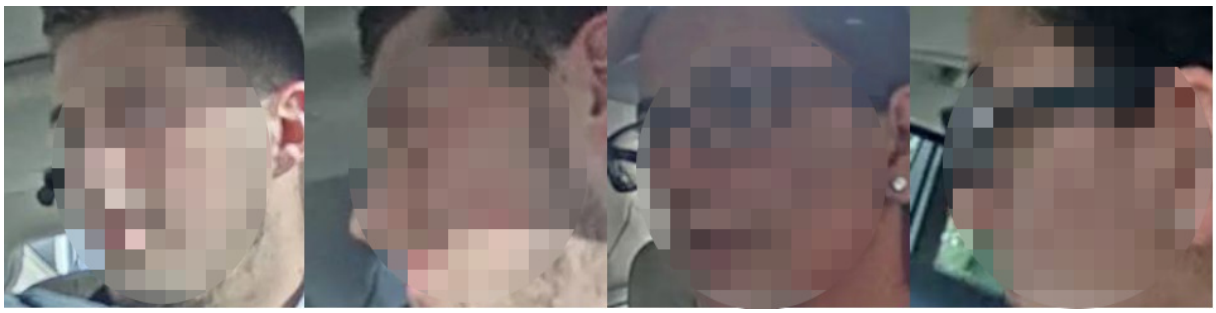


Figure 2.10: Images after face detection applied.

The main drawback of the face detection approach compared to the cropping technique is the problem of false negative detections. Especially for noisy pictures, e.g. night frames, the accuracy of the face detection algorithm drops significantly. Also, the algorithm fails in detecting a head from the back. Two examples of these situations are given in Figure 2.11). If there was no detection with confidence larger than the threshold, the **preprocess_image.py** script returns the variable **valid_image** as **False**, otherwise as **True**.

Figure 2.11: False negative detections.

While working with BMW videos, we almost always had a situation where only a driver's face was visible. Therefore, we took a rectangle with the largest confidence and returned it if the confidence level was larger than the threshold. Even for distracted frames with some part of a driver's face hidden, a passenger's face is still rather small to be detected with a large confidence level, as in the right part of Figure 2.12). Unfortunately, it is not the case for YouTube videos, which we added to our dataset during the 3rd Milestone. YouTube bloggers usually put their cameras in the middle of the car front panel. Hence, a passenger's face is detected with the same confidence level as a driver's face, as in the left part of Figure 2.12. Therefore, the script **preprocess_image.py** was changed and in

the current implementation it finds two region with faces that have the highest level of confidence. As a final result, the script returns the leftmost region for YouTube videos or the rightmost region for BMW videos depending on the value of the *youtube_video* flag. The result of this approach is presented in Figure 2.13. If only one face has been detected with a high confidence level, the script checks for its location in the image and returns the face region if only it is inside the middle or left part for YouTube videos and the middle or right part for BMW videos.

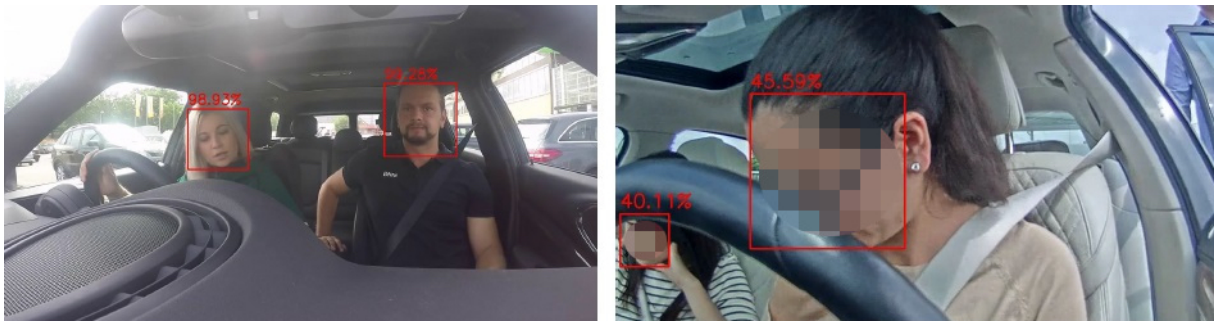


Figure 2.12: Passenger detection problem.

Figure 2.13: Solution to the passenger detection problem.

