Cognitive activity recognition by analyzing eye movement with convolutional neural networks

BOB BROCKBERND
Supervisor(s): GUOHAU LAN, LINGYU DU
EEMCS, Delft University of Technology, The Netherlands

A Dissertation Submitted to EEMCS faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering

Abstract

This research proposes a novel method to classify cognitive behavior based on eye-movement data. Most state-of-the-art approaches use conventional machine learning techniques needing manual feature extraction. This experiment explores the possibility of applying deep learning algorithms to cognitive activity recognition for feature extraction and classification of eye-movement data. Convolutional neural networks will be explored in particular. Two neural networks are proposed and optimized using hyperparameter tuning. This research shows that convolutional neural networks can indeed perform cognitive activity recognition. Some neural networks significantly outperform the state-of-the-art methods for known subjects. However, further research is needed to improve performance in classifying activities for unknown subjects.

1 Introduction

Human activity recognition (HAR) has become more important as the demand for seamless human-computer interaction has increased [1], [2]. Human activities, from physical to cognitive tasks, can be recognized with different types of sensor data, like motion data [3] and visual data [1]. This research focuses on using eye-movement data to predict certain cognitive activities such as reading, writing, and browsing the web. In other words: Gaze-based activity recognition.

Motivation

In modern cars, new techniques are applied to recognize and prevent driver fatigue. By using eye-tracking technologies cars can alarm the driver when attention is lost [2]. This could make driving safer and prevent avoidable accidents. Improving and expanding our knowledge about recognizing cognitive context can lead to better and more reliable systems.

Related works and challenges

Related works indicate that there is a correlation between cognitive tasks and eye movement [4], [5]. Most recognition techniques are based on conventional machine learning techniques like support vector machines or hidden Markov models [5].

As seen in [6, p. 424] one of the challenges with gaze-based human activity recognition lies in the "Heterogeneity in human visual behaviour. Human visual behaviours are heterogeneous across subjects, visual stimuli, and eye-tracking devices." This makes feature extraction a complex task and potentially has an impact on accuracy. Another challenge is the lack of sufficient training data [6]. Collecting eye-movement data has proven to be difficult due to privacy concerns.

Convolutional neural networks (CNN) could solve the complex feature extraction issue since CNNs are designed to learn the features during training. This eliminates the need for manually made features which often require deep knowledge of the data. In addition, CNNs have been proposed before for gaze-based activity recognition [7]. However, a very limited dataset was used only containing two activities. Moreover, in other HAR fields CNNs have created accurate results with small multi-class datasets [8]–[10].

This suggests that CNNs could potentially improve the state-of-the-art gaze-based activity recognition performances.

Research Question

The question this research aims to answer is:

- Can a convolutional neural network classifier be used for gaze-based activity recognition?

To tackle this question the following sub-questions have to be answered.

- What CNN architecture is best suited for gaze-based activity recognition?
- Which CNN hyperparameters perform best for gaze-based activity recognition?
- How do the found CNN performances compare to other machine learning techniques?

2 Methodology

The following methodology is proposed to answer the research question. First, data is required for training the CNNs. The three datasets used in this research are described in section 2.1. Next, preprocessing: although neural networks can work with raw data [11] they can benefit from some preprocessing which is discussed in section 2.2. Following is the search for suitable CNN architectures, the two baseline architectures are presented in section 2.3.The two architectures need tuning to tailor them to the datasets. This process, called hyperparameter tuning, is discussed in section 2.4. Lastly, to measure the performance of each set, cross-validation is applied, see section 2.5.

2.1 Dataset description

The following three datasets are used for training and validating the neural networks.

- **Desktop activity** is a dataset consisting of eight subjects [6]. Four male and four female, between the age of 24 and 35. The performed activities are: browsing the web, playing a game, reading, using a search engine, watching a video, and writing an essay. The data has been collected with the Pupil Core Eyetracker [12].
- **Sedentary activity** is data collected from 24 subjects where 16 participants are male and 8 female [13]. All participants are between the age of 24 and 48. The performed cognitive activities are reading, watching a movie, browsing the web, using a search engine, playing a game, writing code, debugging code, and interpreting code output. The data has been collected with the Tobii Pro X2 [14].
- **Japanese Document** is the last dataset used for this research, consisting of eight subjects [15]. Four males and four females ageing between 21 and 32. The activities are reading five different types of reading materials: a novel, a fashion magazine, a manga, a newspaper, and

a textbook. The data has been collected with the SMI wearable eye-tracking glasses [16].

All data has been recorded with a rate of 30Hz and is shaped as a 2xN array where each sample N contains an X and Y coordinate of the gaze.

2.2 Preprocessing

Preprocessing of the data is depicted in figure 1. It starts with removing outliers. This is done by taking the mean of the signal and removing data points that are further than two times the standard deviation from the mean.

Next, the data is normalized per subject per activity. Normalization shifts and scales the signal such that the smallest data point equals 0 and the largest equals 1. This can be achieved by subtracting all values with the minimum of the signal and dividing all resulting values by the resulting maximum.

