Continuous Driver Activity Recognition from Short Isolated Action Sequences

Patrick Weyers and Anton Kummert
Departement of Electrical Engineering, Bergische Universität Wuppertal, Gaußstraße, Wuppertal, Germany

Keywords: Driver Activity Recongition, Action Recognition, Recurrent Neural Networks, Network Initialization, Limited Dataset, Online Action Recognition.

Abstract: Advanced driver monitoring systems significantly increase safety by detecting driver drowsiness or distraction. Knowing the driver's current state or actions allows for adaptive warning strategies or prediction of the driver's response time to take back the control of a semi-autonomous vehicle.
We present an online driver monitoring system for detecting characteristic actions and states inside a car interior by analysing the full driver seat region. With the proposed training method, a recurrent neural network for online sequence analysis is capable of learning from isolated action sequences only. The proposed method allows training of a recurrent neural network from snippets of actions, while this network can be applied to continuous video streams at runtime. With a mean average precision of 0.77, we reach better classification results on our test data than commonly used methods.

1 INTRODUCTION

Knowing what the driver is doing while driving is going to be an essential aspect of safety and infotainment systems in future cars. Driver inattention is a major factor in most traffic accidents (Dong et al., 2011). Therefore, different driver monitoring systems are developed to detect the driver's state (Weyers et al., 2018; Yan et al., 2016), activity (Weyers et al., 2019; Ohn-Bar et al., 2014) or gestures (Molchanov et al., 2016; Pickering, 2005).

Advanced safety driver assistant systems can adaptively draw the driver's attention back to the road depending on the driver's state or activity if necessary. For semi-autonomous cars, an estimation of the time the driver would need to take over the control of the car can be deduced e.g. from the body pose. Assistant systems like gesture recognition reduce the distraction caused by interaction with infotainment systems.

We present a method for recognizing the driver's state and detecting movements based on infrared (IR) amplitude image sequences of a time of flight camera. We calculate optical flow images between consecutive frames as input for a deep recurrent neural network to detect different movements in the car interior including, but not limited to the driver. Although our systems works on this special kind of input data, the principles can be applied to other 2d image streams.

While most comparable approaches train an online detection network on long data sequences, we use only short clips showing isolated actions or static scenes for training, while classifying continuous image sequences at runtime. With default training methods, this raises the problem of setting appropriate initial internal states of the recurrent network that allow it to function on continuous input.

We present a method for handling the initial internal state problem of recurrent neural networks while training on short clips of actions. In contrast to concatenating different short video clips to simulate longer videos with class transitions, our method relies on a memory module to effectively store and load relevant hidden states at training-time. The trained recurrent neural network is capable of classifying short clips of single actions as well as actions appearing in continuous image sequences at runtime.

2 RELATED WORK

Deep neural networks receive much attention in multiple types of recognition tasks. For video recognition tasks, works like (Wang et al., 2019; Simonyan and Zisserman, 2014; Carreira and Zisserman, 2017; Donahue et al., 2017; Liu et al., 2019) focus either on isolated sequences or on online action recognition

Weyers, P. and Kummert, A.
Continuous Driver Activity Recognition from Short Isolated Action Sequences.
DOI: 10.5220/0010185501580165
In *Proceedings of the 10th International Conference on Pattern Recognition Applications and Methods (ICPRAM 2021)*, pages 158-165
ISBN: 978-989-758-486-2

(Zolfaghari et al., 2018; Molchanov et al., 2016; Lin et al., 2019). Recurrent neural networks (RNN) are a neural network variant that unfolds in time and can be trained with Back Propagation Through Time (Werbos, 1990). Variants of RNNs that are commonly used are Long Short-Term Memory networks (LSTM) (Hochreiter and Schmidhuber, 1997) and Gated Recurrent Unit networks (GRU) (Cho et al., 2014). To focus on the movement aspect of actions, works like (Simonyan and Zisserman, 2014) use optical flow images for action classification and detection. They show that optical flow images of action sequences can improve sequence classification. While much work is done on weight initialization for artificial neural networks (Krizhevsky et al., 2012; Glorot and Bengio, 2010; He et al., 2015), very few publications focus on the initialization of the hidden states of recurrent neural networks.

When using recurrent neural networks for classification or prediction, the initial hidden states of the RNNs are usually set to zero or are randomly initialized and run until the effect of the initialization has no significant influence on the network output (Jaeger, 2002).