2.2.4 Dataset preprocessing

Until now, we have preprocessed and found a face region of a single image. To automate this process, we wrote another script **images_to_tfrecords.py**. It converts multiple images to a single tfrecords file that is used by the network for training. This script also calls a method from the **preprocess_image.py** script to crop and resize the image according to the requirements of the network. If the face detection is used, some detections can fail and do not appear in the tfrecords files. But these images are saved in a false_negative folder on the same path as the output tfrecords file and can be used further, e.g. for plotting ROC curves or calculating accuracy. This script also checks a name of the input image. If an image contains "youtube" in its name, then the script sets **True** value for the *youtube_video* flag while calling the **preprocess_image.py** script.

2.3 Dataset Splitting

Choosing a good or a bad dataset splitting strategy can have a huge influence on the overall performance of the neural network. Therefore, we spent quite a lot of time on analyzing videos we had and finding an efficient partitioning method.

2.3.1 Splitting by fraction

In the beginning, we used the simplest splitting approach, i.e., splitting by fraction. This method was implemented in the **split_dataset.py** script. This script splits the dataset in the training, validation and testing sets. Throughout the project, we used 80% of the dataset for training, 10% for validation and the rest 10% for testing. The script requires training and validation fractions as input arguments, which indicate how much data are used for train and validation respectively and the fraction of test data is given by $1-(train\_fraction+val\_fraction)$. Frames are drawn randomly from the whole dataset, i.e., the same person can appear in all subsets, especially if the number of input frames for that driver is large. Therefore, the network trains and evaluates its performance on identical drivers having the same features, e.g., clothes, hair, facial shapes and forms. And this is the main weakness of the splitting by fraction approach. Although we get almost perfect classification accuracy for both training and evaluation, the network struggles to classify a frame with a person it has not seen before.

Table 2.1: BMW Drivers

Person	Features	# of Photos
Male 1	brown hair, daylight	623
Male 2	bald, daylight, night time	700
Male 3	dark hair, daylight	2248
Male 4	dark hair, daylight, sunglasses	491
Male 5	blond hair, daylight	223
Male 6	bald, daylight	226
Male 7	blonde hair, daylight, simulated driving	256
Male 8	brown hair, daylight, simulated driving	109
Female 1	blonde hair, daylight, night time	2723
Female 2	brunette, daylight	1803
Female 3	brunette, daylight, glasses	377
Female 4	brunette, daylight, simulated driving	378

2.3.2 Splitting by person

The goal of our algorithm is to correctly classify an unknown driver. Therefore, we decided to split the dataset by person, e.g. if a person appears in the training dataset, then we put all other videos of this person solely in the same dataset. Moreover, to make the

splitting sets more diverse, we analyzed the BMW and YouTube videos and put videos with the same features in different sets. All drivers' features are given in Table 2.1 and Table 2.2.

Table 2.2: YouTube Drivers

Person	Features	# of Photos
Male 9	brunette, daylight, headband	32
Male 10	dark hair, night time, glasses	164
Female 5	blonde hair, daylight, headwear	338
Female 6	blonde hair, daylight	40
Female 7	brunette, daylight, sunglasses	32
Female 8	dark hair, daylight, hat	18
Female 9	blonde hair, daylight, hat	64
Female 10	brunette, daylight	20
Female 11	red hair, daylight, sunglasses	40
Female 12	brunette, daylight	29
Female 13	brunette, daylight, glasses	32
Female 14	blonde hair, daylight	118

2.3.3 Splitting overview for BMW and YouTube

To split the dataset by person, we created another script called **split_dataset_by_name.py**. This script partitions the dataset into training, validation and testing sets by names of input videos. It requires training and validation fractions, i.e., names of videos to be used for training and validation. One should use the space bar to add more than one video in a category. The rest is used for testing.

The BMW dataset was split by person as follows:

- Training, 7747 images (76%)
  - Female 1, Female 2, Male 3, Male 4, Male 6, Male 7
  - Features: daylight, night time, sunglasses
- Validation, 1410 images (14%)
  - Female 4, Male 2, Male 5, Male 8
  - Features: daylight, night time
- Testing, 1000 images (10%)
  - Female 3, Male 1
  - Features: daylight, glasses

The main drawback of the BMW dataset is a limited number of drivers and features. Although we tried to split the same features into different subsets and at the same time hold splitting fractions at a reasonable level, we did not get some important features in the testing set, e.g., night time frames nor a person wearing sunglasses.

