University of West Bohemia

Faculty of Applied Sciences

Department of Cybernetics

MASTER THESIS

PLZEŇ, 2020 ROBERT BRADA

Zde bude Originál zadání DP podepsaný děkanem fakulty

Declaration

I hereby declare that this master's thesis is completely my own work and that I used only the cited sources.

Pilsen, March 29, 2020

....

Robert Brada

Acknowledgements

I would like to express my gratitude and appreciation to Ing. Miroslav Hlaváč, Ph.D. and Ing. Marek Hrúz, Ph.D. who provided me with the support and resources necessary for the implementation of the system presented in this work.

Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum provided under the program "Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015042), is also greatly appreciated.

Abstract

The goal of this thesis is to design and implement an end-to-end Driver Attention Warning system that automatically detects distracted driver based on visual data provided by an onboard camera. The system is built using convolutional neural networks using *Python* programming language and *Keras* library. Different neural network architectures like VGG16, InceptionV3, ResNeXt50, EfficientNetB0 and EfficientNetB4 are implemented, modified, optimized and combined in order to achieve the best results.

Keywords

distracted driver detection, convolutional neural networks, classification, Keras

Contents

Part I

Introduction

Driver's distraction plays a major role when it comes to car accidents. According to the U.S. Department of Transportation [1], there were 3,166 people killed in motor vehicle crashes involving distracted drivers in the United States, which is equivalent to 9% of overall fatalities. The development of systems used to detect distracted drivers can help to prevent such accidents. These systems are often referred to as Driver Attention Warning (DAW) systems or Driving Monitoring and Assistance Systems (DMAS).

The objective of DAW is to keep an eye on the driving status of a driver and detect if the driver is paying attention to the road conditions or if the driver is doing something else and thus is distracted. Such systems assist drivers by easing their control efforts, reinforcing their sensing power, warning them in case of mistake, and so on [2].

The basic task is to detect if the driver is distracted from safe driving or not (simple yes/no answer). NTHSA [3] describes distracted driving as *"any activity that diverts the attention of the driver from the task of driving"*. There are more types of distraction and the basic categories can be defined as *manual distraction*, *cognitive distraction* and *visual distraction* [4].

Cognitive distraction occurs when a driver diverts his or her attention to another mentally demanding task. Talking on a hands-free cell phone and using a voice-activated electronic system are two activities that produce almost purely cognitive distraction. Being lost in thoughts or daydreaming are also examples of cognitive distraction [5]. Visual distraction includes distraction when *"driver's eyes are off the road"*, which can happen because of inattention, sleepiness, fatigue or drowsiness. Manual distraction includes situations when the driver takes his hands of the wheel. This includes activities such as drinking, eating, operating the radio, using cell phones, smoking, etc.

To decide about the driver's distraction, DAW has to collect some data and then make a decision based on this data. Data can be gathered from multiple kinds of sources. One source of data can be visual data from a camera, which is the data source chosen in this work. Data can also be obtained from biophysical sensors (e.g. EEG signal, skin temperature, etc.). Another source of data are the vehicle's inside and outside sensors. These are sensors monitoring speed, acceleration, braking and turning dynamics, sound, etc. Sensors in the vehicle's seat can monitor body posture, while sensors in the steering wheel can monitor hand posture. Data from these sensors are collected, processed and then the decision about driver's distraction is made [2].

Usage of a cellphone was identified as a major cause of distraction [3], thus some DAW systems were built on the principle of detecting a cellphone. A system built by Zhang *et al.* was using a camera mounted above the dashboard and the cellphone was detected using Hidden Conditional Random Fields (HCRF) model [6]. Seshadri *et al.* [7] created their own dataset for cell phone usage detection. The authors used the Supervised Descent Method (SDM) and Histogram of Oriented Gradients (HOG) features in combination with various classifiers. Their system achieved 93.9% classification accuracy.

Recent progress in the field of computer vision that was caused by the use of *Convolutional Neural Networks* (CNN, ConvNets) enabled building more accurate DAW systems. In 2018 Baheti *et al.* [8] used VGG16 neural network and achieved 96.31 % accuracy using a dataset created by Abouelnaga *et al.* [9].

The goal of this thesis is to analyze and get familiar with techniques used to build DAW systems and then suggest a method to create a custom DAW system using solely visual data. The chosen method is to build an end-to-end DAW system using modern CNN architectures such as InceptionNet [10], ResNeXt [11] and EfficientNet [12]. The system will rely solely on visual data obtained from a dashboard camera. The input to this end-to-end system is an image of a driver and the output is a specific type of driver's distraction. The distractions will be classified into one of the following categories: *1. safe driving, 2. texting with the left hand, 3. texting with the right hand, 4. talking on a phone with the left hand, 5. talking on a phone* *with the right hand, 6. talking to a passenger, 7. drinking, 8. adjusting hair or putting on make-up, 9. reaching behind, 10. operating the radio.* To build such a system an appropriate dataset to train and evaluate the neural networks has to be obtained.

Part II

Theory

This part describes the methods and principles used to build the Driver Attention Warning system that is part of this thesis. First, the classification task is defined. Next, a simple neuron and neural network are described. Finally, principles of convolutional neural networks and implemented architectures are described.

1 Classification

In machine learning, the problem of classification is a task of categorizing data instances into a predefined set of classes. Algorithm that implements classification is called *classifier*. The most common types of classification are [13]:

- **Binary classification**
  Input data are assigned to either one of two classes (e.g. email *is* or *is not* a spam)
- **Multi-class classification**
  Data instances are categorized into more than two classes, but only one class can be assigned to each instance.
- **Multi-label classification**
  Data instances are categorized into more than two classes and more classes can be assigned to one instance.

The Driver Attention Warning system that is part of this thesis is a *multi-class classification* system, because the goal is to categorize driver's action into one of the predefined set of classes (driving safe, texting, drinking, operating radio, *etc.*). There are more algorithms that are commonly used to perform a classification task and each of them has its own pros and cons in terms of complexity, efficiency, and accuracy. Very commonly used are the following learning algorithms:

- Support Vector Machines
- Artificial Neural Networks
- K-Nearest Neighbors
- Decision Trees
- Bayes classifier

2 Artificial Neural Networks

Artificial Neural Networks (ANN) are algorithms that are inspired by biological neural networks. Using ANNs is in these days very popular and in many cases the most successful approach to take when solving a wide variety of problems such as *classification, regression, image segmentation, object detection* and many more. ANNs are broadly used in many different areas like finance, medicine, advertising, natural language processing, and security applications just to name a few.

A big advantage of ANNs compared to other approaches is that ANNs can learn a lot of important deciding features by itself just by providing the network desired output for a particular input.

2.1 Neuron

The core building block of an ANN is a neuron. A neuron is a function which takes an input x and processes it in the following manner:

$$y = f(x) = f(\sum_{n=1}^{N} w_i x_i + b), \tag{1}$$

where x is the input vector and y is the output number. N is the dimension of the input vector, w_i is a weight associated with input x_i, b is bias term and f is called *activation function.* See Fig. 1 for an illustration.

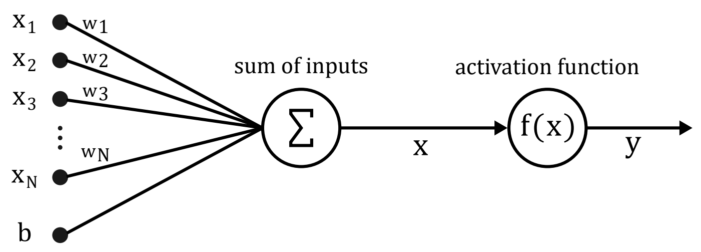


Figure 1: Scheme of an artificial neuron.

2.2 Simple Neural Network

Neurons are usually stacked into layers which means that the output of one neuron is the input to all neurons in the following layer (besides the last layer). Every neural network consists of an *input layer* and an *output layer*. Neural networks usually have also so-called *hidden layers* and in that case we call such architecture *deep neural network* (DNN). Each layer can have a different number of neurons and neurons in each layer can use different activation functions. See Fig. 2 for an illustration.

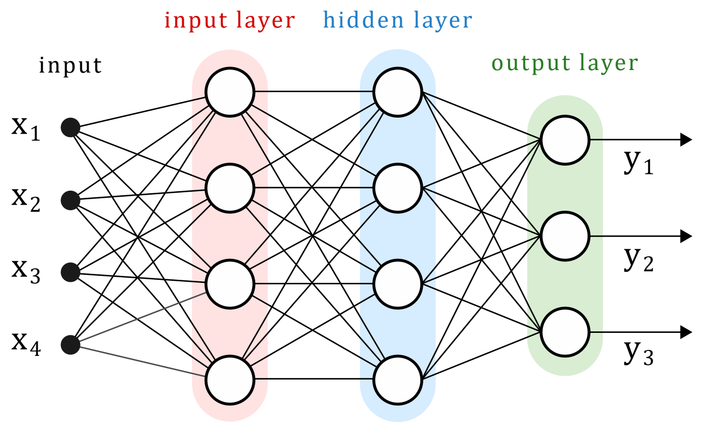


Figure 2: Simple artificial neural network architecture.

2.3 Training ANN

The goal of training ANN is to minimize an error between predicted output and the desired output (also called *target value*). This means we have to adjust the network's weights w_i and bias terms b of each neuron in the network in such a way that the network produces value as close as possible to the desired output. The process of training a neural network consists of the following steps:

1. Computes network's prediction using *Forward Propagation.*
2. Compute the error between predicted and target value according to selected *loss function.*
3. Compute gradients of the loss function with respect to the network's weights using *Backward Propagation.*
4. Update the network's weights using *Gradient Descent* procedure

2.3.1 Loss Function

The loss function is used to optimize the parameter values in a neural network model. The loss is computed by a comparison of predicted and target values. Commonly used loss functions are the following ones:

Mean Squared Error (MSE)
MSE is usually used for regression problems and the error is equal to the square difference of predicted and target value.

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2, \tag{2}$$

where y_i is the target value and, $\hat{y}_i$ is the prediction and N is the number of samples y_i.

Categorical Cross-Entropy (CCE)

CCE is a loss function that is used for single label categorization. Usually, a *softmax* function is applied to the scores before CCE loss is computed.

$$L(y, \hat{y}) = -\sum_{i=1}^{N} y_i \log(\hat{y}_i), \tag{3}$$

where i is the index of the class, y is the true distribution (one-hot vector) and $\hat{y}$ is the predicted distribution.

2.3.2 Forward propagation

The goal of the forward propagation is to generate the network's prediction for a given input. We give the network input data and then we propagate these data through all layers of the network. The last layer produces prediction which is in the following step compared with the target value and the error value is computed.

2.3.3 Backward propagation

Backward propagation (also *backpropagation*) is a way of computing gradients of expressions through recursive application of so-called chain rule.

We are given some function $f(x)$ where x is a vector of inputs. We are interested in computing the gradient of f at x which we denote as $\nabla f(x)$.

In case of neural networks, f will correspond to loss function L, which can be one of the functions described earlier in section 2.3.1. The variable x consists of training data and neural network weights. The gradient $\nabla f(x)$ gives us the direction of the steepest descent (at least in the limit as the step size goes towards zero). If we compute $\nabla L(x)$ with respect to the neural network's weights, we can adjust these weights in the direction of the steepest descent and therefore the loss function gets minimized. This procedure is called *optimization.* The overview of the backpropagation algorithm is illustrated in Figure 3.

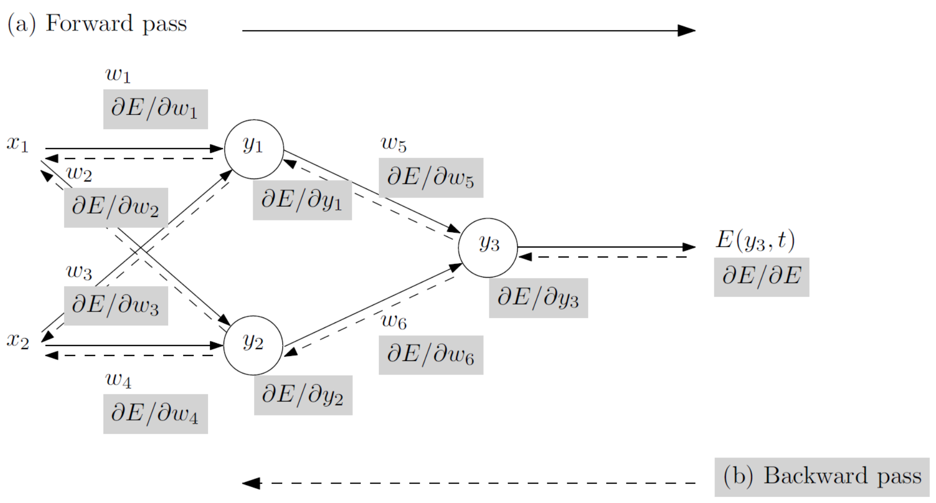

(a) Forward pass
w_1
$\partial E/\partial w_1$
x_1
y_1
w_2
$\partial E/\partial w_2$
$\partial E/\partial y_1$
w_5
$\partial E/\partial w_5$
y_3
$E(y_3, t)$
$\partial E/\partial E$
w_3
$\partial E/\partial w_3$
$\partial E/\partial y_3$
x_2
y_2
w_6
$\partial E/\partial w_6$
w_4
$\partial E/\partial w_4$
$\partial E/\partial y_2$
(b) Backward pass

Figure 3: Overview of backpropagation: (a) Training inputs x_i are fed forward, generating corresponding activations y_i. An error E between the actual output y_3 and the target output t is computed. (b) The error adjoint is propagated backward, giving the gradient with respect to the weights $\nabla_{w_i} E = \left(\frac{\partial E}{\partial w_1}, ..., \frac{\partial E}{\partial w_6} \right)$, which is subsequently used in a gradient-descent procedure (taken from [14]).