For long input sequences, where the entire input sequence does not fit into the memory while training the network, the sequences are cropped to multiple smaller clips. The hidden state for a training step is initialized with the the latest hidden state of the last training step (Zaremba et al., 2014). Another approach to initialize the hidden states of RNNs is to consider the initial state as a trainable parameter. The error that the network produces can then be propagated back to the initial state. This way, a preferable initial state can be learned (Mehri et al., 2016). Moreover, a neural network can be trained to calculate an initial state from a short input sequence. After the initial state prediction is completed, the recurrent network is fed with the subsequent input data (Mohajerin and Waslander, 2017).

3 DATA

The data examples used for training in this work are QVGA IR image sequences of a Time of Flight camera at 30 frames per second, with varying lengths between 20 and 200 frames per clip. They are recorded with a top down camera view in different vehicle interiors. The field of view covers the front row of each car. The participants were instructed to perform 13 different actions without specific explanations of how to perform them. The actions to distinguish are specific body movements of the driver. They include

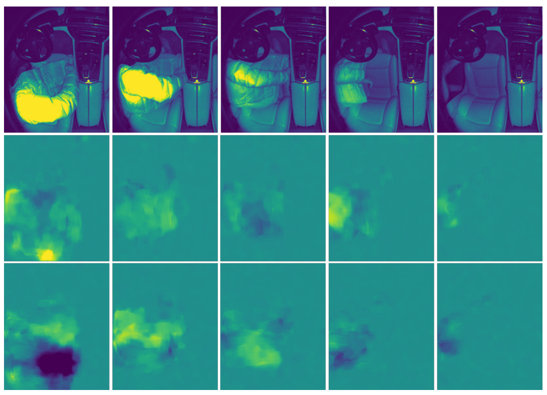


Figure 1: Data sequence example of a driver leaving the car. Top row: IR image sequence, middle and bottom row: corresponding optical flow sequence in x and y components.

movements where the driver leans from a driving position into various non-driving positions. These *leaning actions* are: Leaning towards steering wheel (*To Front*), Leaning towards the passenger seat (*Leaning Right*), Leaning towards the rear seats (*Lean Back*), Leaning back the from steering wheel (*From Front*), Leaning back from the passenger seat (*From Right*), Leaning back from the rear seats (*From Back*). Additionally, the following common driver actions are included as well: *Entering*, *Leaving*, Strapping the seat belt (*Strap*), Unstrapping the seat belt (*Unstrap*). Moreover, we added the following additional static classes: *Empty*, *In driving position*, Out of driving position leaning towards the steering wheel (*Out of Position Front*), Out of position towards the passenger seat (*Out of Position Right*), Out of position towards the rear seats (*Out of Position Back*). Each data example shows one of these action sequences performed in a driver seat of a car. Fig. 1 shows an example of one image sequence with corresponding optical flow images of a driver leaving the car.

The sequences were recorded in multiple different cars with multiple participants performing the mentioned actions.

The dataset used for training the networks consists of several short sequences, each representing one of the action classes or one of the static classes. For testing, longer sequences were recorded, covering multiple actions and static classes as well as transitions between these classes. The long sequences are used only for testing while the short sequences are used for training and validation.

For validation, about one fifth of the samples from each class are randomly selected to detect and prevent overfitting on the short training clips by augmenting the data and regularizing the training. The class distribution of the different examples in the dataset is shown in Fig. 2.

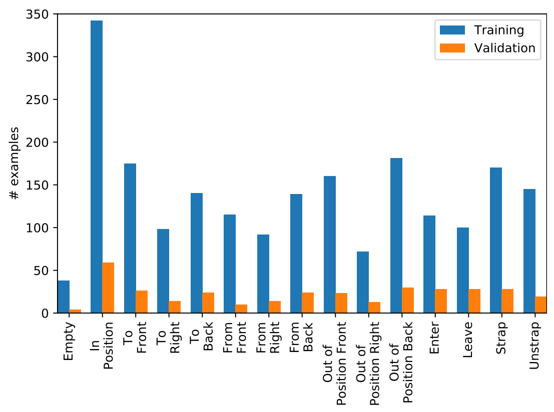


Figure 2: Dataset sample distribution.

The different classes are heavily imbalanced, and the number of samples is comparably small for a dataset used for deep learning. To reduce the influence of the different amount of data per class, the underrepresented class samples are upsampled to the second biggest number of class samples. The maximum number of class samples is not used, because we found that the second class '*In position*' has a higher variation in its imagery and needs more different examples than the other classes.