Adding YouTube frames increased diversity of the data. The YouTube dataset was split by person to make new features appear in the subsets:

- Training, 496 images (54%)
  - Female 5, Female 6, Female 14
  - Features: daylight, night time, sunglasses
- Validation, 163 images (18%)
  - Female 7, Female 8, Female 10, Female 12, Female 13, Male 9
  - Features: daylight, hat, sunglasses, headband
- Testing, 268 images (28%)
  - Female 11, Female 9, Male 10
  - Features: daylight, night time, hat, sunglasses

The size of the YouTube dataset is 10 times smaller than the BMW dataset. Therefore, whatever fractions for subsets we choose, they will not significantly influence the overall proportion. But we have night frames and sunglasses in the testing set, and headwear in the testing and validation sets. In the combined BMW+Youtube Dataset, the training set consists of 74% of images, 14% belong to the validation set and the rest 12% of the data are included in the testing part.

3 Algorithm

We will now discuss the algorithm that we used to classify the images in our dataset. We decided to use a convolutional neural network due to its success in various image classification tasks [4]. We assume the reader is familiar with the way neural networks are working, in particular convolutional, max pooling and fully connected layers. If this is not the case, we suggest reading some literature (e.g. [5]) available on this topic before continuing with this paper.

3.1 Network architecture

For our experiments we chose the popular VGG 16 network due to its simple architecture and the possibility to download pretrained weights for it.
The input to the network is a 224x224x3 RGB image. It then gets passed through multiple blocks consisting of convolutional layers followed by a max pooling layer. All convolutional layers have a kernel size of 3x3, stride 1 and 1 pixel padding on each side, therefore preserving the width and height of the input. Every few convolution layers, a max pooling layer with size 2x2 and stride 2 is inserted [4]. The pooling layers help by not only reducing the dimension by 2 and therefore reducing the computational complexity, but also make the network less sensitive to small translations in the input [5]. After the convolutional and pooling layers there are two fully connected hidden layers followed by one fully connected output layer. The activation function is the rectifier function $f(x) = \max(0, x)$ [6] in all of the hidden layers (both convolutional and fully connected) and the identity function in the output layer.
We use the *softmax* function to map the real-valued output vector of our network to a vector whose values can be interpreted as the probability of our network assigning some example x to some class i. Let C be the number of neurons in the last layer, and therefore also the amount of classes we are training the network with. Now let $s = f(x; W) \in \mathbb{R}^C$ be the output of our network for input x and some weights W. Then the probability p_i of our network assigning class $i \in \{1, ..., C\}$ to the input x is given by

$$p_i = \frac{e^{s_i}}{\sum_{j=1}^{C} e^{s_j}} .$$

As $e^x > 0$ for all $x \in \mathbb{R}$ and $\sum_{i=1}^{C} p_i = \frac{\sum_{i=1}^{C} e^{s_i}}{\sum_{j=1}^{C} e^{s_j}} = 1$, we can see that the entries of p sum up to 1 and $p_i \in [0, 1]$ [7].
A visualization of the complete network architecture can be seen in Figure 3.1.

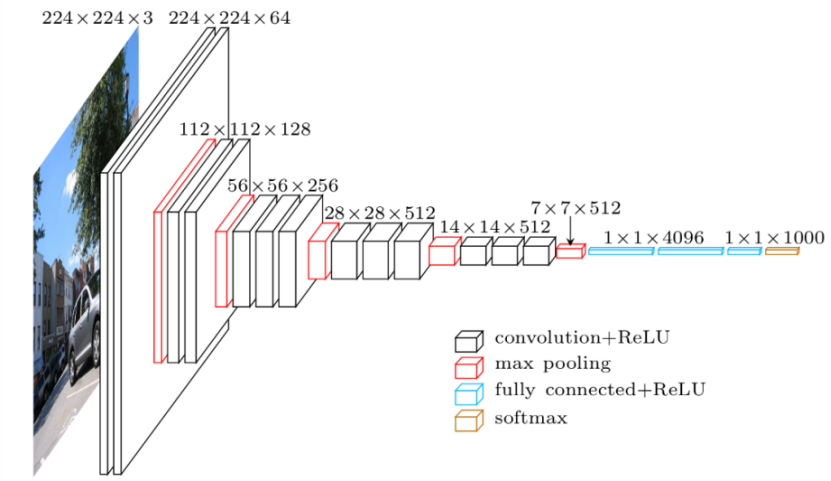


Figure 3.1: VGG 16 network architecture [8].