Then, the normalized data is turned into fixed-length frames using a sliding window. The sliding window also helps in generating more data, the length of the window is a controlled hyperparameter which will be determined in the hyperparameter tuning part of the research. A stride of 30 samples is applied, i.e. the sliding window moves in steps of 1 second.

When training a 2-dimensional CNN the frames are turned into 2D black and white images, where the X and Y coordinates are drawn on a black canvas and the gaze points are connected to incorporate the saccades. A visual example is depicted in figure 2. The resolution of this input image and the width of the line connecting the points, are as well controlled hyperparameters.

During this preprocessing step the time component is lost: the relative angle between two saccades is made clear but the absolute direction is not. To overcome this, the connecting lines on the canvas increase in intensity over time. This feature can be enabled or disabled by the tuning algorithm, which makes it a Boolean hyperparameter.

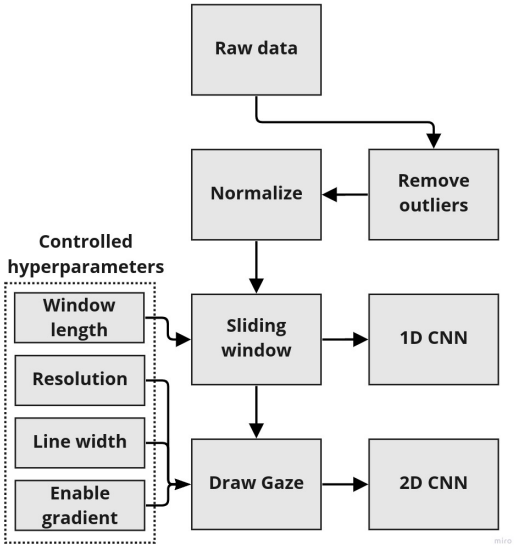


Figure 1: Preprocessing for 1D CNNs has three steps: removing outliers, normalization and a sliding window. A 2D CNN requires one additional preprocessing step: drawing the gaze on an image. The sliding window and gaze drawing are controlled by hyperparameters.

2.3 CNN Architectures

A convolutional neural network is designed by chaining specific layers, these are the building blocks of a CNN architecture. The design process of this research started with two known architectures. Variations are created by adding and removing layers. The best-performing architectures are selected through training and validation.

The building blocks for a CNN

The neural network layers used in this research to build CNNs are explained below.

- **Convolution layers:** These layers use learnable convolution kernels to recognize patterns in the input [11]. It produces a feature map per kernel. Together with the max-pooling layer, it represents the feature extraction part of the network.
- **Max pooling layers:** These layers devide their input in regions of the pool size and pick the maximum value in each region, reducing dimensionality and allowing for more abstract pattern recognition [11].

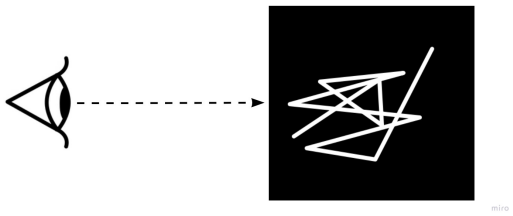


Figure 2: For a 2D CNN, the gaze points are drawn and connected on a black and white image. The line thickness and image resolution are variable and optimized in the hyperparameter tuning part of this research.

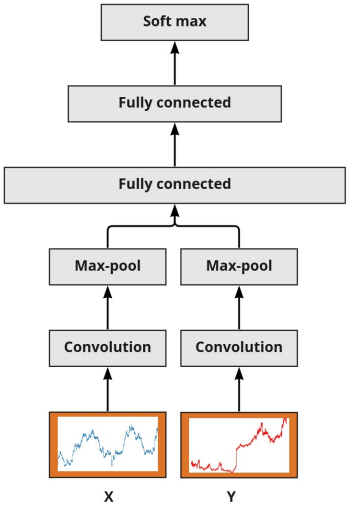


Figure 3: This CNN uses one-dimensional convolution to extract features in X and Y movements separately. After convolving and pooling the data once, the features are merged and used as input for the first fully connected layer. Next, the data flows through a smaller fully connected layer. Lastly, the input is classified using a soft-max classifier.

- **Dropout layers:** These layers are often applied after the activation function of a fully connected or convolutional layer. Dropout layers prevent overfitting by randomly disabling neurons during training [17].
- **Fully connected layers:** Each neuron in a fully connected layer is connected to all neurons from the previous layer [11]. These layers are used for classification.

Baseline architectures

This research has two CNN architectures as starting points. The first architecture, shown in figure 3, uses one-dimensional convolutions and is based on the network used in [9]. This network has been applied to accelerometer data consisting of multiple 1D signals, being very similar to eye-movement data. It extracts features separately from each signal (X, Y and Z) by applying one convolution layer and one max pooling layer. Following, it classifies with two consecutive fully connected layers and a soft max classifier. This has proven to be effective in human activity recognition.