4 ACTION RECOGNITION

Our approach to recognise different actions in online sequences relies on preprocessed input data, a common network architecture for action recognition with RNNs and a special method of initializing the hidden states while training the network.

4.1 Input Data

Dense optical flow image sequences were calculated between consecutive IR frames with the Farneback method (Farnebäck, 2003).

For training and validation, short sequences of either one action class or one static class are used. For testing longer sequences with multiple examples of actions, static examples and transitions between those are applied. At runtime, the system operates on continuous video data.

4.2 Network Structure

To classify image sequences, we trained a combination of a convolutional neural network (CNN) and a RNN in a many to many fashion. This means that the network calculates one classification result for each input frame. Fig. 3 shows the basic network architecture.

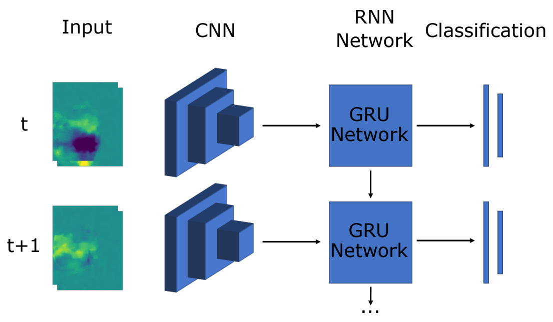


Figure 3: Network architecture.

In the first part of the network, convolutional layers extract features from the inputs. These spatial features are the input to a recurrent neural network, which extracts time series features. At the end of the network, fully connected layers classify the features extracted from each time step.

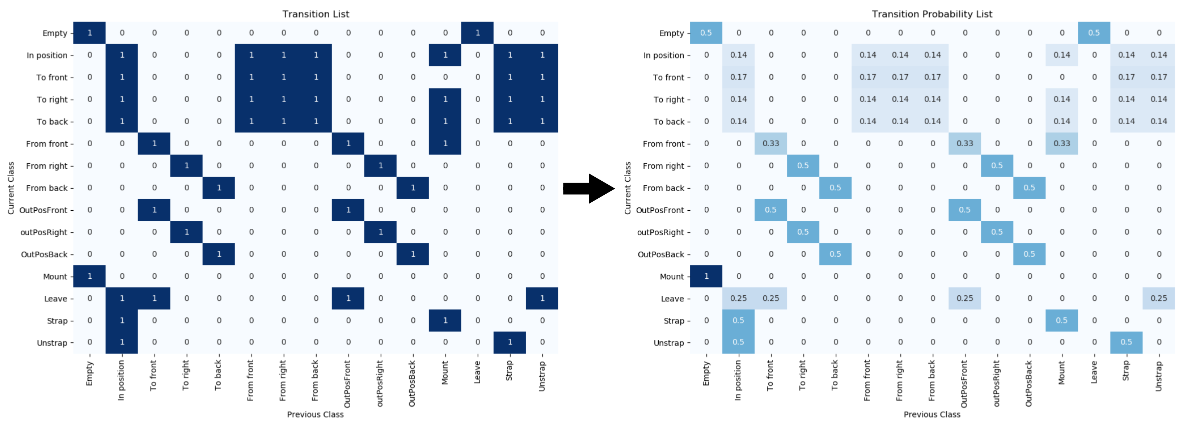
While training, a state handling approach is used to initialize the hidden state at the beginning of each example sequence. At runtime, the hidden state needs to be initialized only once at the very beginning of the live stream.

4.3 Recurrent State Handling

In order to create a continuous classification system, untrimmed continuous training data examples are preferable. However, this data is often not available or too expensive to annotate, which makes it hard to combine data from different sources, and causes a risk of overfitting to the temporal action combinations present in the training data. In contrast, our dataset used to train the network consists of multiple short clips, each representing one example of one of the proposed classes.

To use this data to train a system for continuous video classification, one can concatenate logical matching clips (in terms of classes) to produce longer sequences with class transitions. This not only makes the training process more ineffective, as additional data needs to be loaded and processed, but reduces the flexibility to learn transitions between classes as the network can only learn from directly pre-calculated states.