2.4 Building blocks of ANN

In this section, I will describe some of the activation functions, layers, and techniques that are commonly used when building an artificial neural network. Later, I will describe layers and techniques that are specific to convolutional neural networks.

2.4.1 Activation functions

As shown in section 2.1, neurons use so-called activation function to produce its output. Activation functions are usually non-linear functions. The rationale behind using a non-linear activation function is that it allows the neuron (or whole neural network) to learn non-linear relationships between the given input and output. If a linear activation function is used, the network can not learn non-linear relationships, which is usually required in many problems. The most commonly used activation functions are described on the following pages.

Hyperbolic tangent

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{4}$$

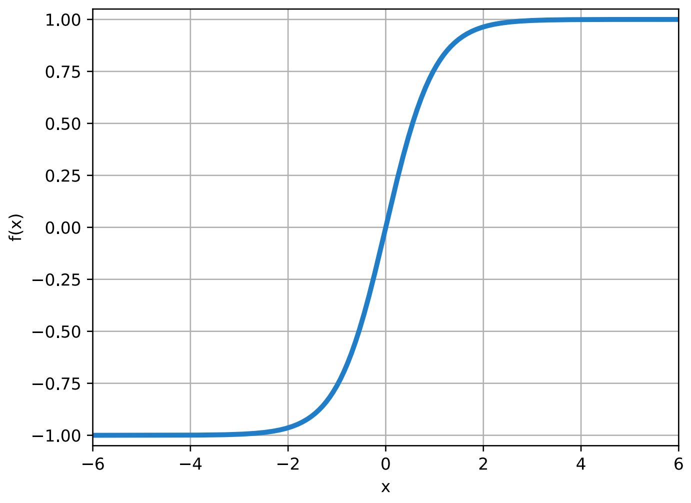


Figure 4: Hyperbolic tangent function.

Hyperbolic tangent is a non-linear activation function. The problem with hyperbolic tangent is that it suffers from a so-called *vanishing gradient problem* [15]. That means that the gradient for relatively large positive and negative input values (approximately $x > 4$ and $x < -4$) is a very small number. The gradient tells us the direction of the steepest descent and this information is essential for network training. If the gradient is small (*vanishes*), then the training process gets more difficult.

ReLU (Rectified Linear Unit)

$$f(x) = max(0, x) \tag{5}$$

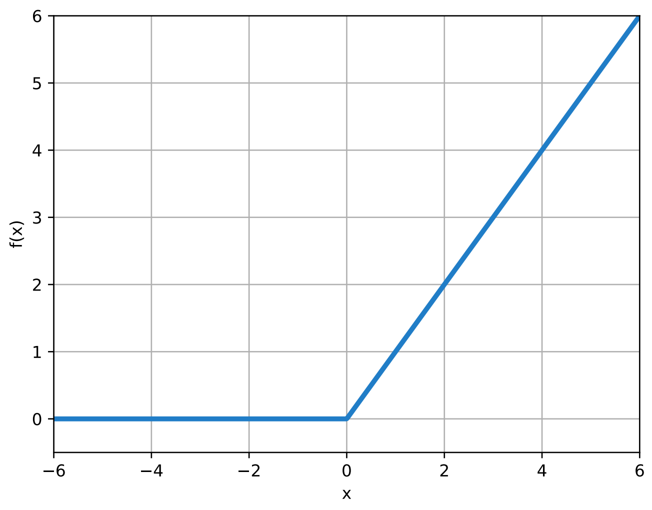


Figure 5: ReLU function.

The advantage of using ReLU instead of the hyperbolic tangent is that the gradient is equal to 1 for all positive values of x, which solves the *vanishing gradient problem* for positive input values. Moreover, the computation of a gradient is much easier. The (theoretical) problem is that the derivative for $x = 0$ is not well defined. Nevertheless, this is an easily solvable problem in real-world implementation, because we can say that the gradient for $x = 0$ is equal either to 0 or 1. Another disadvantage is that the activation value for all $x < 0$ is equal to 0. That means that neurons do not update their weights at all when the input is a negative number. This problem is often referred as *dead neurons problem* [16]. Krizhevsky [17] reports a much faster training with ReLU as opposed to the hyperbolic tangent.

Leaky ReLU

$$f(x) = max(0.1x, x) \tag{6}$$

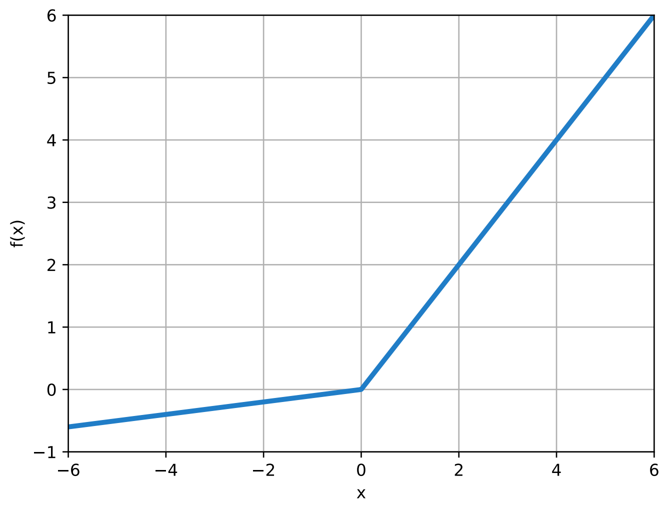


Figure 6: Leaky ReLU function.

Leaky ReLU maintains all the advantages of ReLU and at the same time tries to avoid the *dead neurons problem* by giving the activation function negative slope for $x < 0$. That means neurons will update their weights also for negative input values. We still suffer from not defined derivative for $x = 0$ but we can easily get around this problem in the same way as we did in the case of the ReLU function.

2.4.2 Layers and regularization

Fully connected layer

A fully connected layer connects every neuron in one layer to every neuron in another layer. A fully connected layer expects an input of a fixed size which is defined by the number of input neurons. The dimension of the output data is defined by the number of output neurons. Architecture of this layer is illustrated in Figure 7.

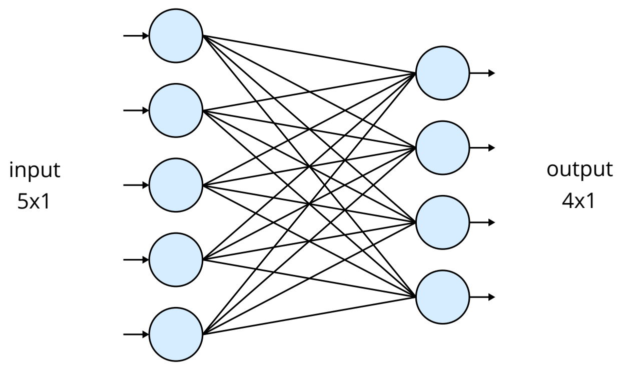


Figure 7: Fully connected layer with 5 input neurons and 4 output neurons.

Softmax layer

The softmax layer performs a softmax operation. Softmax is a function that takes as input a vector x of N real numbers and normalizes it into a probability distribution consisting of N probabilities that are proportional to the exponential of the input elements. The softmax layer is usually used as the last layer in a classification task, where we want to turn the output vector into a probability distribution that tells us the probability that input x belongs to a particular class. The formula for the softmax function is as follows:

$$softmax(x)_i = \frac{e^{x_i}}{\sum_{n=1}^{N} e^{x_n}}, \tag{7}$$

where x_i is $i-th$ element of input vector x, which consists of N values.

Dropout

Dropout is a regularization technique used to reduce *overfitting* in neural networks by repeatedly eliminating randomly selected neurons and their corresponding connections of a neural network during training. Overfitting is a situation when the neural network is "over-trained" on the training data and thus can not generalize well when unseen (test) data are presented.

The neurons in the network are usually dropped out with a probability of 0.5. These "dropped" neurons do not contribute to the forward pass and do not participate in the backpropagation. This means that every time an input is presented, the neural network samples a different architecture, but all these architectures share weights. This technique reduces complex co-adaptations of neurons since a neuron cannot rely on the presence of particular other neurons. It is, therefore, forced to learn more robust features that are useful in conjunction with many different random subsets of the other neurons [18].

Batch Normalization

Batch Normalization [19] is a commonly used technique that enables faster and more stable training of deep neural networks.

This is achieved by augmenting the network with additional layers that force the activations throughout the network to take on a unit Gaussian distribution. This change makes the landscape of the corresponding optimization problem significantly smoother. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks [20].

2.4.3 Convolutional Neural Networks

In this section, I will describe components and techniques that are specific to Convolutional Neural Networks (CNNs). CNN are special type of Artificial Neural Networks. CNNs are most commonly used for the analysis of visual data. They are still made up of neurons that have trainable weights and biases like regular neural networks. The difference is that CNN architectures make the explicit assumption that the inputs are structured data (commonly represented as gray-scale or RGB images), which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network.

Input Layer

In non-convolutional neural networks, the input is usually a flat vector of N real numbers. In CNNs the input typically has the dimension of *(img width; img height; channels)*, where

- *img width*
  represents the image width in pixels.
- *img height*
  represents the image height in pixels.
- *channels*
  is usually 3, which represents R,G,B color channels. If the input is a grayscale image, the *channels* parameter will be equal to 1.

The layer holds the raw pixel values of the image, which means values from 0 to 255. This range is often rescaled to a range from 0 to 1.

Convolutional Layer

The Convolutional layer is the core building block of a Convolutional Neural Network architecture. This layer works on a principle of 3D (or 2D) convolution. Convolution is the simple application of a filter to an input that results in activation. Repeated application of the same filter to an input results in a map of activations called a feature map. See Figure 8 for an illustration. Usually, the convolutional layer has more filters. Each filter creates its own feature map and number of filters specifies the depth of the output of the convolutional layer (see Figure 9). The input to the convolutional layer is often 3 dimensional. In that case, the principle stays the same and the filters are also 3 dimensional.

The advantage of using a convolutional layer is that we can radically reduce the number of parameters compared to a fully-connected layer.

The basic parameters of the Convolutional layer are:

- *kernel size*

  Specifies the dimension of convolution mask.

- *filters*

  Specifies the number of filters which then specifies the depth of the output.

- *strides*

  Specifies how much should we shift the convolution filter after each convolutional operation. See Figure 8 for an illustration.

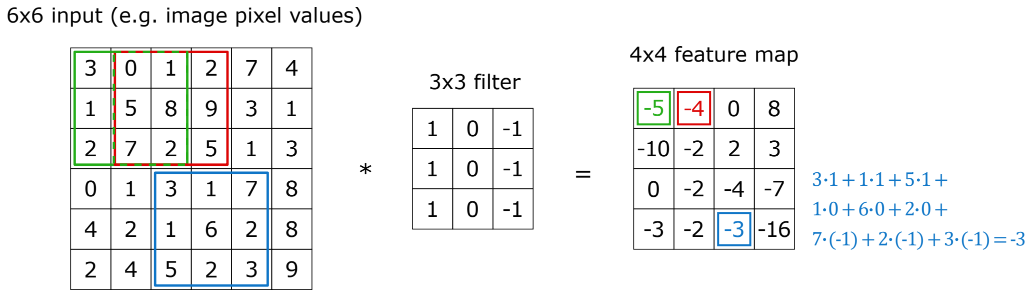


Figure 8: Computation of 2D convolution with 3x3 filter (or kernel) size and stride 1. We can see that the dimension of the output is reduced compared to the dimension of the input. In order to preserve dimensions so-called *padding* operation can be applied to the input.