The second architecture, shown in figure 4, is based on the LeNet-5 network which is used for gaze-based activity recognition in [7]. Additionally, the LeNet-5 network is proven to be very accurate in predicting handwritten characters by extracting patterns from black and white images [18] similar to the preprocessed data shown in figure 2. This network extracts features by sequencing three blocks, where each block has a convolution and a pooling layer. Classification is done by one fully connected layer and a soft max classifier.

2.4 Hyperparameter optimization

The layers and preprocessing steps have hyperparameters that can potentially influence the performance of a CNN. Different datasets with different classes and sizes potentially have different needs. One requires more filter kernels the other a deeper network. Consequently, a hyperparameter search has to be performed for all three datasets. Classification can be done for new or known subjects, these different use cases might need different hyperparameters as well. All these factors (2 baseline architectures, 3 datasets, 2 use cases) result in $2 * 3 * 2 = 12$ sets of hyperparameters that need to be found. Searching for these parameters is done by trying all or a subset of parameter combinations and comparing performance. Since the size of the search space (set of all combinations) grows exponentially with each parameter it is often not feasible to try the complete search space. To overcome this problem smart search algorithms are used to approach the optimal solution in limited time.

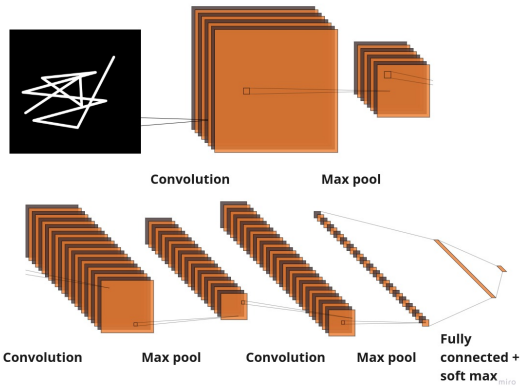


Figure 4: This 2D CNN requires black and white images as input, this input is generated during an extra preprocessing step depicted in figure 2. Feature extraction is done by alternating convolution and pooling layers three times. Classification is done by one fully connected layer and a soft-max classifier.

Hyperparameters

The searched hyperparameters are:

- **Framelength** is the size of the sliding window used during preprocessing. See section 2.2.
- **Image resolution** is the size of the image generated during preprocessing for the 2D CNN. See figure 2.
- **Line thickness** is the width of the line connecting the gaze points on the image during preprocessing step. See figure 2.
- **Filter count** is the number of filters applied in a convolution layers.
- **Kernel size** is the size of a convolution filter.
- **Dropout rate** is the ratio of randomly disabled neurons in a dropout layer.
- **Sizes of the fully connected layers**.
- **Layers** can be enabled or disabled to incorporate different architectures in the search space.
- **Gradient** can be enabled or disabled during preprocessing.

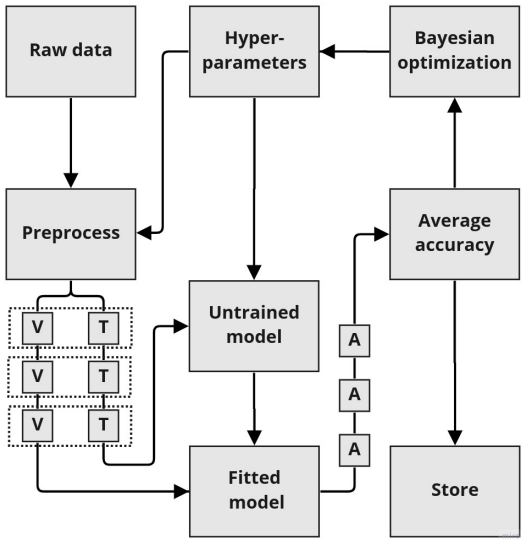


Figure 5: The search space exploration loop. Each trial contains three executions of different randomly picked train and validation data. The accuracies are averaged out.

Defining the search space

Often a hyperparameter is a number which can theoretically be infinitely large. For example, a convolution layer can have three or a million filters. However, three is too small, and a million requires too much memory. To make a search for hyperparameters feasible, a set of sensible values to choose from is required.

An initial search space is defined, by taking inspiration from existing networks such as [7], [18]. Next, a fast tuning algorithm is used to quickly explore the search space. This algorithm, called Hyperband optimization, is discussed in more detail in the *Optimization algorithms* section. If the best performing networks have hyperparameters which are on the edge of the search space a new search space is created. After a few iterations, a search space is determined that can be explored more thoroughly.

Exploring the search space

After determining the search space it is explored using bayesian optimization. Each iteration (trial), the bayesian optimization algorithm provides hyperparameters. These hyperparameters influence preprocessing and shape the neural network. The newly preprocessed data is used to train and validate the newly generated network. The resulting accuracy updates the bayesian optimization algorithm and is stored together with the hyperparameters for evaluation. See figure 5.

Since this research mainly uses small datasets the cut of train and validation data significantly influences the accuracy. To solve this issue, the cut is made three times randomly. Each cut is trained and validated and the accuracies are averaged. This way the chance of a (dis)fortunate cut is much smaller.

Optimization algorithms

The two proposed tuning algorithms are hyperband optimization [19] and bayesian optimization [20], [21].