With our approach to train a system for continuous video classification with the given data, we store the hidden states of each example of the recurrent unit at each training step. The stored hidden states are used to initialize the recurrent unit in future training steps. Older hidden states from earlier training steps



Transition List

Current Class \ Previous Class	Empty	In position	To front	To right	To back	From front	From right	From back	OutPosFront	outPosRight	OutPosBack	Mount	Leave	Strap	Unstrap
Empty	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0
In position	0	1	0	0	0	1	1	1	0	0	0	1	0	1	1
To front	0	1	0	0	0	1	1	1	0	0	0	0	0	1	1
To right	0	1	0	0	0	1	1	1	0	0	0	1	0	1	1
To back	0	1	0	0	0	1	1	1	0	0	0	1	0	1	1
From front	0	0	1	0	0	0	0	0	1	0	0	1	0	0	0
From right	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0
From back	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0
OutPosFront	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0
outPosRight	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0
OutPosBack	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0
Mount	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Leave	0	1	1	0	0	0	0	0	1	0	0	0	0	0	1
Strap	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0
Unstrap	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0

Transition Probability List

Current Class \ Previous Class	Empty	In position	To front	To right	To back	From front	From right	From back	OutPosFront	outPosRight	OutPosBack	Mount	Leave	Strap	Unstrap
Empty	0.5	0	0	0	0	0	0	0	0	0	0	0	0.5	0	0
In position	0	0.14	0	0	0	0.14	0.14	0.14	0	0	0	0.14	0	0.14	0.14
To front	0	0.17	0	0	0	0.17	0.17	0.17	0	0	0	0	0	0.17	0.17
To right	0	0.14	0	0	0	0.14	0.14	0.14	0	0	0	0.14	0	0.14	0.14
To back	0	0.14	0	0	0	0.14	0.14	0.14	0	0	0	0.14	0	0.14	0.14
From front	0	0	0.33	0	0	0	0	0	0.33	0	0	0.33	0	0	0
From right	0	0	0	0.5	0	0	0	0	0	0.5	0	0	0	0	0
From back	0	0	0	0	0.5	0	0	0	0	0	0.5	0	0	0	0
OutPosFront	0	0	0.5	0	0	0	0	0	0.5	0	0	0	0	0	0
outPosRight	0	0	0	0.5	0	0	0	0	0	0.5	0	0	0	0	0
OutPosBack	0	0	0	0	0.5	0	0	0	0	0	0.5	0	0	0	0
Mount	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Leave	0	0.25	0.25	0	0	0	0	0	0.25	0	0	0	0	0	0.25
Strap	0	0.5	0	0	0	0	0	0	0	0	0	0.5	0	0	0
Unstrap	0	0.5	0	0	0	0	0	0	0	0	0	0	0	0.5	0

Figure 4: Transition lists for proposed classes. Left: Transition list to show the possible class transitions, right: Transition probability list to show the probabilities of transitions between classes.

are deleted, as they get less relevant for newer training steps.

As not all stored hidden states are meaningful to use as initialization for each example, each class only gets initial states from reasonable previous classes. This reasonable previous classes are predefined in a transition table, where all reasonable class transitions are stored. From this transition list a transition probability list is calculated to get the transition probabilities between the classes. The transition lists used for this work is shown in Fig. 4. The transition list shows which classes can precede which class, while the transition probability list shows the probability for each class to precede another class. An example showing an *Empty* seat can only be preceded by an example showing an *Empty* seat, or an example showing the class *Leave*. The probability for each of these two classes to precede this current example is 0.5 respectively, as only examples from two classes can precede the class *Empty*. With this transition tables, we introduce a logic to select initial hidden states to initialize the recurrent units for each training example. This means that only hidden states resulting from reasonable previous examples are used to initialize the recurrent units. Reasonable previous examples are examples that can precede the current examples in the real continuous world. For example, using a hidden state resulting from an *Empty* sample where only an empty driver seat is visible is a reasonable initialization for a sample where someone enters the car. On the other hand, a hidden state resulting from an example where someone is already sitting on the driver seat is rather not relevant for the examples of someone entering the car. The hidden state, used for initializing the recurrent unit at the beginning of a training step can either be a single reasonable hidden state or the mean of multiple reasonable hidden states.

Fig. 5 illustrates this state handling in the training phase of the network.

The hidden states of each time step of the recurrent network are stored inside a *State Memory*. This *State Memory* has the capacity to store the hidden states for each training example once. Like this, we prevent the network to learn from old hidden states, which become obsolete due to the training progress. For each following training sequence one reasonable hidden state is selected with the probabilities given by the transition probability table from all stored hidden states in the *State Memory*. For validation, one initial hidden state is selected with the probabilities given by the transition probability table from all reasonable hidden states stored in the validation procedure.