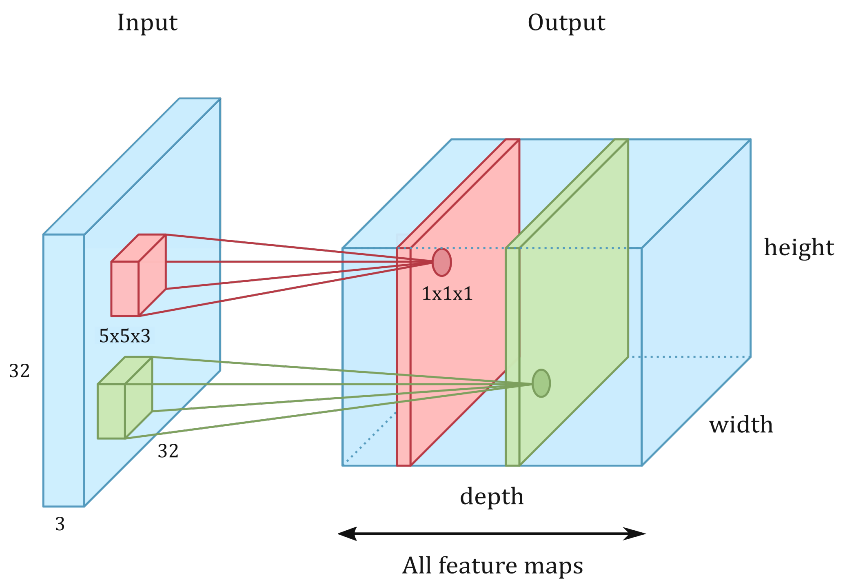


Figure 9: 3D convolutional layer. The depth of the output depends on the number of filters we have. The width and height depend on the stride parameter.

Pooling Layer

The Pooling Layer is used to compress the information propagated to the next level of network and therefore reduce the number of parameters and computational difficulty. A common form of pooling layer is applying filter size 2x2 with a stride of 2 using MAX operation. See Figure 10 for this example. Another common form of pooling is the one with $AVERAGE$ operation.

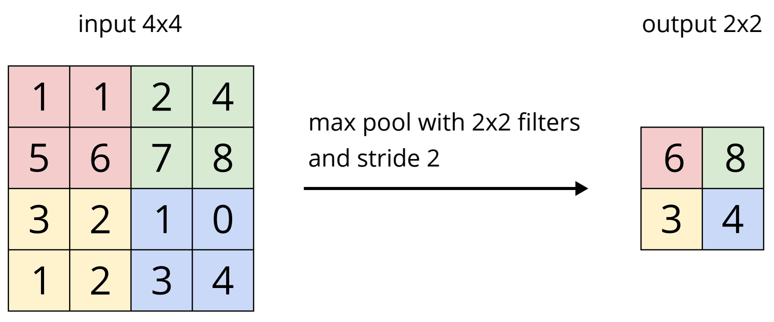


Figure 10: Example of MAX pooling operation with 2x2 filter and stride 2

Residual blocks

Deeper neural networks are more difficult to train. This problem is addressed by so-called *Residual Neural Network* that was presented in 2015 [21]. Residual Network implements *residual connections*, which connect the output of one layer with the input of a layer that is a few (usually 2-3) layers ahead. The residual connections simply perform *identity* mapping, and their outputs are added to the outputs of the stacked layers (see Figure 11). Identity shortcut connections add neither extra parameter nor computational complexity. Instead of learning an original underlying mapping $H(x)$, the model learns residual mapping $F(x) = H(x) - x$. The original mapping is recast into $F(x)+x$. The hypothesis is that these residual mappings are easier to optimize and therefore it makes training of deep neural networks easier.

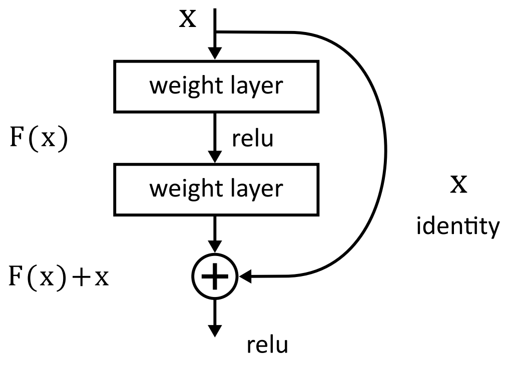


Figure 11: Residual connection

3 Architectures

This section briefly describes network architectures used in this work. I chose to use VGG16, InceptionNet, ResNeXt, and EfficientNet neural network architectures. In 2014, VGG16 was one of the first CNNs which won the ImageNet Large Scale Visual Recognition Challenge [22]. Therefore this architecture proved that CNNs have a big potential to solve image classification tasks. I will use the performance of the VGG16 network as a baseline performance of convolutional architectures. The InceptionNet and ResNeXt architectures use more advanced concepts than VGG16 and are considered to be state-of-the-art convolutional neural networks. EfficientNet is a modern network architecture which should achieve slightly worse accuracy than InceptionNet and ResNeXt, but should require significantly less computational resources.

3.1 VGG16

VGG16 is a convolutional neural network model introduced in 2014 by K. Simonyan and A. Zisserman [23]. The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes [22]. This architecture was built on the idea of having smaller filter sizes and deeper network. The model architecture consists of 12 convolutional, 6 max pooling, 3 fully-connected and one softmax layer (see Figure 12). Convolutional layers use 3x3 filters. We use this network to get a baseline accuracy of convolutional network architectures. Architectures described on the following line are expected to achieve better accuracy.

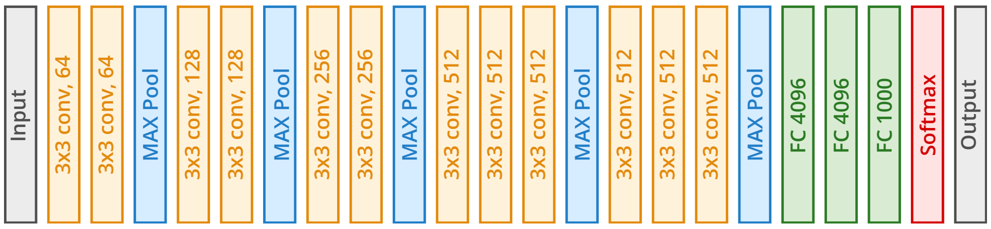


Figure 12: VGG16 network architecture.

3.2 Inception Networks

One of the common problems in computer vision is that the salient parts of the image can significantly vary in sizes. For example, we try to detect a cat, the cat can fill the whole image or there can be just a small cat in the top right corner. The area occupied by the cat is different in each image. This is important because this influences the choice of the right kernel size of convolutional layers. A larger kernel is preferred for information that is distributed more globally, and a smaller kernel is preferred for information that is distributed more locally.

Inception networks try to solve this problem by using multiple kernel sizes that operate on the same level. Outputs given by the application of these kernels are then concatenated and sent to the next inception module [10]. The overview of the working principle is captured in Figure 13.

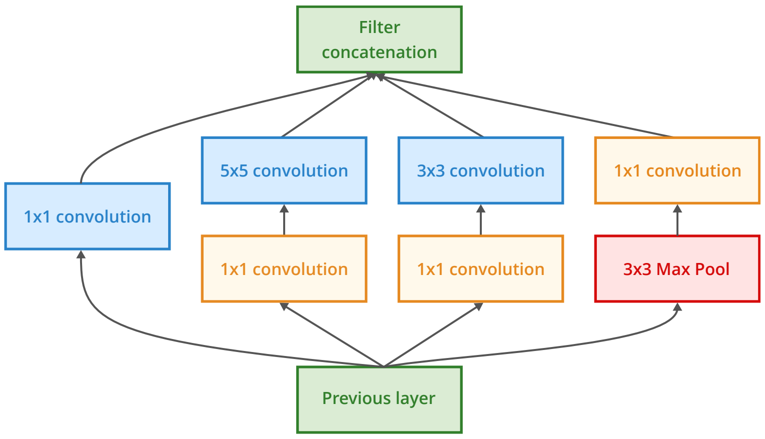


Figure 13: Inception module. Convolutions using three different filter sizes (blue) are applied to the data coming from the previous layer. Additionally, max pooling is also performed. The outputs are depth-wise concatenated and sent to the next inception module. Additionally, in order to reduce computational complexity, other 1x1 convolutions (yellow) are added to reduce the depth of the data coming from the previous layer. Taken from [10].

3.3 ResNeXt

ResNeXt [11] is an image classification network that tries to decrease the complexity of designing a network architecture as the number of hyper-parameters grows (width, filter sizes, strides, *etc.*). This network achieved a 3.03% top-5 error on the test set in ILSVRC competition [22].

The network is constructed by repeating a building block that aggregates a set of transformations with the same topology. The simple design results in a homogeneous, multi-branch architecture that has only a few hyper-parameters to set.

ResNeXt builds on the architecture of ResNet [24] which takes advantage of residual blocks described in section 2.4.3. ResNeXt builds on this idea and exposes a new dimension called ”cardinality” (the size of the set of transformations). The ”classical” residual blocks are then replaced with ”cardinality” blocks. If the depth of the block is equal to 2, the block reformulation creates trivial topologies that are equivalent to each other in complexity (see figure 14). If the block has depth ≥ 3, the reformulations produce nontrivial topologies. For this example see Figure 15.

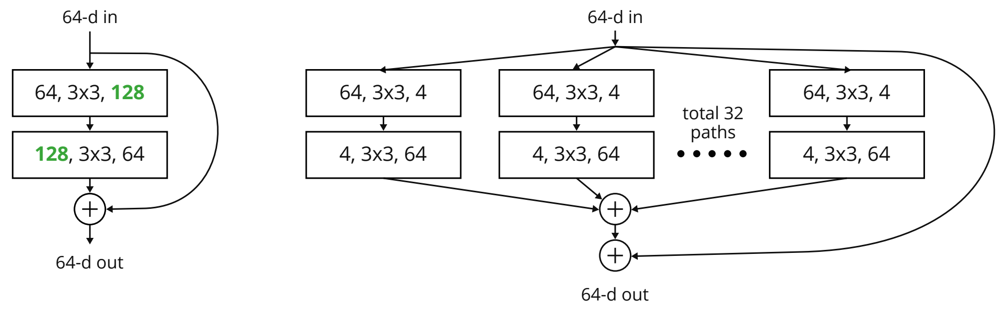


Figure 14: Equivalent blocks. **Left:** A residual block of ResNet [24]. **Right:** A block of ResNeXt with the cardinality of 32, with the same complexity. Note that $32 \cdot 4 = 128$, which is the number of input/output channels in ResNet block (bold). A layer is shown as (number of input channels, filter size, number of output channels).

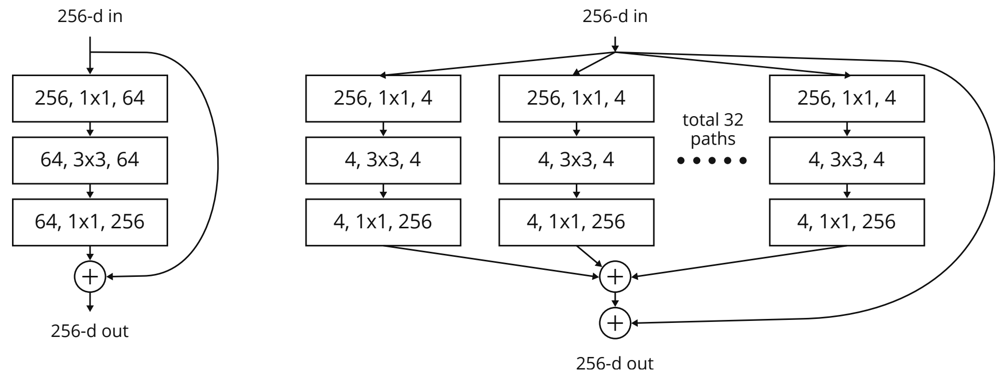


Figure 15: **Left:** A residual block of ResNet [24]. **Right:** A block of ResNeXt with the cardinality of 32, with roughly the same complexity. A layer is shown as (number of input channels, filter size, number of output channels).

3.4 EfficientNet

Scaling up Convolutional Neural Networks is commonly used practice in order to achieve better accuracy. On of the ways to scale CNNs is to increase their depth

[24], width [25] or image resolution [26]. However, in practice choosing the right scaling and its parameters can be a tedious process leading to sub-optimal accuracy.

EfficientNet [12] proposes a new scaling method that uniformly scales all scaling dimensions (width, depth, resolution) using a simple yet highly effective "compound scaling method". The authors of the original paper [12] empirically observed that the scaling dimensions are not independent thus it is necessary to coordinate and balance these dimensions in order to get the best results. The principle of the EfficientNet approach is to scale each of the network's dimensions with a constant ratio. See 16 for an illustration.