The hyperband algorithm starts with a random search, thus randomly picking parameter combinations, and trains each network for a small number of epochs. The best-performing networks are selected to be further trained with more epochs. This selecting and training continues until one best network remains. [19]

Bayesian optimization treats a neural network as a black box function where the hyperparameter values are the input and the network accuracy is the output. It starts with a random search to sample the search space. When having reached a predetermined amount of samples the optimizer starts to search iteratively for the minimum. Each iteration it fits the black box function using a gaussian process regressor. An acquisition function chooses a point which has potentially the minimum value within its uncertainty. This point represents a hyperparameter combination and can be evaluated by training the network. The evaluated point is added to the sampled points and a new iteration starts. This algorithm runs for a predetermined number of trials, based on time and cost constraints [20], [21].

Finalizing hyperparameters

Even when averaging the results of three random splits, there exists a chance of three consecutive lucky picks. Alternatively, multiple different hyperparameter sets can have similar scores. Therefore, the ”best” scoring hyperparameters might not be the best after cross-validation or the best in terms of network size. Hence, the final hyperparameters are chosen heuristically from the top 20 best-performing networks. Taking value frequencies, averages and network size into account.

2.5 Validation

To determine the performance of a CNN the data is split into a training and testing set. Splitting data can be done before combining the subjects or after.

Data that is split before combining, results in a subject level cut, meaning that all data of each subject is either in the test or train set but not both. This cut measures performance in the classification of new subjects. A split after combining requires a random shuffle to represent all subjects equally in test and train.

The performance can vary significantly depending on the subject or subset used for testing, k-fold cross-validation is applied to overcome this problem. 10 folds are chosen for a frame cut and 8 for a subject cut. The 10 folds value is heuristically picked and aims to be a balance between computing time and performance. The 8 folds value cannot be larger since the smallest dataset only has 8 subjects. The performance metric used is the accuracy for classifying activities in the test set.

3 Experimental Setup and Results

3.1 Frameworks and tools

All processing and training has been done with python and the following libraries. Numpy, pandas and opencv are used

to handle and preprocess data. Tensorflow and keras are used to design, train, and validate CNNs. Keras-tuner is used for hyperband and bayesian optimization. Hyperparameters of the 2D network have been optimized using the DelftBlue supercomputer [22].

3.2 Implementation details

Model details

After all neural layers (Convolution and Dense) the *Rectified Linear Unit* activation function is used. The loss function of the models is *Categorical Crossentropy*. The *Adam optimizer* is used for gradient descent. Parameters of the Adam optimizer are left at the Keras default [23]. The input size of the pooling layers should be a multiple of the pooling size. However, that cannot be guaranteed during hyperparameter tuning and therefore padding is applied to the pooling layers.

Training details

The batch size for 1D networks is set to 64 when training on a local machine with an Nvidia Quadro P1000 GPU (4 GB VRAM). And for 2D networks a batch size of 32 or 16 is used, depending on the input resolution. When training the 1D networks on the DelftBlue (1x Nvidia Tesla V100S 32GB) a batch size of 256 is applied. 2D networks trained on the DelftBlue have a batch size of 256. These batches are split into 4 batches of 64 and trained distributedly on 4 Nvidia Tesla GPUs. Distributed training with data parallelism is achieved with the Keras mirrored strategy [24].

Custom tuning loop

Some hyperparameter combinations may generate an invalid network or require excessive amounts of memory. These issues can result in an error and will halt the Keras tuning loop. The Keras tuning framework cannot recover after an error and a restart is required which results in searching search space from the beginning and thus losing time. Due to time constraints, a custom tuner class is created to catch these errors and give negative feedback to the search algorithms by returning a loss of 100 for this trial.

Early stopping

A hyperparameter tuning algorithm runs as long as the time budget allows. By decreasing the duration of the trials, more hyperparameter combinations can be evaluated. A trial (training one network) takes in principle 30 epochs. However, the accuracy often reaches its optimum earlier. Continuing training for the remaining epochs is a waste of time and can often cause overfitting [25].

An early stopping callback is implemented to stop training whenever the loss value doesn't significantly decrease. The difference threshold is 1e-4 with a patience of 2 epochs. Patience is the number of epochs that is waited for the loss to further decrease.

3.3 Results

Tables 1 and 2 present the results for 1D CNNs and 2D CNNs respectively. The tables contain three elements: the determined search space by hyperband optimization, the found optimal hyperparameters for each dataset by subject cut and frame cut, and the evaluated accuracies during cross-validation.

The search space is defined as a set of values per hyper parameter. Each hyperparameter can take a value from *Min* to *Max* with *Step* being the distance between each value. *Exp* takes exponential steps. The hyperparameters in the *Subject split* columns are tuned while splitting data on subject. Similarly, in the *Frame split* columns are hyperparameters found with splitting data on frame level.

Each set of hyperparameters is validated on both subject and frame split even if it has been tuned using only one of the two. The bold accuracies are validated using the split it was tuned with.