5 TRAINING

The networks presented in this paper are trained using the AdamW optimizer (Loshchilov and Hutter, 2019) with Back Propagation Through Time (Werbos, 1990).

5.1 Augmentation

To get more variation to the data, we augmented the input data. We randomly rotate and translate the IR images within certain limits. Each frame of one single sequence is augmented in the same way to prevent temporal artifacts in a rotation and translation stable scene. This means that each frame from one sequence is rotated with the same angle and translated the exact same way. If the frames are not augmented in the same way, movement between frames would appear where no real movement is happening.

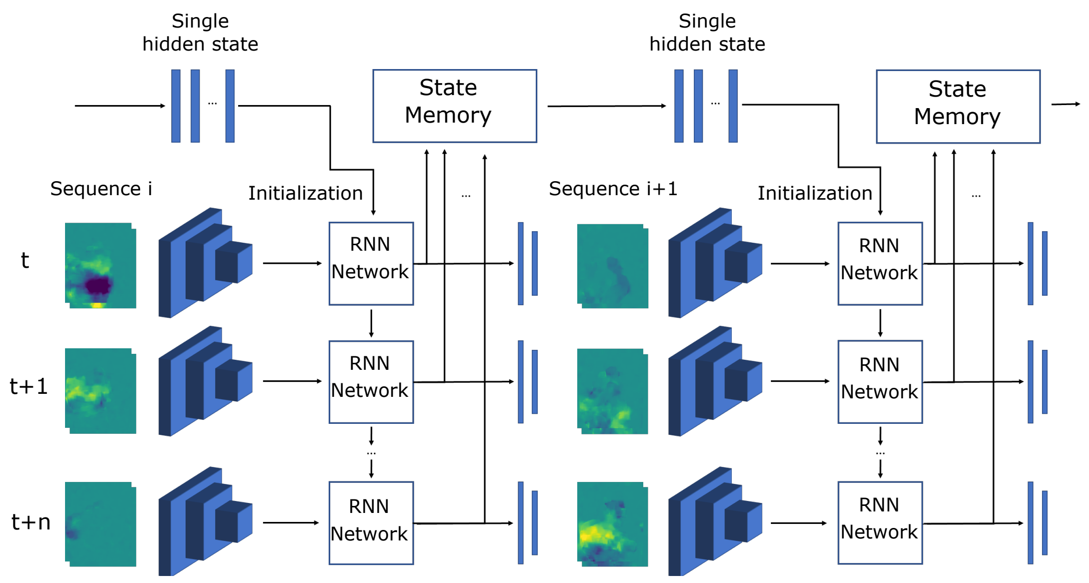


Figure 5: State handling in the training and validation phase with a single hidden state: For each time step spatial features are extracted with a CNN. The recurrent network is initialized with a previous hidden state from the state memory. Temporal features are extracted with an initialized recurrent network. The new hidden state for each time step is stored to the state memory and the temporal features are classified.

To augment the time component of the sequences, frames at different time steps are duplicated or dropped, resulting in slowing down or speeding up the parts of the sequence respectively (Weyers et al., 2019).

5.2 State Handling

While training, we store the hidden states as well as the corresponding label of the examples from the latest training step. The hidden states for the following training step are initialized with relevant stored hidden states. Information about relevant hidden states and the transition probabilities are stored in the previous shown transition lists. The used previous class is randomly selected from all relevant previous classes. As soon as new hidden states for a class are calculated, previously stored hidden states are replaced with the new ones. If no hidden state for a randomly selected previous class exists, the hidden state is initialized with entirely random values. This way, we ensure that only up to date hidden states are used for the initialization and old hidden states, resulting from earlier training steps, are discarded as they might not represent valid hidden states anymore.

The proposed state handling method is used for training the network. For testing, the hidden states of the recurrent units are initialized at the very beginning of each test sequence. Afterwards, the hidden state is calculated by the recurrent unit. At runtime, the hidden state is initialized at the very beginning of the streaming, and from then on evolves continuously without any artificial resets or the like.

6 RESULTS

For testing, we recorded 30 long sequences showing multiple action classes, static classes and transitions between those. A snippet of an example for a test sequence is shown in Fig. 6. To evaluate our state handling approach, we compare our results to the results of networks trained with two standard training approaches, namely initializing the hidden states of the recurrent network with zeros or with random values, and concatenated input data. While testing the networks trained with zero and random initialization we test different strategies to reset the hidden states at runtime. The first strategy is to not reset the hidden state at all. The other strategies are resetting the hidden state every n_{reset}, where we choose n_{reset} to be the minimum sequence length, the maximum sequence length and the mean sequence length of the training examples.