3.1.1 Different network architectures

In recent years, results have shown that increasing the depth (amount of layers) of a network leads to better performance, especially in visual recognition tasks [9]. However it gets harder to train these networks the more layers are added. He et al. suggest deep residual learning to deal with problems arising in very deep networks. Instead of learning the actual desired mapping $\mathcal{H}(x)$, stacked blocks of layers inside the network are made to fit another mapping $\mathcal{F}(x) := \mathcal{H}(x) - x$ [9]. They achieve this by introducing shortcut connections around these stacked layers that perform an identity mapping. Empirical results from their paper show that this is indeed easier to optimize and allows the training of much deeper networks.
As this network design outperformed the VGG 16 network on various image-related tasks, we experimented with the Resnet50 architecture for our dataset. However we found no difference in the results achieved by both architectures, so we decided to keep training with the VGG 16 network (results can be found in Appendix Table A.1).

3.2 Implementation

The implementation of our algorithm was done in Python using the deep learning framework TensorFlow [10]. We trained our network on a Nvidia Pascal P100 GPU provided to us by the Leibniz Supercomputing Centre of the Bavarian Academy of Sciences and Humanities. The best results were achieved after 80 epochs on average, which using this hardware took around 40 minutes. Classifying a single image took 7,1 milliseconds (68,8 milliseconds using face detection) on average on a GeForce GTX 1080 Ti GPU. Taking no other computations into account, our algorithm could classify 140 images (14 using face detection) per second.

4 Training

In this chapter we discuss different aspects of training our network. This includes if (and how) we should merge certain labels of our dataset, how to deal with the uneven data distribution and how we chose the hyperparameters used for training.

4.1 Grouping of labels

Before actually training our network, there were several options to use our dataset that would yield different interpretations of the results. In this section we discuss the approaches we decided to use and why we think they are beneficial.
When training using all six classes, additional data can be incorporated more easily to identify whether the driver is actually distracted. One of the shortcomings of this work is that the decision whether the driver is distracted or not is based solely on the head position. Turning the car left or right might for example also be a reason to turn the head to the side without actually being distracted. Sensor data from the car could be used together with the classified head position to improve the decision, which would not be as easy if the algorithm was only trained on two classes. Another benefit of training with six classes is that it helped us identify problems in our algorithm design and labeling process. After observing lots of misclassifications between certain classes, we defined stricter criteria for labeling and refined our dataset, which ended up increasing the accuracy significantly.
Training with two classes on the other hand also has some advantages. First of all, it is an easier problem than training with more classes so it should be easier to get satisfactory results. Furthermore, if not combining the results of six class classification with additional data, the results have to be merged anyway to classify whether a driver is distracted or not. In this case it would be more straightforward to train with only two classes from the beginning.
For the training with two classes, we tried combining the six classes in two different ways. The first alternative we tried was splitting it in two classes by considering the *straight* class as *not distracted*, and merging all the other ones together as *distracted*. The second alternative was merging *straight*, *slightly left* and *slightly right* as *not distracted*, and the remaining ones as *distracted*. While the first option requires the driver to look straight to the front to classify him as not distracted, the second option allows a little more freedom for the head to be slightly turned.
Results of both class merging approaches as well as training with six classes will be discussed in the next chapter.

4.2 Loss function

The cross entropy between two probability distributions p and q is given by

$$H(p, q) = \mathbb{E}_p\left[-\log\, q\right] = H(p) + D_{KL}\left(p \parallel q\right),$$

where $H(p)$ is the entropy of p and $D_{KL}\left(p \parallel q\right)$ is the Kullback–Leibler divergence (KL divergence) from q to p [5]. Minimizing the cross entropy with respect to q is equivalent to minimizing the KL divergence because the entropy does not depend on q. As the KL divergence measures the distance between two probability distributions, we can interpret the optimization in our network as finding an ouput distribution that approximates the real label distribution as good as possible.
Using the cross entropy between the true labels and the predicted probabilites of our network we obtain the following loss function:

$$\mathcal{L}(W) = -\sum_{n=1}^{N}\sum_{c=1}^{C} y_{nc}\,\log\, p_{nc}\,,$$

where N is the number of training examples, C the number of classes, $p_n \in \mathbb{R}^C$ contains the softmax predictions of the network and $y_n \in \mathbb{R}^C$ is a one-hot vector containing a 1 at the index of the correct class and zeros otherwise [7].
When starting to create the dataset, we labeled multiple classes to keep open options for the future, however we were still planning to train on two classes. Therefore, and also due to the drivers looking straight for a majority of the frames in our videos, we ended up with the data distribution described in section 2.1. The dataset is approximately uniformly distributed when training on two classes (*straight* and the combination of all other classes), but heavily unbalanced when training with all six individual classes.
When training with six classes we therefore tried cost sensitive learning, which outperformed other methods dealing with an imbalanced dataset in [11]. This was done by adapting the loss function to assign a different cost per example depending on the inverse class frequency in the dataset, resulting in a higher loss for misclassified examples in a minority class than for misclassified examples in a majority class.
Let t_n be the true class for training example n. The modified loss function is then given by

$$\mathcal{L}(W) = -\sum_{n=1}^{N} \frac{1}{classFrequency(t_n)} \sum_{c=1}^{C} y_{nc}\,\log\, p_{nc} \quad \text{with}$$

$$classFrequency(c) = \frac{\#\text{training examples with class c}}{N}\,.$$

The algorithm made only few mistakes even for classes with few examples, and our experiments using this modified loss function lead to no improvements (see Appendix Table A.2 for details). We therefore decided to use the standard cross entropy loss for simplicity.

4.3 Optimizer

Due to the high dimensionality of the parameter space in our problem, higher order optimization methods are not that well suited. We therefore decided to use the popular Adam Optimizer to train our algorithm, which has been empirically shown to outperform other first-order optimization algorithms on various tasks. It is a first order optimization method that uses both the first and second moment of the gradient to compute individual adaptive learning rates per parameter [12]. The update formula of the optimization algorithm is given by

$$x_t = x_{t-1} - \alpha \cdot \frac{m_t}{1-\beta_1^t} \Big/ \left(\sqrt{\frac{v_t}{1-\beta_2^t}} + \epsilon \right)$$

with α being the learning rate, β_1 and β_2 the exponential decay rates for the first and second oder gradient moments, ϵ some small constant to avoid division by zero,
$v_t = \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot \nabla_x f(x_{t-1}) \odot \nabla_x f(x_{t-1})$ where $\odot$ refers to the elementwise multiplication, and $m_t = \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot \nabla_x f(x_{t-1})$ [12].

Loshchilov and Hutter discovered a problem in the way most modern Deep Learning frameworks introduce weight decay in a combination with the Adam Optimizer. This implementation leads to a decrease in effectiveness of weight decay [13]. We ran experiments with both Adam and Stochastic Gradient Descent (SGD), and even with this implementation issue Adam seemed to achieve similar results while converging faster than SGD (results can be found in Appendix Table A.3). We therefore decided to keep training using the Adam Optimizer.

4.4 Transfer learning

Empirical results have shown that weights in the early layers of convolutional neural networks are not specific to the dataset they were trained on, but rather general and used to extract features like edges or colors in the image [14]. Due to this observation and our small data set making it hard to train the whole network from scratch, we decided to initialize our network with pretrained weights on ImageNet [15]. To achieve a similar data distribution as the data that was used for training the pretrained weights, we decided to follow the same preprocessing steps, mainly subtracting the mean RGB value of the ImageNet training set.
Yosinski et al. show that it is important to choose the right amount of layers that have their weights transferred [14]. Both this and deciding whether to fine-tune or freeze the transferred layers impacts the performance of the network. Due to our dataset being relatively small and the huge amount of parameters in the fully connected layers of the VGG network, we decided to transfer the weights for all 13 convolutional layers from the pretrained ImageNet weights. We then ran the following experiments:

1. Keep all the convolutional layers frozen for the whole training duration.

2. Start training the convolutional layers immediately. Due to a high initial learning rate this probably destroys the carefully tuned pretrained weights immediately.

3. Start training the convolutional layers after some epochs.

Not training the convolutional layers at all worked best, so we ran our remaining experiments using this setting (results can be found in Appendix Table A.4).

4.5 Network hyperparameters

There were several other hyperparamters that we had to choose for training our network, including the learning rate, exponential decay rates β_1 and β_2 for Adam, batch size and how to regularize the network using weight decay [16] and dropout [17]. Due to time and computing power constraints, we kept some of the hyperparameters at default values that seem to work well on a variety of problems. We then performed a grid search over the remaining hyperparamters and used our validation set to evaluate the generalization performance [5]. Section 5.2 will discuss the results of the best models evaluated on our test set.