A clear explanation of all hyperparameters can be found in section 2.4. A graphical representation of a 1D and 2D CNN can be found in figure 6a and 6b respectively.

3.4 Discussion

Accuracy analysis

From the accuracy section in both tables it is clear that splitting train and test on frame, results in significantly better scores. That could be expected considering the heterogeneity of visual behaviour of the test subjects. Meaning that the neural networks perform better when tested with known subjects. Strikingly, neural networks tuned with subject split perform better on frame split as well.

The 1D networks tuned and trained on the *Sedentary* dataset perform better on frame split but with a smaller margin. This might be due to the variability in signal length in the sedentary dataset.

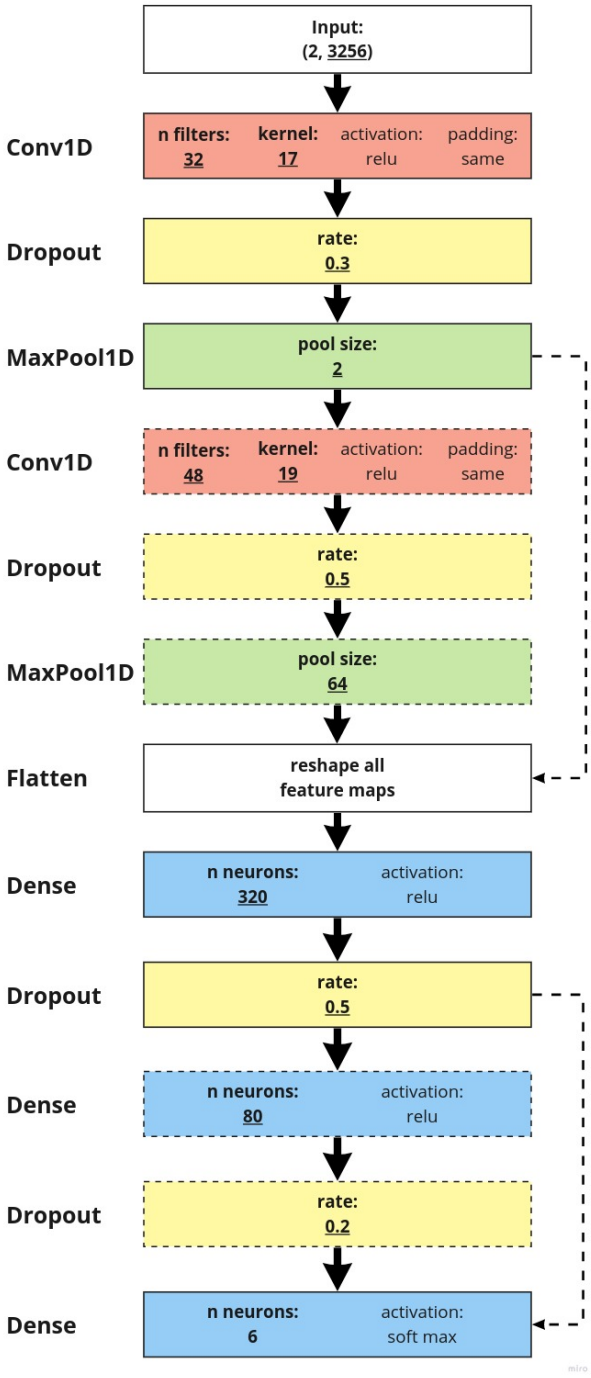
In general, while measuring performance using subject split, the better performing networks are networks tuned with subject split. However, the difference in performance compared to networks tuned with frame split is small. The same holds for networks tuned and validated with frame split. One exception is the 2D networks tuned on the *Reading* dataset. A significant performance gain can be found in both directions.

The 2D networks validated with frame splitting have exceptionally high performances. This is probably due to the sliding window. During this preprocessing step a small stride is used (30 samples) which is helpful in creating more data from small signals, a side effect is that data frames are quite similar, especially when using large window sizes (up to 4096). After shuffling and splitting, the test set contains windows with very similar frames. For the 2D networks, these frames are drawn as patterns on an image for which the LeNet5 network is proven to be very effective [18].