Moreover we train networks on concatenated sequence examples. For this a reasonable previous example is randomly selected in the same way as with the state handling approach. The selected sequence is concatenated with the current sequence to simulate the transition of classes directly in the input data. Beside using optical flow images as input to our networks, we trained networks with grayscale images

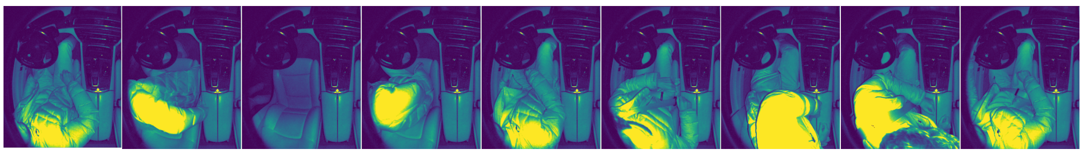


Figure 6: Snippet of a test sequence. From left to right: *In position*, *Leaving*, *Empty*, *Entering*, *In position*, *Leaning backwards*, *Out of position back*, *Leaning back from back*, *In position*.

and depth images as input to see the impact of the input format to the results. Moreover we trained networks with two consecutive grayscale images per timestep as input to the network. All compared networks have the same architecture, except for the input layer, as grayscale and depth images have only one channel and the optical flow images and two consecutive grayscale images have two input channels. As metric, we use the macro F1-scores:

$$F_{1macro} = \frac{1}{|L|}\sum_{l\in L} F_1(y_l, \hat{y}_l) \tag{1}$$

with L as the set of labels, y_l as the set of prediction pairs for label l and $\hat{y}_l$ as the set of true pairs for label l, and the balanced accuracy scores as the mean of class-wise accuracy:

$$ACC_{balanced} = \frac{1}{|L|}\sum_{l\in L} \frac{TP_l}{TP_l FN_l} \tag{2}$$

with TP_l as the true positive classifications and FN_l as the false negative classifications, as well as the mean average precision and the average precision per class. We use the macro F1-score and the balanced accuracy as metrics to evaluate all classes equally, as the test set is imbalanced. This results from the fact that some actions take longer to perform and therefore are represented in more frames. Moreover some actions are represented more often than others as they occur naturally between actions.

Table 1 shows the F1-scores and balanced accuracy scores of the trained networks for the different input formats, the initialization and reset strategies.

Comparing the network results on the test set, the networks trained with explicit state handling trained on optical flow images and consecutive grayscale images exceed all other approaches in terms of F1-scores and balanced accuracy. The networks trained with zeros and random initialization result in the worst scores when not reset. When resetting these networks the performance on the test set rises as the networks regularly get new artificial starting points. This resembles the training procedure more than the strategy where the hidden states are not reset. The network trained on consecutive sequences scores better than the networks trained with zero or random initialization when trained on flow image sequences, though not better than those trained with the state handling approach. For the single grayscale and depth images the best reset strategy results in similar scores compared to our approach. When using consecutive grayscale frames or optical flow images as input to the network, our approach works best in terms of F1-score and balanced accuracy. The CNN can extract more valuable features for the RNN from optical flow images or consecutive frames. The single image input networks might score better with bigger networks. However, training bigger networks resulted in early overfitting to the available training data and would require additional data to avoid this effect. This applies to networks that were increased in width and depth and for implemented two-stream like networks similar to the work of (Simonyan and Zisserman, 2014).

Table 2 shows the mean average precision and av-

Table 1: Macro F1-score and balanced accuracy score comparison for the networks trained on the different initialization and reset strategies and the different inputs.

	IR		Depth		IR 2 frame		Flow	
Initialization method	F1	accuracy	F1	accuracy	F1	accuracy	F1	accuracy
States (proposed)	**0.42**	**0.41**	**0.43**	**0.43**	**0.62**	**0.64**	**0.66**	**0.71**
Zeros, no reset	0.19	0.21	0.10	0.11	0.17	0.19	0.16	0.18
Zeros, reset min	0.34	0.37	0.38	0.40	0.45	0.47	0.42	0.48
Zeros, reset max	0.40	0.40	0.42	**0.43**	0.45	0.45	0.46	0.50
Zeros, reset mean	0.30	0.30	0.30	0.28	0.34	0.33	0.36	0.38
Random, no reset	0.24	0.24	0.11	0.14	0.22	0.23	0.26	0.30
Random, reset min	0.31	0.35	0.32	0.35	0.44	0.47	0.40	0.47
Random, reset max	0.40	**0.41**	0.36	0.39	0.47	0.48	0.46	0.52
Random, reset mean	0.32	0.32	0.27	0.27	0.37	0.36	0.39	0.43
Concatenated	0.30	0.32	0.36	0.36	0.47	0.50	0.58	0.64

Table 2: Mean average precision and average precision per class scores of flow networks for different initialization, reset and training strategies.