For example, if we want to use 2^N times more computational resources, then we can simply increase the network depth by α^N, width by β^N, and image resolution by γ^N, where α, β, γ are constant coefficients determined by a small grid search on the original small model [12].

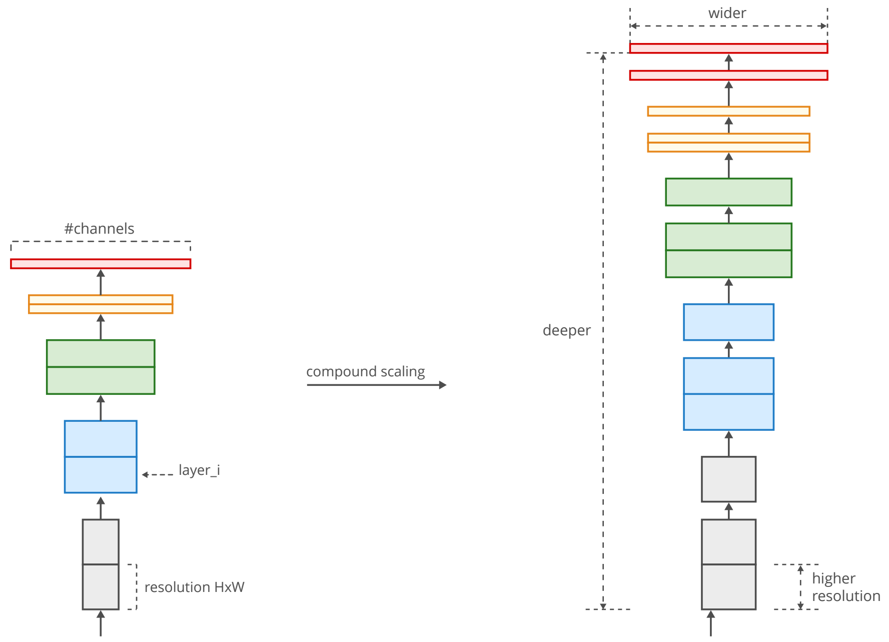


Figure 16: EfficientNet scaling method. **Left:** baseline model **Right:** model scaled using compound scaling (taken from [12]).

Part III

System Imlementation and Evaluation

Building an end-to-end Driver Attention Warning system using convolutional neural networks consists of the following steps:

1. Obtain an appropriate dataset to train the network. This dataset should include all actions we want to detect and should include as much data as possible because that is an essential requirement for training deep neural networks.
2. Build a convolutional neural network architecture. One could build a brand new architecture or use an architecture that was already tested on a different problem and make appropriate modifications to it. This practice is known as *transfer learning* [27] and it is the approach I chose in this thesis.
3. Train the network and tune network parameters in order to achieve the best performance. Parameters can be related to network architecture (number of layers or number of neurons, activation function type, dropout, convolution kernel size *etc.*) or the parameters can be related to the training algorithm (like learning rate or optimization algorithm type).
4. Build the final system in such a way the the input is an image and the output is driver's action.

See Figure 17 for a high-level overview of such a system.

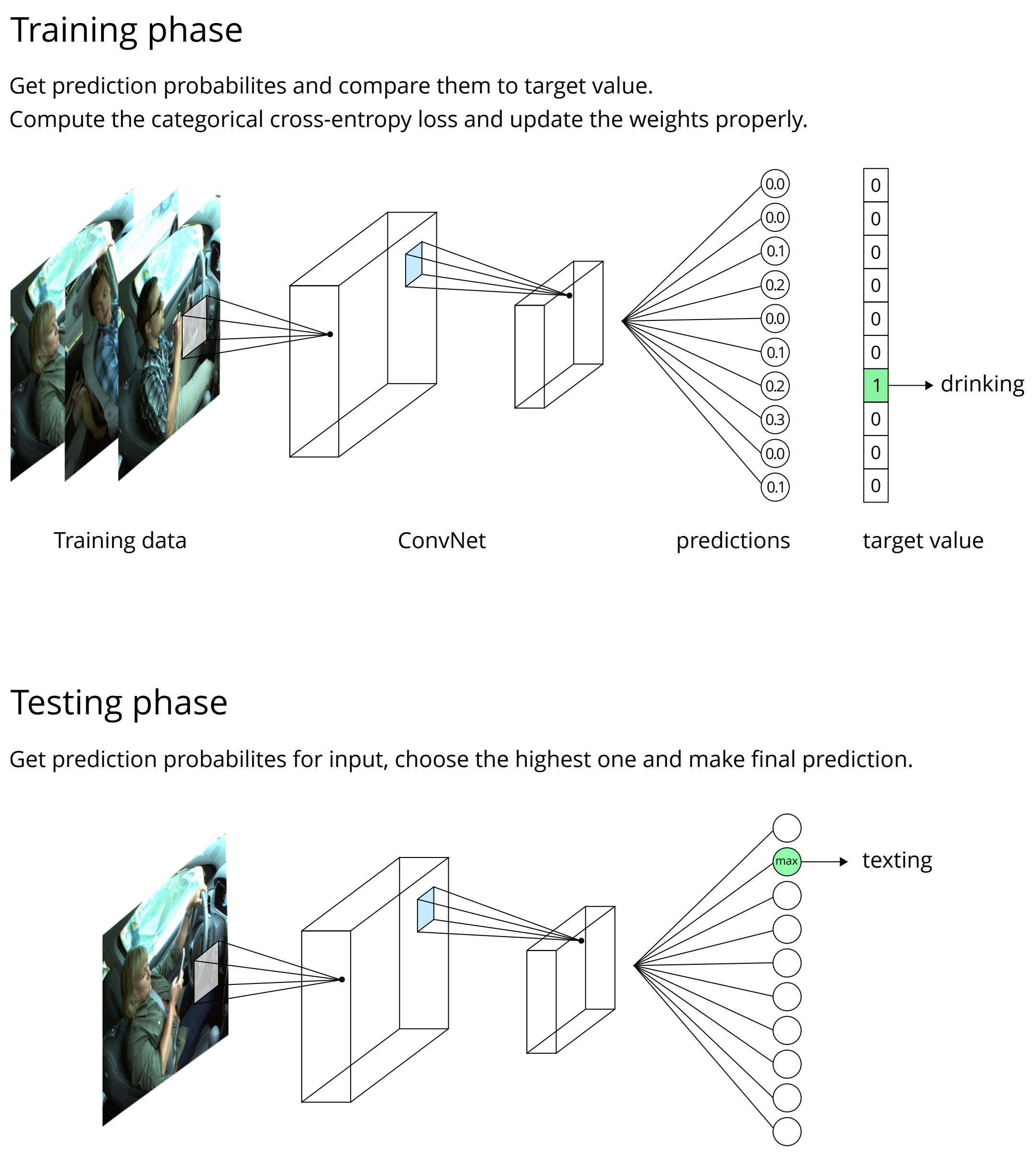

Training phase
Get prediction probabilites and compare them to target value.
Compute the categorical cross-entropy loss and update the weights properly.
0.0
0.0
0.1
0.2
0.0
0.1
0.2
0.3
0.0
0.1
0
0
0
0
0
0
1
0
0
0
drinking
Training data
ConvNet
predictions
target value
Testing phase
Get prediction probabilites for input, choose the highest one and make final prediction.
max
texting

Figure 17: A high-level overview of building a basic end-to-end DAW system using convolutional neural networks.

4 Datasets description

I use two different datasets in this work. To train the networks, I use State Farm's dataset that was published as a part of a Kaggle competition [28]. The model is also evaluated on this data using Kaggle's evaluation algorithm. However because this dataset was presented as part of a competition, the ground truth labels for test data are unknown and therefore it is difficult to do any further analysis (e.g. compute confusion matrix or check the exact accuracy of the model's predictions).

For the reasons mentioned above another dataset created by Abouelnaga et al. [9] is used for a more rigorous analysis. Using two different datasets with the same categories of driver's distraction is also a great way how to evaluate the network's ability to apply learned knowledge from one dataset to another slightly different dataset.

4.1 State Farm dataset

This dataset is used for training and evaluating the model. The dataset includes 102 150 images of drivers that are separated into 10 categories which are labeled as follows:

- c0: safe driving
- c1: texting - right
- c2: talking on the phone - right
- c3: texting - left
- c4: talking on the phone - left
- c5: operating the radio
- c6: drinking
- c7: reaching behind
- c8: hair and makeup
- c9: talking to passenger

Sample images from the dataset are shown in Figure 20. The dataset is split into *train* and *test* sets. The *train* set contains 22 424 images and the *test* set contains

79 726 images. All of the images have the same resolution of $640 \times 480\ px$. We know the class and the driver ID for each image in the training data (driver ID specifies which driver is on the image). There are 26 different drivers in the dataset. Class distribution is plotted in Figure 19.

As mentioned before, the images are split into *train* and *test* sets. The better approach is to split data into the three following sets:

- **train**
  These images are used solely to train the model. The model learns its weights from this data. The accuracy of predictions for training data is not very important if we want to evaluate the model's real-world accuracy. The model can have great accuracy on training data, but it does not mean it will generalize well when unseen data are presented.
- **validation**
  The model is not trained using these images. These images are used just measure the accuracy of the model while tuning model hyper-parameters. We cannot use validation accuracy as the actual accuracy of the model on unseen data. That is because we chose the best parameters for the one specific set of validation data.
- **test**
  These images represent the real world (never seen before) data. We do not train the model or tune any parameters using these images. The model's final accuracy is the accuracy achieved on this dataset.

This approach gives us a more realistic estimation of the model's accuracy. In this thesis, validation set size is set as 25 % of training data and the classes in validation set are distributed equally as in the training set.

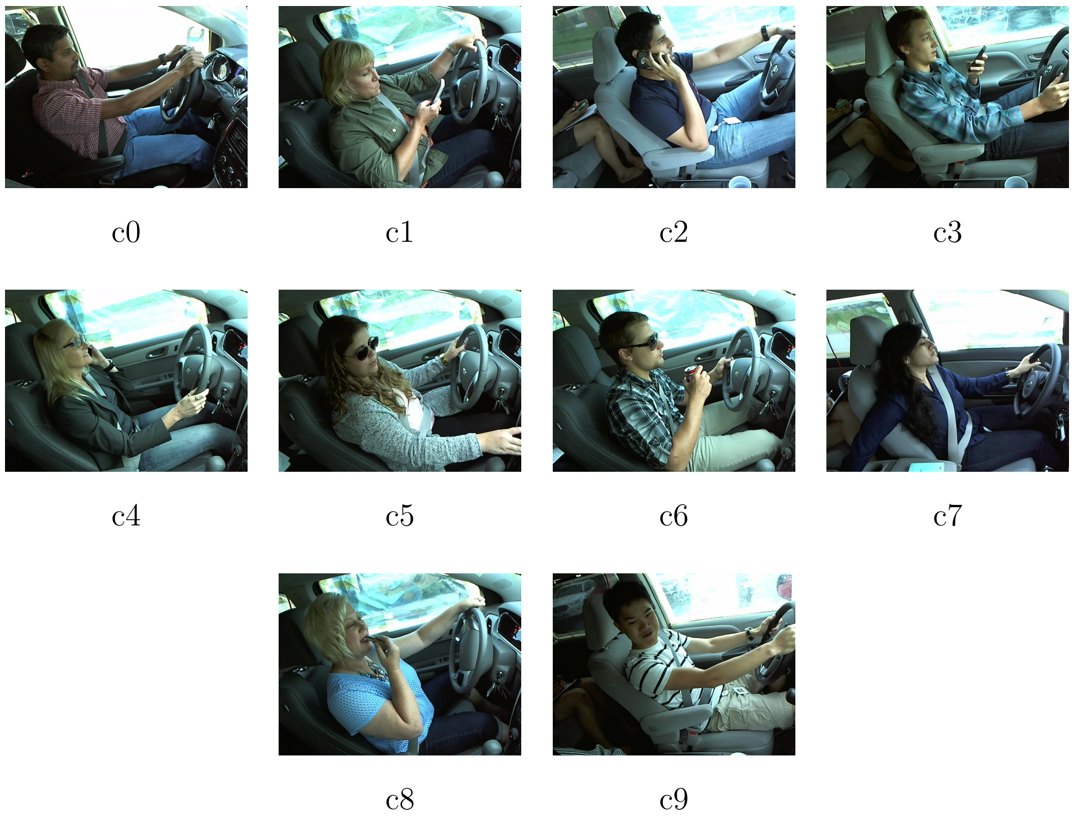


Figure 18: Sample images for each class from the State Farm dataset.