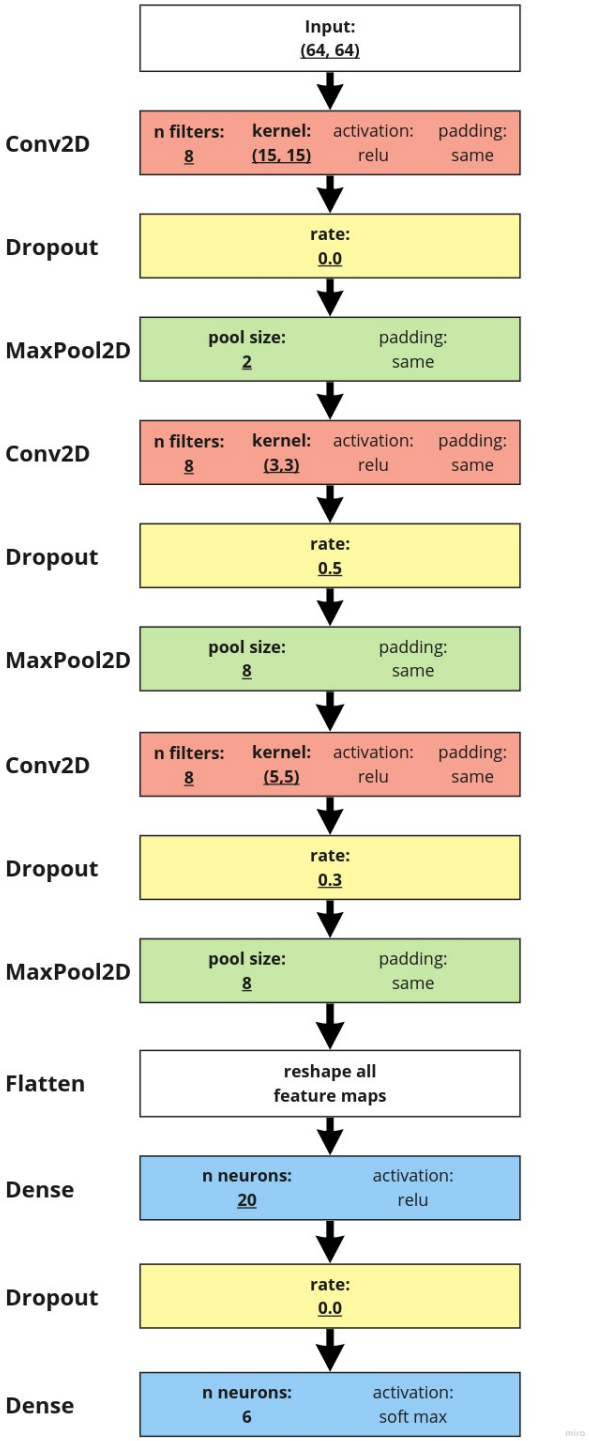
The most important takeaway is that there is no one size fits all.

Hyperparameter analysis

When comparing hyperparameters between datasets and splitting strategy, it is notable that there are a lot of variations and consequently again no size fits all. However there are some parameters the networks unanimously agree upon, for example all 1D networks perform better with a second convolution layer in place. Or that all 2D networks prefer a gradient in the drawn gaze.



(a) A 1-dimensional CNN found by hyperparameter tuning using the reading activity dataset. All underlined values are hyperparameters found during tuning. These and hyperparameters for other datasets are presented in table 1. The dashed layers are optional layers which can be enabled or disabled during tuning.



(b) A 2-dimensional CNN found by hyperparameter tuning using the reading activity dataset. All underlined values are hyperparameters found during tuning. These and hyperparameters for other datasets are presented in table 2.

Figure 6: 1D and 2D CNNs for the *Reading* dataset using subject split.

Table 1: This table presents the search space, found hyperparameters and accuracies for 1D CNNs. Each row represents one hyperparameter.

1D CNN	Search space			Tuned with subject split			Tuned with frame split		
Hyperparameter	**Min**	**Max**	**Step**	**Reading**	**Sedentary**	**Desktop**	**Reading**	**Sedentary**	**Desktop**
Frame length	256	4096	64	3256	512	4096	4096	256	4096
Conv 1: filter count	32	256	8	32	256	256	32	96	32
Conv 1: kernel size	3	25	2	17	3	3	25	25	25
Conv 1: drop out	0.0	0.5	0.1	0.3	0.3	0.0	0.3	0.0	0.0
Conv 1: max pool	2	16	Exp	2	2	16	16	16	16
Enable Conv 2	-	-	-	True	True	True	True	True	True
Conv 2: filter count	32	256	8	48	216	256	256	88	80
Conv 2: kernel size	3	25	2	19	25	3	3	3	3
Conv 2: drop out	0.0	0.5	0.1	0.5	0.5	0.5	0.0	0.3	0.0
Conv 2: max pool	2	64	Exp	64	8	64	64	2	64
Dense 1: size	60	340	20	320	340	220	340	340	340
Dense 1: drop out	0.0	0.5	0.1	0.5	0.3	0.3	0.0	0.1	0.0
Enable Dense 2	-	-	-	True	False	False	False	False	False
Dense 2: size	60	200	20	80	-	-	-	-	-
Dense 2: drop out	0.0	0.5	0.1	0.2	-	-	-	-	-
Subject score	-	-	-	**0.667**	**0.687**	**0.382**	0.646	0.623	0.394
Frame score	-	-	-	0.972	0.725	0.971	**0.998**	**0.658**	**0.989**

Table 2: This table presents the search space, found hyperparameters and accuracies for 2D CNNs. Each row represents one hyperparameter.