Initialization method	mAP	Empty	In position	To front	To right	To back	From front	From right	From back	Front	Right	Back	Enter	Leave	Strap	Unstrap
States (proposed)	**0.77**	**0.92**	**0.92**	**0.68**	0.63	0.57	0.75	**0.70**	**0.74**	**0.84**	**0.80**	0.71	0.75	**0.76**	**0.64**	**0.58**
Zeros, no reset	0.33	0.51	0.64	0.14	0.12	0.13	0.06	0.14	0.11	0.20	0.41	0.22	0.53	0.46	0.42	0.16
Zeros, reset min	0.41	0.31	0.66	0.56	**0.65**	0.36	0.47	0.41	0.42	0.35	0.57	0.21	0.26	0.19	0.22	0.15
Zeros, reset max	0.41	0.51	0.71	0.21	0.18	0.19	0.26	0.27	0.21	0.35	0.42	0.29	0.63	0.36	0.42	0.23
Zeros, reset mean	0.51	0.50	0.78	0.36	0.44	0.33	0.53	0.45	0.45	0.39	0.58	0.30	0.72	0.30	0.33	0.37
Random, no reset	0.24	0.43	0.60	0.39	0.20	0.22	0.30	0.55	0.14	0.26	0.47	0.33	0.75	0.44	0.33	0.14
Random, reset min	0.35	0.30	0.64	0.54	**0.65**	0.40	0.51	0.40	0.40	0.30	0.50	0.22	0.28	0.18	0.19	0.13
Random, reset max	0.35	0.40	0.70	0.35	0.33	0.28	0.41	0.47	0.24	0.35	0.56	0.49	0.69	0.41	0.36	0.31
Random, reset mean	0.43	0.43	0.72	0.48	0.52	0.44	0.60	.52	0.45	0.34	0.63	0.40	0.66	0.36	0.36	0.31
concatenated	0.67	0.87	0.87	0.61	0.58	**0.63**	**0.76**	0.65	0.61	0.80	0.63	**0.73**	**0.90**	0.57	0.53	0.54

erage precision per class for the networks with optical flow images as inputs for the different initialization and reset strategies. Our state handling approach results in the highest mean average precision and average precision per class with a significant margin for most classes. In Fig. 7 we show the classification results for each frame of the test sequences in a confusion matrix. This matrix demonstrates that most frames are classified correctly. The biggest confusion occurs between similar classes like *To right* and *To back*, where there is some similarity to phases of other actions. This can be seen in the classification results for the class *Empty*, where the 20% wrongly classified frames belong to the classes *Mount* and *Leave*, where images of nearly empty seats are visible.

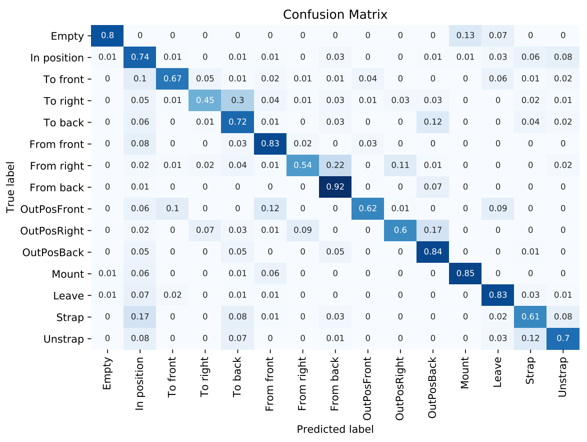


Figure 7: Confusion Matrix for test results of the state handling approach.