5 Experiments

The last two chapters discussed the architecture of our network and some details on training it. We will now explain different performance measures for our algorithm and then use them to present and evaluate the results achieved over the course of this project.

5.1 Performance measures

In this section we discuss the different measures we used to evaluate the performance and compare different versions of our algorithm.
When using face detection as a preprocessing step, we decided to count all images where no face was detected as classified incorrectly. Depending on the application of the algorithm one could however also classify them as *distracted* or use more sophisticated approaches that we will discuss in chapter 6.

Let N be the total number of images including the ones where no face was detected, C the number of classes. For $i \in \{1, ..., C\}$, TP_i refers to the number of images correctly classified as class i, FP_i refers to the number of images incorrectly classified as class i and FN_i refers to the number of images of true class i that were incorrectly classified.

- **Accuracy** is the overall percentage of images that our algorithm classified as the correct head position [18].

$$Accuracy = \frac{\sum_{i=1}^{C} TP_i}{N}$$

- **Precision** describes what percentage of images classified as class i are classified correctly [18].

$$Precision_i = \frac{TP_i}{TP_i + FP_i}$$

- For some class i, the percentage of true images in that class classified correctly is called **Recall** [18].

$$Recall_i = \frac{TP_i}{TP_i + FN_i}$$

Furthermore we used confusion matrices and receiver operating characteristic curves (ROC curves) to visualize the results and identify pairs of classes that our algorithm was struggling with. The ROC curve can be plotted using values calculated from the confusion matrix at different decision tresholds, in particular the recall and the false positive rate [19]. The area under the curve (AUC) of the ROC curve can then be used to measure

the performance of the algorithm [18].
While the accuracy allows us to compare the overall ability of our algorithm to classify the direction in which the driver is looking, precision and recall give more insights into what kind of errors the algorithm is making. The values reported for both precision and recall of our algorithm are for the class *not distracted*. As the algorithm might be used in safety-critical applications, high precision is preferred over high recall. The vehicle might only display a warning if the driver is incorrectly classified as distracted, failing to warn a distracted driver classified as not distracted however might have more severe consequences.

5.2 Results

Since we tried a lot of different methods and parameters, as described in previous sections, we had a great number of accuracy, precision values, and confusion matrices for each test, including for hyperparameters tuning. Because of that, we present in Table 5.1 a summary of our best results for the most prominent changes. We present which dataset was paired to which processing method and choice for a number of classes. We show their resulting accuracy rates and a final accuracy to better express how well methods using face detection performed after their false negative rate (FN rate). Next, we discuss some important details and present our best result in Table 5.2.

We learned that using few different driver faces would cause overfitting and consequently a bad generalization for unseen video data. Having several frames for a same driver on a dataset can result in repetitive data, which do not provide knowledge of head direction to our model. We solved this problem by adding more drivers, as described in section 2, and making sure that the model would be evaluated and tested against complete new video data. Not only a different video but an unknown driver. That way we could train the network to achieve the best generalization for our application.

Labeling consistently was a quite difficult task. We had to revise our labeled dataset, especially after our network exposed inconsistencies in our manual labels. The stricter our labeling criteria, the better our model became. We believe there is still room for improvement. A new dataset revision could be able to solve our struggles between classes *straight* and *slightly right*. This problem between these two classes is presented in Figure 5.1. The worse classification performance are 51 frames belonging to the *straight* class are wrongly classified as *slightly right*.

The usage of a face detection network in preprocessing had a substancial impact in improving the accuracy of our model as seen in Table 5.1. Although we added complexity to our preprocessing procedure, this project decision was very beneficial. Besides improved performance on the BMW dataset, face detection was the main reason for the good performance of our model in public available videos. We achieved lower accuracies in all models that involved public videos, primarily due to different camera angles.