2D CNN	Search space			Tuned with subject split			Tuned with frame split		
Hyperparameter	**Min**	**Max**	**Step**	**Reading**	**Sedentary**	**Desktop**	**Reading**	**Sedentary**	**Desktop**
Frame length	512	4096	64	512	1024	4096	4096	1024	4096
Resolution	64	320	64	64	320	320	128	320	320
Line width	1	4	1	1	1	4	4	1	4
Gradient	-	-	-	True	True	True	True	True	True
Conv 1: filter count	8	128	8	8	8	8	8	8	8
Conv 1: kernel size	3	17	2	15	17	17	17	17	3
Conv 1: drop out	0.0	0.5	0.1	0.0	0.0	0.5	0.5	0.0	0.5
Conv 1: max pool	2	16	Exp	2	2	16	2	2	4
Conv 2: filter count	8	128	8	8	8	8	72	104	128
Conv 2: kernel size	3	17	2	3	3	7	17	17	17
Conv 2: drop out	0.0	0.5	0.1	0.5	0.5	0.2	0.5	0.0	0.4
Conv 2: max pool	2	16	Exp	8	2	2	16	2	16
Conv 3: filter count	8	128	8	8	80	88	96	8	8
Conv 3: kernel size	3	17	2	5	3	17	3	3	3
Conv 3: drop out	0.0	0.5	0.1	0.3	0.3	0.1	0.0	0.5	0.0
Conv 3: max pool	2	16	Exp	8	16	2	2	2	2
Dense: size	20	140	20	20	140	120	140	20	140
Dense: drop out	0.0	0.5	0.1	0.0	0.0	0.0	0.5	0.5	0.5
Subject score	-	-	-	**0.525**	**0.682**	**0.372**	0.328	0.432	0.369
Frame score	-	-	-	0.662	0.997	0.9995	**0.9990**	**0.9997**	**0.979**

Table 3: Best performing results compared to conventional machine learning techniques: random forest (RF), support vector machine (SVM), k nearest neighbours (k-NN). And compared to a different deep learning technique: long short-term memory (LSTM).

Algorithm	Reading	Sedentary	Desktop
RF	0.67	0.65	0.58
SVM	0.75	0.52	0.60
k-NN	0.71	0.48	0.54
LSTM	0.31	0.67	0.32
CNN	0.67	0.69	0.40

Table 4: Best performing results compared to conventional machine learning techniques: random forest (RF), support vector machine (SVM), k nearest neighbours (k-NN). And compared to a different deep learning technique: long short-term memory (LSTM).

Algorithm	Reading	Sedentary	Desktop
RF	0.96	0.94	0.92
SVM	0.85	0.86	0.95
k-NN	0.91	0.77	0.84
LSTM	0.98	0.98	0.95
CNN	1.00	1.00	0.99

Additionally, larger frame lengths are preferred by most. With an exception of the networks tuned on the *Sedentary* dataset. This has a practical reason, namely that the dataset contains signals of variable length where the shortest signal is only 1700 samples long.

Comparing performance

Performances for subject split and frame split are compared against other machine learning techniques in table 3 and 4 respectively. The scores of the other techniques are the results of other experiments within the same research group. Feature extraction for random forest, k nearest neighbours and support vector machine have been done manually per dataset. Long short-term memory has been developed separately, similar to this experiment.

For subject split, the support vector machine performs best for *Reading* and *Desktop* datasets. However, for *Sedentary* CNN performs best. For frame split, CNN outperforms all other techniques.

Again, no one size fits all.

4 Responsible Research

4.1 Biases

The datasets used in this research are small and often collected from a group of similar people in terms of ethnicity and age. This could potentially lead to heavy biased CNNs in terms of performance. For example, the Japanese document dataset consists of subjects reading Japanese documents. These documents are read vertically instead of horizontally which is the case for most western documents. Therefore the neural network trained on this dataset might not be able to correctly classify subjects reading western documents.

Even though the proposed neural networks should not be used for commercial purposes they are applicable in the context of this research: proving whether CNNs can be used for cognitive activity recognition.

To responsibly build neural networks for cognitive activity recognition, dataset size and diversity should be taken into consideration. This holds for training and for hyperparameter tuning since classifying a more diverse dataset might need a more complex network.

4.2 Reproducibility

Based on the implementation details and methodology it should be possible to reproduce the accuracies found for each set of hyperparameters. Due to the non-deterministic nature of neural networks, it is possible the scores slightly differ but not significantly.

For the hyperparameter search, the search spaces and pipelines are defined and should be reproducible as well. However, since bayesian optimization is a probabilistic process and every trial only has three executions, different hyperparameters may end up in the top-performing sets.

Final hyperparameter results can differ as well since they are heuristically picked. Nonetheless, the top 20 parameter sets used for heuristical picking all have similar performances. Thus picking slightly different hyperparameters in the same top 20 should not significantly impact performance.

Finally, some 2D neural networks have such a size that validation was only possible on the DelftBlue. Therefore, part of this research is only reproducible when having access to a high-performance computing cluster.

When reproducing this research, the same conclusion should be reached. Namely, answering if CNNs are suitable for gaze-based activity recognition.

5 Conclusions and Future Work

5.1 Conclusion

This research has proven that CNNs indeed are capable of classifying cognitive activities based on eye-movement data. For classifying known subjects it outperforms state-of-the-art conventional machine learning techniques and outperforms a different novel deep learning approach (LSTM). For classifying unknown subjects old machine learning techniques are still more powerful.