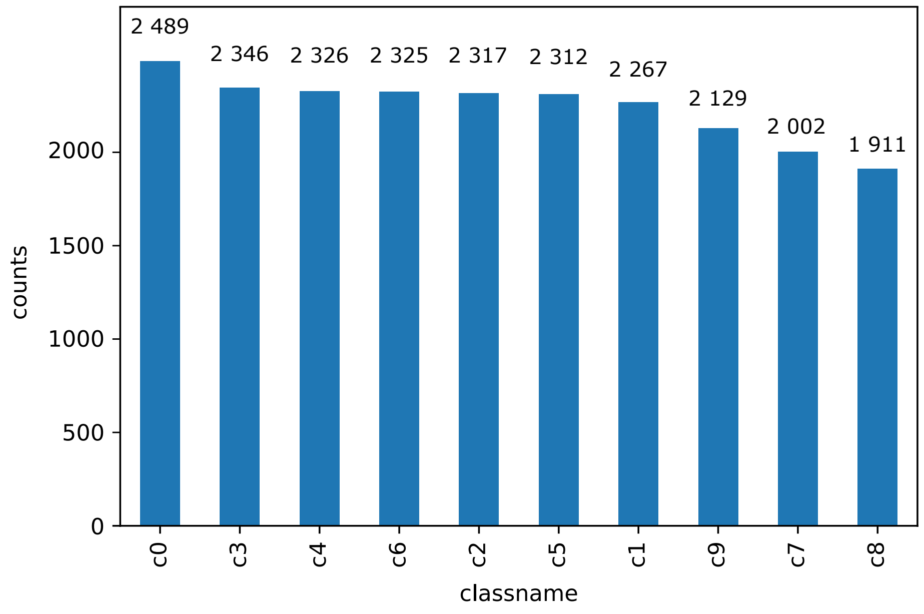


Figure 19: Distribution of classes in the training data of the State Farm dataset.

4.2 Abouelnaga dataset

This is a smaller dataset and contains 17 308 images. Images are separated into the same classes as images in the State Farm dataset. This dataset includes different car interiors and different people than the State Farm's dataset. Images have a different resolution (1920 × 1080 *px*) and are taken from a different angle than the images in the State Farm's dataset. If the model trained on the State Farm's dataset performs well on this dataset too, it means the model has a good ability to generalize its knowledge. Sample images from Abouelnaga dataset are shown in Figure 20. Class distribution is in Figure 21.

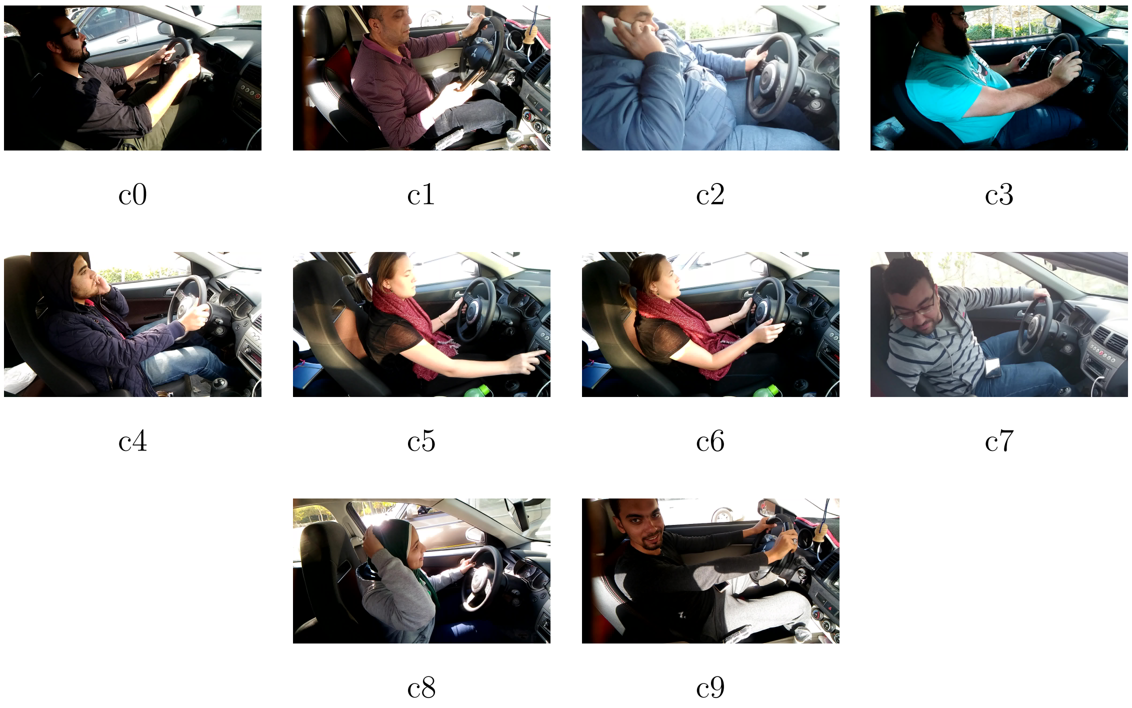


Figure 20: Sample images for each class from the Abouelnaga dataset.

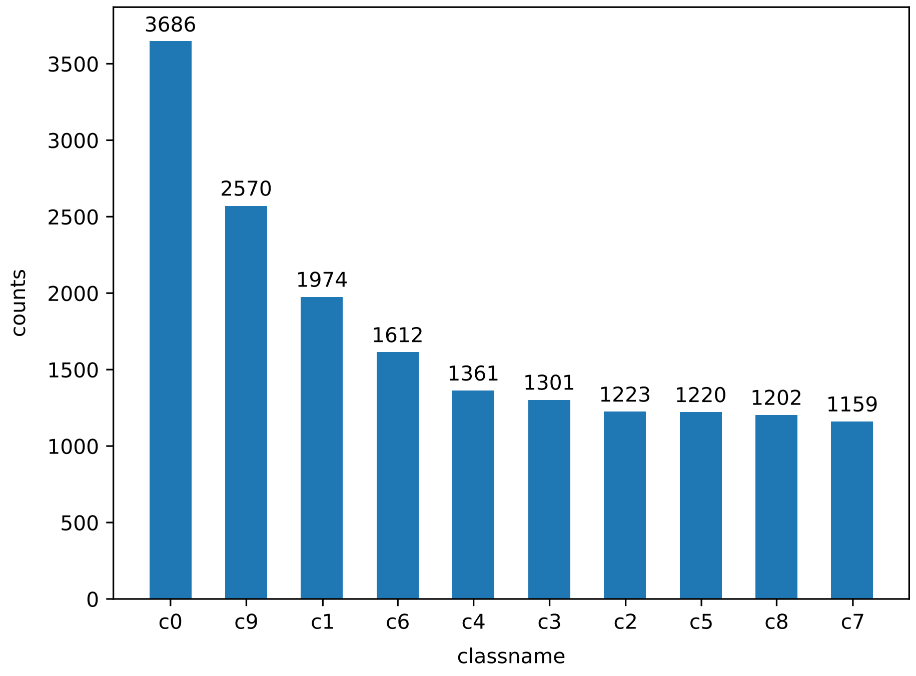


Figure 21: Distribution of classes in Abouelnaga dataset.

4.3 ImageNet

ImageNet [22] is a large image database containing more than 14 million hand-annotated images. In this thesis I take advantage of ImageNet database for the purpose of *transfer learning* [27]. Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task.

Neural networks used in this thesis were trained to classify images from the ImageNet database into one of 1 000 categories. These pre-trained weights can be used in the problem of driver's action detection. The rationale behind using transfer learning in computer vision is that a model that is trained on a big-scale database like ImageNet has already been trained to recognize universal low-level image features (e.g. edges). If we use these pre-trained weights, it is often sufficient to retrain these weights in order to solve a more specific task. This technique saves a lot of computational resources, less data is needed to train the model and we can take advantage of already well-tested network architectures.

5 Working Environment

In this section, I will describe software and hardware that I used to build the driver attention warning system presented in this thesis.

5.1 Software

The code for this thesis is written in Python version 3.7.4 and takes advantage of the following libraries:

- **tensorfow 1.13.1**
  TensorFlow is an end-to-end open-source platform for machine learning [29]. In this thesis, I use TensorFlow as a backend of Keras library which is used for building and training artificial neural networks.
- **keras 2.2.5**
  Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation [30].
- **numpy 1.17.3**
  NumPy is the fundamental package for scientific computing with Python. It makes manipulation with high dimensional data easier and more effective [31].
- **pandas 0.24.2**
  Pandas is an open-source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language [32].
- **matplotlib 3.0.3**
  Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms [33].

5.2 Hardware

The models were trained using graphics processing units (GPUs) which are very suitable for training neural networks. The main reason is that GPU can perform matrix multiplication and convolution more efficiently than the central processing unit (CPU) and the computation can be easily parallelized.

The GPUs were provided by MetaCentrum VO, which is a catch-all virtual organization of the Czech National Grid Organization MetaCentrum NGI. MetaCentrum operates and manages distributed computing infrastructure consisting of computing and storage resources owned by CESNET as well as resources of co-operative academic centers within the Czech Republic [34].

GPUs used in MetaCentrum:

- nVidia Tesla T4 16GB
- nVidia Tesla K20 5GB
- GPU nVidia GeForce GTX

6 Network implementation

Architectures for VGG16, ResneXt and InceptionV3 networks were implemented using Keras Applications API [35]. Custom parameters like trainable layers, input and output dimensions were specified by popping or adding custom layers. Example of the InceptionV3 network implementation can be seen in appendix.

EfficientNet architecture is not part of the official Keras or TensorFlow API at the time of writing this thesis. For that reason, this network architecture had to be reimplemented and the code is inspired by the implementation made by Pavel Yakubovskiy [36]. Because of installation issues at MetaCentrum, Yakubovskiy's implementation was modified for the purpose of this thesis using Keras library with TensorFlow backend only.

7 Data Preprocessing

The default image pixel values are in range 0 - 255. All image pixel values were before the training and testing process re-scaled from a range of 0 - 255 to the range 0 - 1 because that is the format expected by the pre-trained network architectures. The data augmentation technique was also applied before training the networks.

7.1 Data augmentation

Having enough data is a key element for a successful training process of deep neural networks, which helps the networks to learn more robust features and thus prevent overfitting. Augmenting the training data we already have is a common practice. Based on the experiments the following augmentation techniques improved the models' performance the most.

Zoom

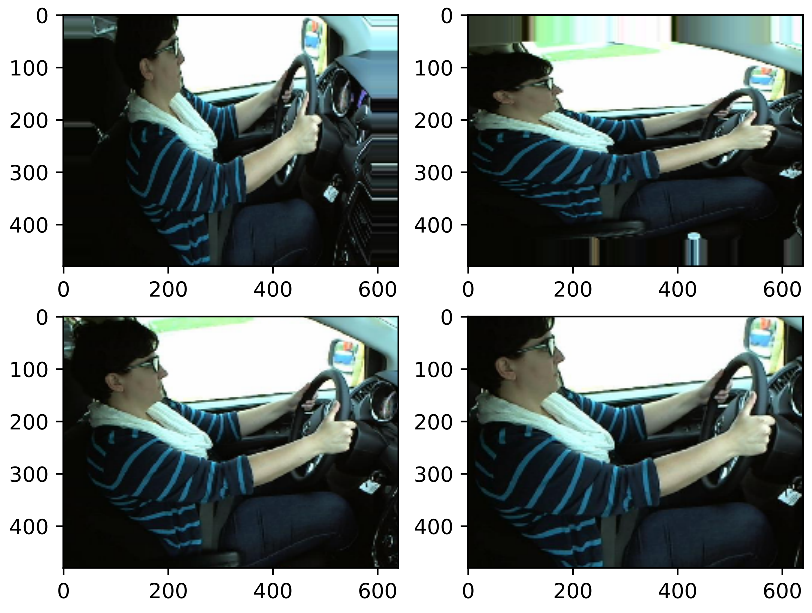


Figure 22: Zoom augmentation. The same image with different zoom ratios.

Shear

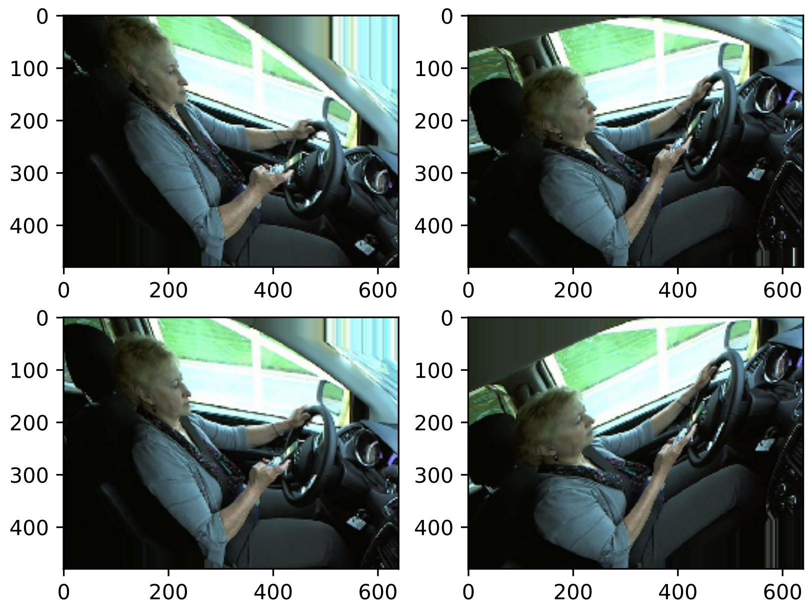


Figure 23: Shear augmentation. The same image with different shear ratios.