Table 5.1: Results for different approaches

Dataset	Preprocessing	Splitting	FN Rate	Acc	Final Acc
BMW	Cropping	straight vs. rest	-	0.85	0.85
BMW	Cropping	straight + slightly	-	0.83	0.83
BMW	Cropping	6 classes	-	0.83	0.83
BMW	Cropping	straight + slightly	0.02	0.84	0.82
BMW	Face Detection	straight vs. rest	0.02	0.90	0.88
BMW	Face Detection	6 classes	0.02	0.89	0.88
Revised BMW	Face Detection	6 classes	0.02	0.92	0.90
Youtube + BMW	Face Detection	6 classes	0.039	0.85	0.82
Youtube + BMW	Face Detection	straight + slightly	0.039	0.92	0.88
Youtube + BMW	Face Detection	straight vs. rest	0.039	0.87	0.84

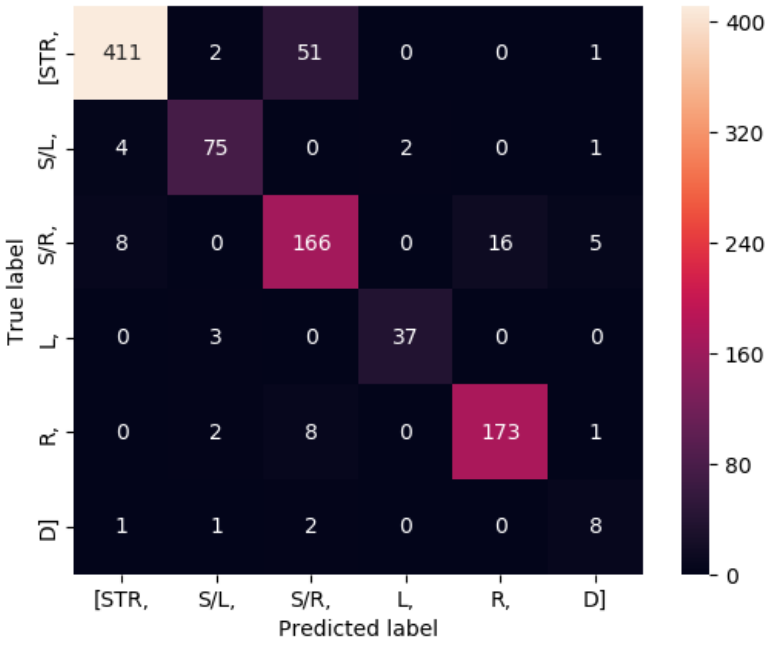


Figure 5.1: Confusion matrix for the Revised BMW dataset using Face detection and trained with 6 classes.

Table 5.2: Best result details.

Dataset	BMW Revised
Preprocessing	Face detection
Number of classes	6
Batch size	64
Learning rate	0.0001
Learning decay rate	0.9
Learning rate decay steps	1000
Weight decay rate	0.05
Testing accuracy	0.92
Total testing accuracy	0.90
Precision "straight"	0.97
Recall "straight"	0.88

6 Conclusion and Future Works

In this work, we presented an algorithm to identify a driver's head movement using a neural network with deep learning. We found that to achieve a good accuracy and precision, several adjustments were needed from data selection and splitting to network layer setup and hyperparameter tuning. Nonetheless, we achieved satisfactory results (especially with BMW video data) and proved that our concept can be applied to public videos with a worsened performance (around 8% difference as seen in Table 5.1).

Still, the success of the network's prediction relies on a positive face detection in preprocessing. For future works, alternatives could be implemented to process frames selected as false negatives. One possible solution could be, for instance, apply our earlier model that uses the *cropping* preprocessing method, since it already had an accurate performance.

To improve the network's accuracy with BMW videos, the usage of an attention heatmap (or class activation maps) could help understanding why our model is struggling between *straight* and *slightly right* head positions. For a better performance with public available videos, an even dataset containing the same amount of BMW and public video frames could be used for training. This would mean more examples for slightly different camera positions and new driver faces.

Bibliography

[1] *Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems*, Society of Automotive Engineers International, January 2014.

[2] *Critical Reasons for Crashes Investigated in the National Motor Vehicle Crash Causation Survey*, National Highway Traffic Safety Administration, 1200 New Jersey Avenue SE., Washington, DC 20590, March 2018, published by NHTSA's National Center for Statistics and Analysis.

[3] P. Thomas, A. Morris, R. Talbot, and H. Fagerlind, "Identifying the causes of road crashes in europe," in *Annals of Advances in Automotive Medicine*, vol. 57. Association for the Advancement of Automotive Medicine, September 2013, pp. 13–22.

[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[5] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[7] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[8] (2016) Vgg 16 architecture. [Online]. Available: https://heuritech.files.wordpress.com/2016/02/vgg16.png?w=940

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[10] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: a system for large-scale machine learning." in *OSDI*, vol. 16, 2016, pp. 265–283.