The best architectures for known subjects are CNNs based on 2-dimensional convolution, whereas 1-dimensional architectures are better for unknown subjects.

5.2 Future work

An interesting experiment to further explore the capabilities of CNNs for cognitive activity recognition is combining 1D and 2D convolution and merging the results either at the classification layer or at the flatten layer. This could potentially lead to a CNN which can both classify unknown and known subjects.

To improve performance for unknown subjects in general, it might be interesting to experiment with transfer learning by

first training a network on a large dataset and then calibrating it with a few data frames of a new subject.

Lastly, this research has focussed on relatively shallow deep learning. Adding more convolution and/ or dens layers might improve performance as well.

References

[1] J. K. Aggarwal and L. Xia, "Human activity recognition from 3d data: A review," *Pattern Recognition Letters*, vol. 48, pp. 70–80, 2014.

[2] P. Norloff, *Eye tracking technology is making new cars safer*, Sep. 2019. [Online]. Available: https://eyegaze.com/eye-tracking-technology-is-making-new-cars-safer/.

[3] M. Shoaib, S. Bosch, O. D. Incel, H. Scholten, and P. J. Havinga, "Complex human activity recognition using smartphone and wrist-worn motion sensors," *Sensors*, vol. 16, no. 4, p. 426, 2016.

[4] A. Bulling, J. A. Ward, H. Gellersen, and G. Tröster, "Eye movement analysis for activity recognition," in *Proceedings of the 11th international conference on Ubiquitous computing*, 2009, pp. 41–50.

[5] ——, "Eye movement analysis for activity recognition using electrooculography," *IEEE transactions on pattern analysis and machine intelligence*, vol. 33, no. 4, pp. 741–753, 2010.

[6] G. Lan, B. Heit, T. Scargill, and M. Gorlatova, "Gazegraph: Graph-based few-shot cognitive context sensing from human visual behavior," in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, 2020, pp. 422–435.

[7] Y. Yin, C. Juan, J. Chakraborty, and M. P. McGuire, "Classification of eye tracking data using a convolutional neural network," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2018, pp. 530–535. DOI: 10.1109/ICMLA.2018.00085.

[8] S. Münzner, P. Schmidt, A. Reiss, M. Hanselmann, R. Stiefelhagen, and R. Dürichen, "Cnn-based sensor fusion techniques for multimodal human activity recognition," in *Proceedings of the 2017 ACM International Symposium on Wearable Computers*, 2017, pp. 158–165.

[9] M. Zeng, L. T. Nguyen, B. Yu, *et al.*, "Convolutional neural networks for human activity recognition using mobile sensors," in *6th international conference on mobile computing, applications and services*, IEEE, 2014, pp. 197–205.

[10] S.-M. Lee, S. M. Yoon, and H. Cho, "Human activity recognition from accelerometer data using convolutional neural network," in *2017 ieee international conference on big data and smart computing (bigcomp)*, IEEE, 2017, pp. 131–134.

[11] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.

[12] M. Kassner, W. Patera, and A. Bulling, "Pupil: An open source platform for pervasive eye tracking and mobile gaze-based interaction," in *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*, ser. UbiComp '14 Adjunct, Seattle, Washington: Association for Computing Machinery, 2014, pp. 1151–1160, ISBN: 9781450330473. DOI: 10.1145/2638728.2641695. [Online]. Available: https://doi.org/10.1145/2638728.2641695.

[13] N. Srivastava, J. Newn, and E. Velloso, "Combining low and mid-level gaze features for desktop activity recognition," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 2, no. 4, pp. 1–27, 2018.

[14] *Tobii pro x2-30 screen-based eye tracker*, Jun. 2015. [Online]. Available: https://www.tobiipro.com/product-listing/tobii-pro-x2-30/.

[15] K. Kunze, Y. Utsumi, Y. Shiga, K. Kise, and A. Bulling, "I know what you are reading: Recognition of document types using mobile eye tracking," in *Proceedings of the 2013 international symposium on wearable computers*, 2013, pp. 113–116.

[16] *Smi eye tracking glasses*, Mar. 2022. [Online]. Available: https://imotions.com/hardware/smi-eye-tracking-glasses/.

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

[18] Y. LeCun *et al.*, "Lenet-5, convolutional neural networks," *URL: http://yann. lecun. com/exdb/lenet*, vol. 20, no. 5, p. 14, 2015.

[19] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6765–6816, 2017.

[20] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2015.

[21] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," *Advances in neural information processing systems*, vol. 25, 2012.

[22] D. H. P. C. C. (DHPC), *DelftBlue Supercomputer (Phase 1)*, https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase1, 2022.

[23] K. Team, *Keras documentation: Adam*. [Online]. Available: https://keras.io/api/optimizers/adam/.

[24] ——, *Keras documentation: Multi-gpu and distributed training*. [Online]. Available: https://keras.io/guides/distributed_training/.

[25] J. Brownlee, *Use early stopping to halt the training of neural networks at the right time*, Aug. 2020. [Online]. Available: https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping/.