8 Experiments

In this section, I describe all of the experiments I have done in order to achieve the best performance on the State Farm dataset using the Kaggle evaluation algorithm. Kaggle evaluates the predictions using a multi-class logarithmic loss function. The formula for *multi-class logaritmic loss* is then,

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} log(p_{ij}), \tag{8}$$

where N is the number of images in the test set, M is the number of image class labels, log is the natural logarithm, y_{ij} is 1 if observation i belongs to class j and 0 otherwise, and p_{ij} is the predicted probability that observation i belongs to class j.

The output of the file that is submitted to Kaggle is a CSV file with the image file name, and a probability for each class. The format of the submission file can be seen in Figure 24.

```
img, c0, c1, c2, c3, c4, c5, c6, c7, c8, c9

img_0.jpg, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0

img_1.jpg, 0.3, 0.1, 0.6, 0, 0, 0, 0, 0, 0, 0
...
```

Figure 24: Example submission CSV file that is ready for the Kaggle evaluation algorithm.

8.1 Initial experiments - no augmentation

The models were trained using the original dataset without any augmentations or preprocessing. The learning rate remained constant throughout the whole training process. The training process was stopped once the validation accuracy did not improve. All of the models use pre-trained weights on the ImageNet database and these weights were set as a starting point. Training without pre-trained weights was not successful. When training the InceptionV3 network, only the last two inception blocks and one fully connected layer were trained (see Section 6). Updating all layers of InceptionV3 did not result in successful training and better accuracy was achieved by preventing the first 249 layers from training. When training VGG, ResNeXt, and EfficientNet, all weights were updated during the training, because this approach led to the best results. Results are in Table 1.

Architecture	learning rate	optimizer	epochs	Kaggle loss	val. accuracy (%)
VGG16	$1e^{-4}$	SGD	10	1.1053	99.26
InceptionV3	$1e^{-4}$	Adam	5	2.6683	73.18
ResNeXt50	$1e^{-4}$	SGD	10	0.7088	99.15
EfficientNetB0	$1e^{-4}$	SGD	10	0.7524	99.44
EfficientNetB4	$1e^{-4}$	SGD	10	0.7569	99.06

Table 1: Training results for initial experiments.

From Table 1 we see that the best test score was achieved by ResNeXt50, which achieved the Kaggle loss score equal to 0.7088. InceptionV3 network was by far the most difficult one to train. Training InceptionV3 with SGD optimizer did not lead to successful training and Adam optimization algorithm had to be used. Also, only the last two inception blocks with a fully connected layer on top of it were trained. Training all layers of the InceptionV3 network did not lead to successful training. Both the EfficientNetB0 and EfficientNetB4 achieved almost the same score, but EfficientNetB0 was faster to train thanks to its more compact architecture. VGG16's performance in the middle. It performed significantly better than InceptionV3, but also significantly worse than the other networks.

8.2 Augmenting data and tuning parameters

In this experiment, the data were augmented before training the network. Based on the experiments the best result were achieved by implementing *shear* and *zoom* augmentation techniques as described in section 7.1. In terms of parameters, the most important was the right choice of a starting learning rate, optimization algorithm, and the number of training epochs. By slowly decreasing the learning rate as the training loss converged another extra accuracy percentages were gained. Another technique that helped training the ResNeXt nad EfficientNet was training the last few layers first and after that, the training process continued while training all layers of the network.

Architecture	learning rate	optimizer	epochs	Kaggle loss	val. accuracy (%)
VGG16	$1e^{-4}$	SGD	8	0.5327	98.81
InceptionV3	$1e^{-4}$	Adam	6	0.6356	85.18
ResNeXt50	$1e^{-5}, 1e^{-6}$	SGD	8 + 2	0.4439	97.97
EfficientNetB0	$1e^{-3}$	SGD	10	0.6506	99.66
EfficientNetB4	$1e^{-3}$	SGD	10	0.6084	98.99

Table 2: Training results for augmented data and tuned optimizer parameters.

If we take a look at Table 2 we notice significant overall improvement on test data among all of the network architectures. Moreover, the validation accuracy of VGG, ResNeXt50 and EfficientNet networks decreased. This indicates that the over-fitting problem was reduced which is mainly caused by data augmentation.

The best result on test data was achieved by ResNext50, which performs significantly better than any other network. While ResNext50 is the most accurate one, it requires much more computational resources for example compared to EfficientNetB0, which performs the worst, but still better than the ResNeXt50 trained on non-augmented data. For comparison, while training the ResNext50 model took about 24 hours, EfficientNetB0 was trained under 10 hours using the same hardware. The size of the file which stores ResNext50's trained weights is 115 MB while the same file containing EfficientNetB0's weights is of size 32 MB. We can see there is some trade-off between accuracy and the required computational resources.

EfficientNetB4 is re-scaled EfficientNetB0 so that it is deeper and wider. It should theoretically achieve better results than EfficientNetB0 because deeper network should be capable of better, or at least the same performance as a smaller network theoretically. This assumption was confirmed and EfficientNetB4 indeed achieved better performance than EfficientNetB0. The better performance came with the cost of higher computational requirements

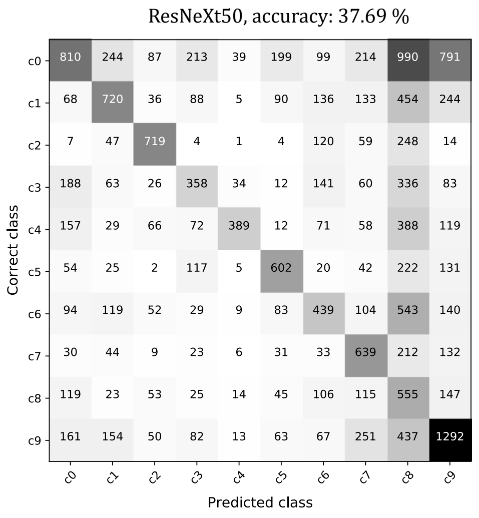
VGG16 network achieved the second-best result and we can see that data augmentation and parameter optimization had a huge effect on its performance. The biggest drawback of the VGG16 network was its inefficiency compared to other networks. These networks get trained faster and also the predictions were made faster. For example, EfficientNets were about 50% faster in making predictions, which is a significant difference especially if the goal would be to categorize images in real-time.

The augmentation process helped the InceptionV3 network the most. The loss score was reduced from 2.6683 to 0.6356 which results in the fourth-best performance. We can see that the validation accuracy is by far the lowest one compared to all the other networks, while the test loss score is not that different. This fact suggests an idea that the network is not over-fitted and thus will generalize well. Because only the last two inception blocks and one fully connected layer were trained, the training process was very fast (about 50 % faster compared to ResNeXt and VGG16 networks using the same hardware).

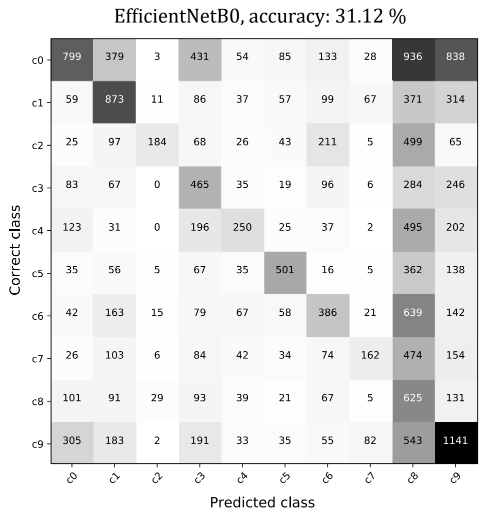
8.3 Focusing on problematic classes - c0, c8, c9

For a more rigorous analysis I plotted confusion matrices in Figure 25 for models trained in section 8.2. The data in these matrices correspond to the Abouelnaga dataset, which contains slightly different images than the State Farm dataset and therefore the accuracy does not seem that great. On the other hand, the accuracy of a different dataset allows us to asses models' ability to generalize. We can notice a common problem from these matrices. Most of the networks have the biggest problem with the correct classification of classes *c0 - driving safe*, *c8 -hair and make-up* and *c9 -talking to passenger*. Although InceptionV3 network didn't perform the best on State Farm's dataset, it achieved by far the highest accuracy on Abouelnaga dataset which is 9.75 % higher than the second best-model, which is VGG16. This result confirms the presumption I made in the previous section that this network will generalize well according to the results from Section 8.2.

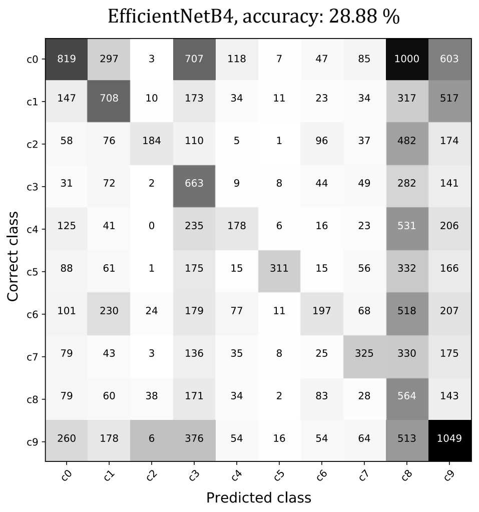
If we take a closer look at the most problematic classes we can understand why it is problematic to classify these classes correctly. I plotted sample images of these classes in Figure 26. These activities are often ambiguous and are difficult to correctly classify even for a human observer. Further experiments focus on easing this problem.



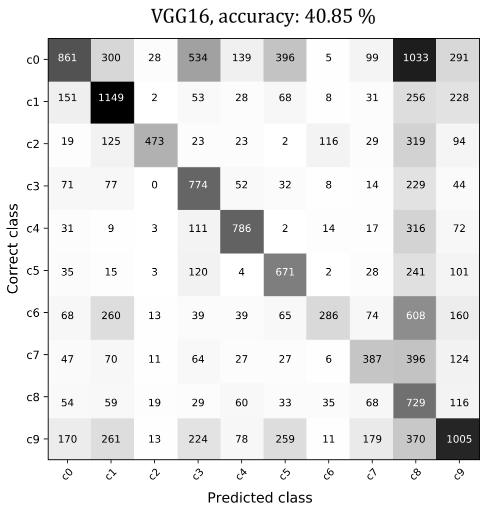
	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9
c0	810	244	87	213	39	199	99	214	990	791
c1	68	720	36	88	5	90	136	133	454	244
c2	7	47	719	4	1	4	120	59	248	14
c3	188	63	26	358	34	12	141	60	336	83
c4	157	29	66	72	389	12	71	58	388	119
c5	54	25	2	117	5	602	20	42	222	131
c6	94	119	52	29	9	83	439	104	543	140
c7	30	44	9	23	6	31	33	639	212	132
c8	119	23	53	25	14	45	106	115	555	147
c9	161	154	50	82	13	63	67	251	437	1292



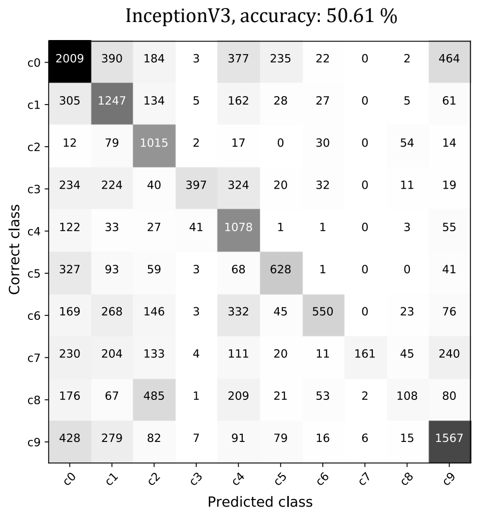
	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9
c0	799	379	3	431	54	85	133	28	936	838
c1	59	873	11	86	37	57	99	67	371	314
c2	25	97	184	68	26	43	211	5	499	65
c3	83	67	0	465	35	19	96	6	284	246
c4	123	31	0	196	250	25	37	2	495	202
c5	35	56	5	67	35	501	16	5	362	138
c6	42	163	15	79	67	58	386	21	639	142
c7	26	103	6	84	42	34	74	162	474	154
c8	101	91	29	93	39	21	67	5	625	131
c9	305	183	2	191	33	35	55	82	543	1141