7 CONCLUSION

We presented an adaptive initial state handling method for training online action classification recurrent neural networks on short action clips. With this method, the knowledge about the expected class transitions can be used to enhance the training of a recurrent scene classifier for online action classification, when only isolated action examples are available. In contrast to other common initialization methods like zero or random initialization, the hidden state does not need to be reset during actual execution, as class transitions that can occur at runtime are part of the training procedure, even if such transitions are not directly included in the short video clips. Simulating the transitions by concatenating sequences resulted in better classification results than training on single sequences with random or zero initialization. However, our approach outscores this intuitive approach not only with respect to classification results, but also speeds up training, as only one example sequence per training step needs to be loaded and calculated instead of two. When using optical flow images or consecutive frames as input to the network, the network does not need to be trained with a big data set like most deep learning approaches.

ACKNOWLEDGEMENTS

The authors thank Aptiv and in particular David Schiebener, Xuebing Zhang and Alexander Barth for supporting this work with fruitful input and data.

With financial support from the state government of North Rhine-Westphalia.

REFERENCES

Carreira, J. and Zisserman, A. (2017). Quo vadis, action recognition? a new model and the kinetics dataset. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4724–4733.

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734.

Donahue, J., Hendricks, L. A., Rohrbach, M., Venugopalan, S., Guadarrama, S., Saenko, K., and Darrell, T. (2017). Long-term recurrent convolutional networks for visual recognition and description. *IEEE Trans. Pattern Anal. Mach. Intell.*, page 677–691.

Dong, Y., Hu, Z., Uchimura, K., and Murayama, N. (2011). Driver inattention monitoring system for intelligent vehicles: A review. *IEEE Transactions on Intelligent Transportation Systems*, 12(2):596–614.

Farnebäck, G. (2003). Two-frame motion estimation based on polynomial expansion. In *Proceedings of the 13th Scandinavian Conference on Image Analysis*, page 363–370. Springer-Verlag.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, page 1026–1034.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9:1735–80.

Jaeger, H. (2002). Tutorial on training recurrent neural networks, covering bppt, rtrl, ekf and the echo state network approach. *GMD-Forschungszentrum Informationstechnik, 2002.*, 5.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc.

Lin, J., Gan, C., and Han, S. (2019). Tsm: Temporal shift module for efficient video understanding. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 7083–7093.

Liu, J., Shahroudy, A., Perez, M., Wang, G., Duan, L.-Y., and Kot, A. C. (2019). Ntu rgb+d 120: A large-scale benchmark for 3d human activity understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Loshchilov, I. and Hutter, F. (2019). Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Mehri, S., Kumar, K., Gulrajani, I., Kumar, R., Jain, S., Sotelo, J., Courville, A., and Bengio, Y. (2016). Samplernn: An unconditional end-to-end neural audio generation model.

Mohajerin, N. and Waslander, S. L. (2017). State initialization for recurrent neural network modeling of time-series data. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2330–2337.

Molchanov, P., Yang, X., Gupta, S., Kim, K., Tyree, S., and Kautz, J. (2016). Online detection and classification of dynamic hand gestures with recurrent 3d convolutional neural networks. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4207–4215.

Ohn-Bar, E., Martin, S., Tawari, A., and Trivedi, M. M. (2014). Head, eye, and hand patterns for driver activity recognition. In *2014 22nd International Conference on Pattern Recognition*, pages 660–665.

Pickering, C. A. (2005). The search for a safer driver interface: a review of gesture recognition human machine interface. *Computing Control Engineering Journal*, 16(1):34–40.

Simonyan, K. and Zisserman, A. (2014). Two-stream convolutional networks for action recognition in videos. In *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc.

Wang, L., Xiong, Y., Wang, Z., Qiao, Y., Lin, D., Tang, X., and Van Gool, L. (2019). Temporal segment networks for action recognition in videos. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41:2740–2755.

Werbos, P. (1990). Backpropagation through time: what does it do and how to do it. In *Proceedings of IEEE*, volume 78, pages 1550–1560.

Weyers, P., Barth, A., and Kummert, A. (2018). Driver state monitoring with hierarchical classification. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 3239–3244.

Weyers, P., Schiebener, D., and Kummert, A. (2019). Action and object interaction recognition for driver activity classification. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 4336–4341.

Yan, C., Coenen, F., and Zhang, B. (2016). Driving posture recognition by convolutional neural networks. *IET Computer Vision*, 10(2):103–114.

Zaremba, W., Sutskever, I., and Vinyals, O. (2014). Recurrent neural network regularization.

Zolfaghari, M., Singh, K., and Brox, T. (2018). Eco: Efficient convolutional network for online video understanding. In *The European Conference on Computer Vision (ECCV)*.