[11] M. Buda, A. Maki, and M. A. Mazurowski, "A systematic study of the class imbalance problem in convolutional neural networks," *CoRR*, vol. abs/1710.05381, 2017. [Online]. Available: http://arxiv.org/abs/1710.05381

[12] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[13] I. Loshchilov and F. Hutter, "Fixing weight decay regularization in adam," *CoRR*, vol. abs/1711.05101, 2017. [Online]. Available: http://arxiv.org/abs/1711.05101

[14] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Advances in neural information processing systems*, 2014, pp. 3320–3328.

[15] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. Ieee, 2009, pp. 248–255.

[16] A. Krogh and J. A. Hertz, "A simple weight decay can improve generalization," in *Advances in neural information processing systems*, 1992, pp. 950–957.

[17] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[18] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information Processing & Management*, vol. 45, no. 4, pp. 427 – 437, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0306457309000259

[19] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 233–240.

A Appendix

Table A.1: Comparison of VGG 16 and Resnet50. All experiments were performed on the BMW dataset without face detection, the labels combined into two classes (*straight* and the rest).

Network	Learning rate	Weight decay	Accuracy
VGG 16	0.0001	0.0005	0.85
VGG 16	0.001	0.0005	0.89
VGG 16	0.001	0.005	0.88
Resnet50	0.0001	0.0005	0.83
Resnet50	0.001	0.0005	0.89
Resnet50	0.001	0.005	0.84

Table A.2: Comparison of training with and without class weights. All experiments were performed on the BMW dataset without face detection using all six classes.

Class weights	Learning rate	Accuracy
No	0.0001	0.78
No	0.001	0.77
Yes	0.0001	76
Yes	0.001	74

Table A.3: Comparison of Adam and SGD. All experiments were performed on the BMW dataset without face detection, the labels combined into two classes (*straight* and the rest).

Optimizer	Learning rate	Weight decay	Accuracy
Adam	0.0001	0.0005	0.85
Adam	0.001	0.0005	0.89
Adam	0.01	0.0005	0.83
SGD	0.0001	0.0005	0.82
SGD	0.001	0.0005	0.85
SGD	0.01	0.0005	0.83

Table A.4: Transfer learning comparison. All experiments were performed on the BMW dataset without face detection, the labels combined into two classes (*straight* and the rest).

Convolutional layers	Learning rate	Weight decay	Accuracy
Frozen	0.0001	0.0005	0.85
Frozen	0.001	0.0005	0.89
Frozen	0.001	0.005	0.88
Frozen for first 15 epochs	0.0001	0.0005	0.88
Frozen for first 15 epochs	0.001	0.0005	0.88
Frozen for first 15 epochs	0.001	0.005	0.81
Trained from beginning	0.0001	0.0005	0.80
Trained from beginning	0.001	0.0005	0.88
Trained from beginning	0.001	0.005	0.73

Table A.5: Accuracy results for the first milestone. We trained the dataset with 2 and 4 classes for 5 different weight decays.

Weight decay	2 Classes	4 Classes
0.0005	0.85	0.74
0.01	0.83	0.72
0.1	0.82	0.74
1	0.82	0.72
10	0.73	0.59

Table A.6: Accuracy results for the second milestone. We trained the dataset with 6 classes for 5 different weight decays employing the cropping preprocessing method.

Weight decay	6 Classes
0.0005	0.78
0.01	0.71
0.1	0.77
1	0.68
10	0.68

Table A.7: Accuracy results for the second milestone. We trained the dataset with 6 classes for 5 different weight decays employing the face detection preprocessing method.

Weight decay	6 Classes
0.0005	0.82
0.01	0.85
0.1	0.85
1	0.82
10	0.82

Table A.8: Accuracy results for the second milestone. We trained the Youtube dataset with 6 classes for 5 different weight decays employing the cropping and face detection preprocessing methods. Here we display the best result out of all 5 weight decays.

Cropping	**Face detection**
0.35	0.78

Table A.9: Accuracy results for the second milestone. We trained the BMW dataset with two different splits for 2 classes for 5 different weight decays employing the cropping and face detection preprocessing methods. Here we display the best result out of all 5 weight decays.

2 Classes split	**Cropping**	**Face detection**
straight vs rest	0.84	0.90
straight+slightly vs rest	0.83	0.84