	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9
c0	819	297	3	707	118	7	47	85	1000	603
c1	147	708	10	173	34	11	23	34	317	517
c2	58	76	184	110	5	1	96	37	482	174
c3	31	72	2	663	9	8	44	49	282	141
c4	125	41	0	235	178	6	16	23	531	206
c5	88	61	1	175	15	311	15	56	332	166
c6	101	230	24	179	77	11	197	68	518	207
c7	79	43	3	136	35	8	25	325	330	175
c8	79	60	38	171	34	2	83	28	564	143
c9	260	178	6	376	54	16	54	64	513	1049



	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9
c0	861	300	28	534	139	396	5	99	1033	291
c1	151	1149	2	53	28	68	8	31	256	228
c2	19	125	473	23	23	2	116	29	319	94
c3	71	77	0	774	52	32	8	14	229	44
c4	31	9	3	111	786	2	14	17	316	72
c5	35	15	3	120	4	671	2	28	241	101
c6	68	260	13	39	39	65	286	74	608	160
c7	47	70	11	64	27	27	6	387	396	124
c8	54	59	19	29	60	33	35	68	729	116
c9	170	261	13	224	78	259	11	179	370	1005



	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9
c0	2009	390	184	3	377	235	22	0	2	464
c1	305	1247	134	5	162	28	27	0	5	61
c2	12	79	1015	2	17	0	30	0	54	14
c3	234	224	40	397	324	20	32	0	11	19
c4	122	33	27	41	1078	1	1	0	3	55
c5	327	93	59	3	68	628	1	0	0	41
c6	169	268	146	3	332	45	550	0	23	76
c7	230	204	133	4	111	20	11	161	45	240
c8	176	67	485	1	209	21	53	2	108	80
c9	428	279	82	7	91	79	16	6	15	1567

Figure 25: Confusion matrices for the Abouelnaga dataset. c0-safe driving, c1-texting - right, c2-talking on the phone - right, c3-texting - left, c4-talking on the phone - left, c5-operating the radio, c6-drinking, c7-reaching behind, c8-hair and makeup, c9-talking to passenger

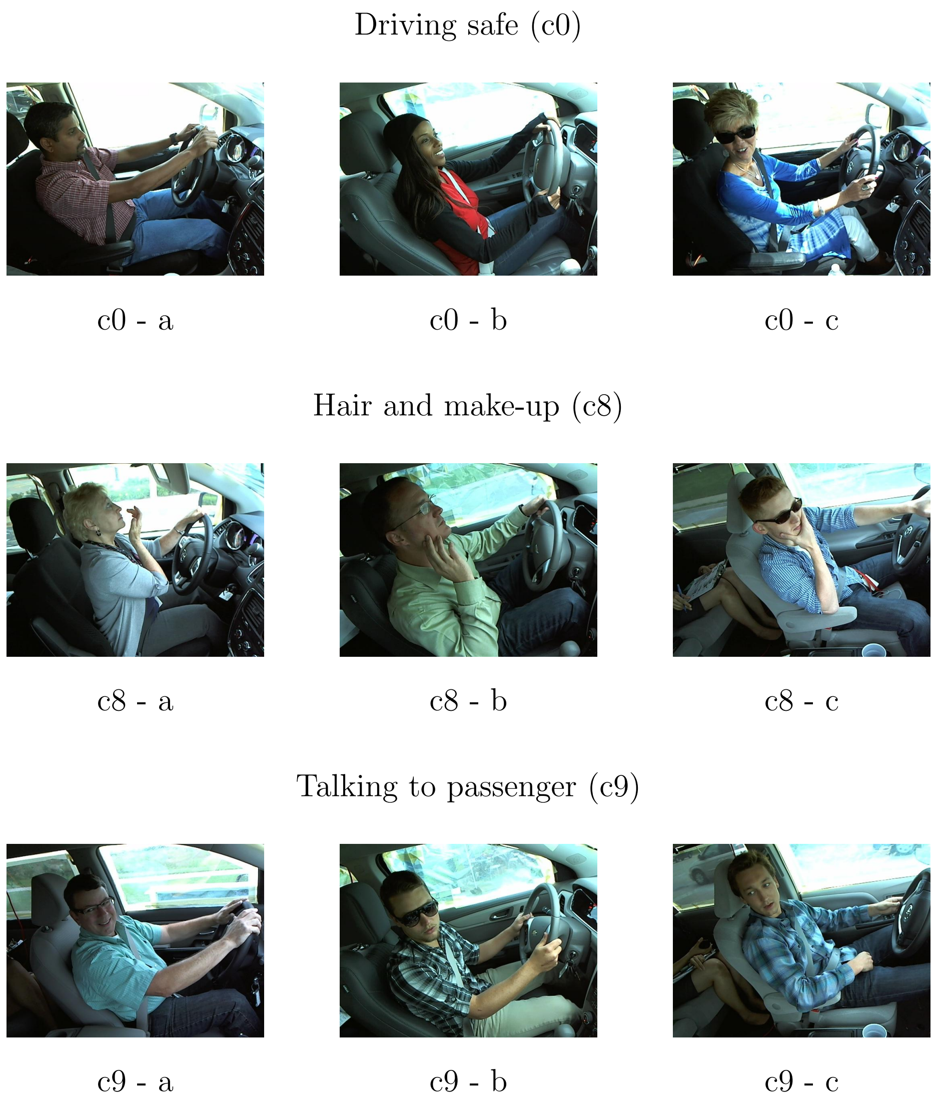


Figure 26: Sample images from classes c0, c8, c9. While images in the first column can be categorized without any ambiguity, images in the second and third columns are ambiguous and it is difficult to categorize those images even for a human observer. For example, we could say, that the woman in the picture *c0-b* is talking to a passenger based on her smile, but the image is labeled as *save driving*. We see that the picture *c8-c* labeled as *hair and make-up* distraction, which is not that obvious. The man might be just waiting at traffic lights and is not distracted at all. The image *c9-b* is labeled as *talking to passenger*, which might be the case. However, another possibility is that the driver is just looking out of the side window to have a better view of the road and the traffic conditions and thus is not distracted at all.

8.3.1 Combining two models' predictions

In this experiment, I will try to improve the accuracy of the so-far best model (ResNeXt50 from Section 8.2) by focusing on the better classification of classes c0, c8 and c9. From Figure 25 we see that the InceptionV3 network performs the best when it comes to the classification of these classes. For that reason, I tried to combine ResNeXt50's and InceptionV3's predictions in order to get the final prediction probability. I used ResNeXt's predictions as default predictions. If the class predicted by ResNeXt was c0 or c9, then I checked what is the Inception's prediction. If these two predictions were different and the probabilities differed by more than some threshold (0.22), I used the Inception's probabilities. This procedure performed the best from all the other combinations I tried and it resulted in 3.6% decrease of the loss score received on Kaggle.

8.3.2 Lowering probability of c8 and c9 classes

Another problem we can notice from Figure 25 is that networks are prone to classify c0 as c8 or c9, but not the opposite. Therefore properly lowering the probability of c8 or c9 in favor of other classes might improve accuracy. As a result of many experiments, the biggest improvement was achieved by lowering the probability of c8 and c9 classes according to the logic shown in the following piece of code.

```
# If the predicted class is c8 and its probability is higher than the
    second largest probability according to the following rule, then
    lower c8's probability by 70 %
if class_predicted == 'c8':
    if c8_probability - second_max_probability < 0.999:
        c8_probability = c8_probability - 0.7 * c8_probability

# Apply the same principle to c9 class if c9 is predicted.
if class_predicted == 'c9':
    if c9_probability - second_max_probability < 0.999:
        c9_probability = c9_probability - 0.2 * c9_probability
```

A further extension of this logic is a combination of multiple models in order to decide about the thresholds. I trained an InceptionV3 network in the same configuration as in Section 8.2 to distinguish between c0, c8 and c9 classes only. The hypothesis is that network trained to classify only these problematic classes might perform better at classifying these classes than the default network trained on all classes. By application of this technique, the loss score received on Kaggle was decreased by 0.23 %, which is not much when compared to the added complexity and computational requirements. The logic of this technique can be seen in the following piece of code.

```python
if class_predicted == 'c8':
    if c8_probability - second_max_probability < 0.999:
        # if the model trained on c0, c8 and c9 classes thinks the same
            as the default model, lower the probability by a lower
           amount. Otherwise lower the probability of c8 by larger
           percentage.
        if class_predicted_by_c0c8c9 == 'c8':
            c8_probability = c8_probability - 0.65 * c8_probability
        else:
            c8_probability = c8_probability - 0.7 * c8_probability

# Apply the same principle to c9 class.
if class_predicted == 'c9':
    if c9_probability - second_max_probability < 0.999:
        if class_predicted_by_c0c8c9 == 'c9':
            c9_probability = c9_probability - 0.28 * c9_probability
        else:
            c9_probability = c9_probability - 0.38 * c9_probability
```

Conclusion

By application of all of the techniques focusing on the better classification of classes c0, c8 and c9, the Kaggle loss score of the default ResNeXt50 model was decreased by 5.26 %.

8.4 Finding region of interest - driver detection

The original images from State Farm's dataset include the driver and also some surrounding which is not very important for a human observer when he decides about the driver's action. For that reason, I decided to detect the driver and blacken the surrounding. The hypothesis is that this helps to reduce noise in the image and thus the network can focus only on the important parts of the image. The driver is detected using the YOLOv3 algorithm [37]. The implementation is based on Huynh

Ngoc Anh's example [38]. The result of the driver detection algorithm is shown in Figure 27.

I trained the same model architectures as in Section 8.2. and adjusted the learning rates and the number of epochs in order to achieve the best results. The training configuration and performance on the State Farm dataset can be seen in Table 3. Comparison of results before and after driver detection reflects Table 4.

Architecture	learning rate	optimizer	epochs	Kaggle loss	val. accuracy (%)
VGG16	$1e^{-4}$	SGD	10	0.6387	90.00
InceptionV3	$1e^{-4}$	Adam	2	0.9916	81.95
ResNeXt50	$1e^{-5}$	SGD	7	0.5067	95.93
EfficientNetB0	$1e^{-3}$	SGD	9	0.6600	92.98
EfficientNetB4	$1e^{-3}$	SGD	10	0.6385	92.5

Table 3: Training configuration and results for State Farm dataset after driver detection.

Architecture	Kaggle loss before/after	Abouelnaga accuracy before/after (%)
VGG16	0.5327 / 0.6387	40.85 / 41.14
InceptionV3	0.6356 / 0.9916	50.61 / 45.49
ResNeXt50	0.4439 / 0.5067	37.6 / 36.29
EfficientNetB0	0.6506 / 0.6600	31.12 / 28.66
EfficientNetB4	0.6084 / 0.6385	28.88 / 26.76

Table 4: Comparison of results before and after driver detection.

From Table 4 we see that the only improvement that happened is VGG16's accuracy for Abouelnaga dataset and event the difference is not big. Moreover, the driver detection algorithm adds another extra layer of computational complexity and slows down the prediction process significantly. Therefore it can be concluded driver detection was not helpful in this particular case, but it can be helpful in other situations (e.g. when there is more surrounding noise around the driver).

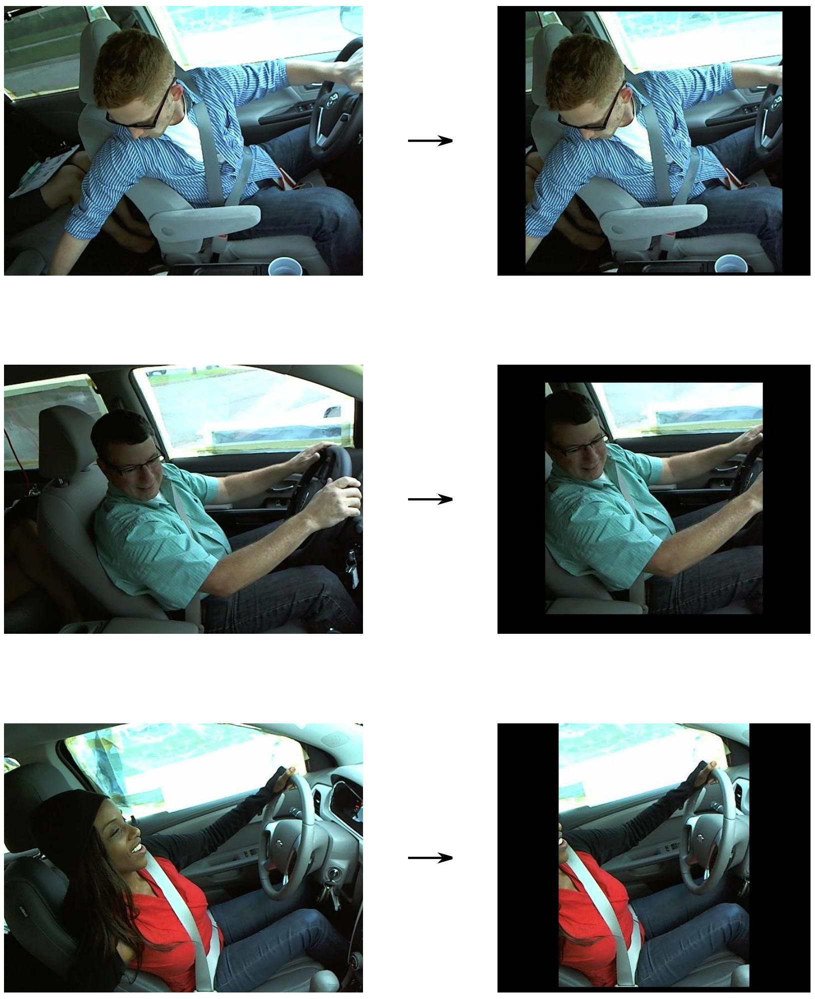


Figure 27: Application of the YOLOv3 algorithm to detect the driver. We can see that the detection does not always work perfectly (third image) which may cause some decrease in accuracy. Also, black padding is added to the images so that all of the images have the same resolution before entering the network.

9 Best model overview

The best model is considered the one which achieved the lowest loss score on the Kaggle evaluation site. This loss score is equal to multi-class logarithmic loss and is computed as shown at the beginning of Section 8. The scheme of this model is shown in Figure 28. The loss score achieved by this model is 0.4205, which is 5.26 % less than the loss score achieved by pure ResNeXt50 trained on augmented data with optimized parameters.

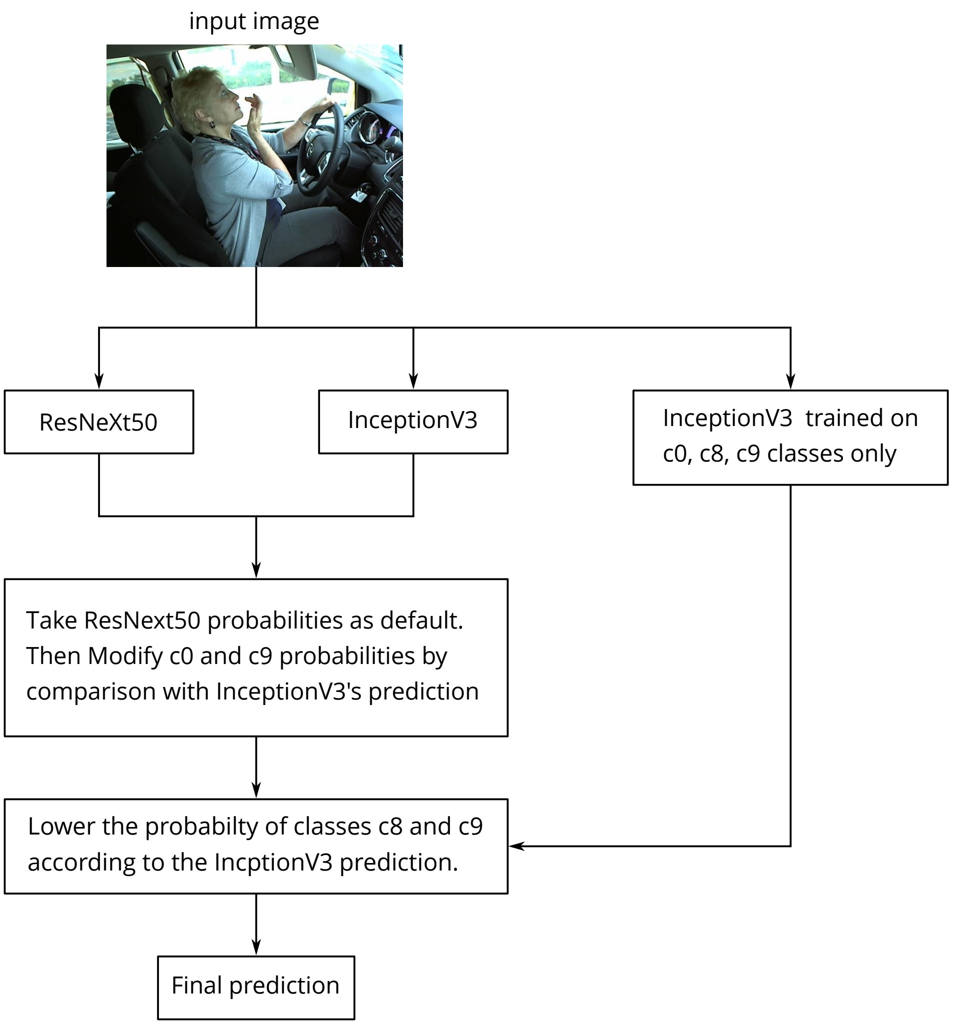

input image
ResNeXt50
InceptionV3
InceptionV3 trained on c0, c8, c9 classes only
Take ResNext50 probabilities as default. Then Modify c0 and c9 probabilities by comparison with InceptionV3's prediction
Lower the probabilty of classes c8 and c9 according to the IncptionV3 prediction.
Final prediction

Figure 28: A high-level overview of the model with the best performance on the State Farm dataset.

10 Conclusion

The main goal of this thesis was to build an end-to-end Driver Attention Warning system using convolutional neural networks and compare the performance of different neural network architectures like VGG, Inception, ResNeXt, and EfficientNet.

First, I obtained datasets that are necessary for training and testing such a system. Two different datasets were used - State Farm and Abouelnaga datasets. The State Farm dataset is much larger and was used to train the networks. Because the State Farm dataset is available only as a part of a Kaggle competition and thus the labels for test data are unknown, I also used the Abouelnaga dataset to do a more rigorous analysis of models' performance.

After the initial experiments which did not include any data manipulation or thorough optimization of hyper-parameters, all models but InceptionV3 network were trained with accuracy higher than 90 % for validation data. However, the problem of over-fitting was present. This problem was reduced by using appropriate augmentation techniques. The best results were achieved by zooming and shearing the training images. Another step to improve accuracy was parameter tuning. Choosing the right optimization algorithm and decreasing the learning rate accordingly as the training process progressed improved the networks' performance. Another technique that helped in some cases was training the last few layers first and after that training all layers of the network. The application of just mentioned techniques significantly improved the performance of all tested networks (error rate was reduced by 15 % up to 60 %). The best results were achieved by ResNeXt50 network architecture. Very good performance was achieved by VGG16 network which received the second-best score on Kaggle website. However, VGG16's biggest drawback is its slow training time. In terms of efficiency, EfficientNetB0 and EfficientNetB4 performed very well. These networks performed slightly worse than VGG16 but were significantly faster in terms of training and prediction times. EfficientNetB4 performed better than EfficientNetB0 in terms of accuracy but was slower to train, which is an expected result because EfficientNetB4 is deeper and wider than EfficientNetB0. While the InceptionV3 network did not perform the best on the State Farm dataset, it achieved the highest accuracy on the Abouelnaga dataset and therefore we can say this network generalizes the best.

By inspecting the confusion matrices it was discovered that the most difficult problem for all of the architectures is to distinguish between actions *driving safe*, *hair and make-up* and *talking to passenger*. This is due to the fact that the correct label is not always obvious even for a human observer. For example, sometimes it is hard to decide if the driver is talking to a passenger next to him or if the driver is just looking out of the side window. I tried to address this problem by combining multiple models' predictions. While the ResNeXt50 achieved the best overall accuracy, InceptionV3 network seemed to perform better when these problematic classes (c0, c8, c9) were presented. Another problem is the fact that the networks were prone to predict c8 or c9 class when the actual label was c0. This problem was addressed by decreasing the probability of classes c8 and c9 by an experimentally chosen value. By application of these techniques the error rate of so far best network (just optimized ResNeXt50) was lowered by another 5.26 %.

Possible improvements

As we could see in Section 8.3 , the most problematic is to correctly classify actions *driving safe*, *hair and make-up* and *talking to passenger.* It is difficult even for a human observer to classify these actions accurately when only static photos are presented. Therefore one possible improvement might be using video data instead of just static images. This would help us recognize better if the driver is talking to the passenger sitting next to him or is talking to someone using some hands-free device. This could be detected for example by detecting the movement of the driver's mouth. By using video data other types of distractions like sleepiness could be detected by analyzing the driver's facial movements. Another way to make the Driver Attention Warning system more robust and accurate is to use other than just image-based data. For example, by gathering the audio data driver's inattention could be detected more accurately.

References

[1] Traffic safety facts, 2019.

[2] M. Q. Khan and S. Lee. A Comprehensive Survey of Driving Monitoring and Assistance Systems. *Sensors (Basel)*, 19(11), Jun 2019.

[3] National highway traffic safety administration. `https://www.nhtsa.gov/risky-driving/distracted-driving`. Accessed: 2019-12-22.

[4] Centers for disease control and prevention. `https://www.cdc.gov/motorvehiclesafety/distracted_driving`. Accessed: 2019-12-23.

[5] Cognitive distraction. `https://www.aktriallaw.com/Articles/What-is-cognitive-distraction-and-why-is-it-so-dangerous-to-drivers.shtml`. Accessed: 2019-12-23.

[6] X. Zhang, N. Zheng, F. Wang, and Y. He. Visual recognition of driver hand-held cell phone use based on hidden crf. In *Proceedings of 2011 IEEE International Conference on Vehicular Electronics and Safety*, pages 248–251, July 2011.

[7] K. Seshadri, F. Juefei-Xu, D. K. Pal, M. Savvides, and C. P. Thor. Driver cell phone usage detection on strategic highway research program (shrp2) face view videos. In *2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 35–43, June 2015.

[8] B. Baheti, S. Gajre, and S. Talbar. Detection of distracted driver using convolutional neural network. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1145–11456, June 2018.

[9] Yehya Abouelnaga, Hesham M. Eraqi, and Mohamed N. Moustafa. Real-time distracted driver posture classification. 2017.

[10] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014.

[11] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks, 2016.

[12] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. 2019.

[13] Osvaldo Simeone. A brief introduction to machine learning for engineers, 2017.

[14] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. 2015.

[15] Yuhuang Hu, Adrian Huber, Jithendar Anumula, and Shih-Chii Liu. Overcoming the vanishing gradient problem in plain recurrent networks, 2018.

[16] Lu Lu, Yeonjong Shin, Yanhui Su, and George Em Karniadakis. Dying relu and initialization: Theory and numerical examples, 2019.

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 25, 01 2012.

[18] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, 2012.

[19] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[20] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization?, 2018.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[22] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[23] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[25] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks, 2016.

[26] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism, 2018.

[27] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016.

[28] State farm distracted driver detection. `https://www.kaggle.com/c/state-farm-distracted-driver-detection`.

[29] Tensorflow. `https://www.tensorflow.org`. Accessed: 2019-12-22.

[30] Keras. `https://keras.io`. Accessed: 2019-12-22.

[31] Numpy. `https://numpy.org`. Accessed: 2019-12-22.

[32] Pandas. `https://pandas.pydata.org`. Accessed: 2019-12-22.

[33] Matplotlib. `https://matplotlib.org`. Accessed: 2019-12-22.

[34] Metacentrum vo. `https://metavo.metacentrum.cz/`.

[35] Keras applications api. `https://keras.io/applications/`. Accessed: 2019-12-29.

[36] Efficientnet implementation in keras. `https://github.com/qubvel/efficientnet`. Accessed: 2019-12-29.

[37] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018.

[38] Yolov3 implementation in keras. `https://github.com/experiencor/keras-yolo3`. Accessed: 2020-20-1.

Appendix

```python
# Inception V3 architecture implementation
# ——————————————————————————————————————————
from keras.applications.inception_v3 import InceptionV3
from keras.layers import Dense, GlobalAveragePooling2D
from keras.models import Model

# Load model architecture without the top layer
# (this allows us to use custom input shape).
# Also load pre—trained ImageNet weights.
model_base = InceptionV3(weights='imagenet',
                         include_top=False,
                         input_shape=(640, 480, 3))
x = model_base.output

# Add global average pooling and fully connected layer on top of the
    base model.
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)

# Add softmax layer to get probabilities for each class as an output
predictions = Dense(units=10, activation='softmax')(x)

# Create complete model architecture
model = Model(inputs=model_base.input, outputs=predictions)

# Prevent first 249 layers from training. This causes that only the
    last 2 inception blocks will be trained.
n_lock = 249
for layer in model_base.layers[:n_lock]:
    layer.trainable = False

for layer in model_base.layers[n_lock:]:
    layer.trainable = True
```