SN Computer Science (2021) 2:59
https://doi.org/10.1007/s42979-020-00444-0

ORIGINAL RESEARCH

Classification of Eye Tracking Data in Visual Information Processing Tasks Using Convolutional Neural Networks and Feature Engineering

Yuehan Yin[1] · Yahya Alqahtani[2] · Jinjuan Heidi Feng[1] · Joyram Chakraborty[1] · Michael P. McGuire[1]

Received: 17 August 2020 / Accepted: 31 December 2020 / Published online: 30 January 2021

Abstract
Eye tracking technology has been adopted in numerous studies in the field of human–computer interaction (HCI) to understand visual and display-based information processing as well as the underlying cognitive processes employed by users when navigating a computer interface. Analyzing eye tracking data can also help identify interaction patterns with regard to salient regions of an information display. Deep learning technology is increasingly being used in the analysis of eye tracking data by allowing for the classification of large amounts of eye tracking results. In this paper, eye tracking data and convolutional neural networks (CNNs) were used to perform a classification task to predict three types of information presentation methods. As a first step, a number of data preprocessing and feature engineering approaches were applied to eye tracking data collected through a controlled visual information processing experiment. The resulting data were used as input for the comparison of four CNN models with different architectures. In this experiment, two CNN models were effective in classifying the information presentations with overall accuracy greater than 80%.

Keywords Eye tracking · HCI · CNN · Deep learning · Feature engineering

Introduction

Eye tracking technology has been increasingly adopted to study how users perceive and interact with computer interfaces. Eye tracking hardware measures the eye movements of a person when visually processing information [41]. Most commercial eye tracking hardware exploit the corneal reflection method to discover and trace the motion of the eye when it is shifting from one position to another [48]. In this method, a light source is used to illuminate the eye and detect the cornea using a high resolution camera [48]. Recently, machine learning has been used in the analysis of eye tracking data to successfully solve problems such as event detection [59] and automatic clustering of eye gaze data [37]. This paper is motivated by the need to predict how humans process computer information displays with the ultimate goal of creating more efficient interfaces that can adapt to gaze patterns [38].

Deep learning is a subfield of machine learning, and its approaches include deep supervised learning, deep unsupervised learning, and deep reinforcement learning [35]. Deep learning models consist of multiple processing layers that are used to learn representations of data with numerous levels of abstraction [31]. In other words, "deep" means that the models have a certain degree of credit assignment path (CAP) depth [36, 49]. Neural network models are the most widely adopted approaches in deep learning since error back-propagation can better solve the credit assignment problem. Among different deep learning neural networks, there are various model types such as convolutional neural networks (CNN), autoencoders (AE), recurrent neural networks (RNN), residual networks, and many others.

✉ Yuehan Yin
yyin1@students.towson.edu

Yahya Alqahtani
yalqahtani@jazanu.edu.sa

Jinjuan Heidi Feng
jfeng@towson.edu

Joyram Chakraborty
jchakraborty@towson.edu

Michael P. McGuire
mmcguire@towson.edu

1 Department of Computer and Information Sciences, Towson University, Towson, MD, USA

2 College of Computer Science and Information Technology, Jazan University, Jazan, Saudi Arabia

Convolutional neural networks have been widely applied to solve various computer vision problems associated with processing images and video. The most famous early CNN model was the LeNet-5 model [32, 33] which was designed to identify hand-written digits. With advances in GPU technology and the rapid development of deep learning platforms, many CNN models have been proposed including AlexNet [30], VGGNet [51], GoogLeNet [53] and YOLO [43]. The successes and widespread application of CNN models in the prediction of features in traditional image datasets has given way to the application of CNN models in other domains where data are converted to an image or tensor format.

There is no standard documented process with which one can transform eye tracking data to be used in deep learning models. Thus, a secondary motivation of this paper is to determine how to best model eye tracking data as a set of images that can be used as input for a CNN model. Feature engineering is the procedure of utilizing knowledge of the data along with transformation methods to improve the performance of a machine-learning algorithm [9]. In the context of CNN models, feature engineering can be used to enhance features in the images that are used as input to the model. Whether there is a need to employ feature engineering to improve CNN model performance for eye tracking data is currently unknown. With this in mind, we employed several feature engineering approaches and evaluated their impact on model performance.

In this study, four CNN models were tested with seven data preprocessing and feature engineering approaches to predict three different information presentation methods using eye tracking data. Overall, the results prove the effectiveness of CNN models to predict information presentation methods with accuracy greater than 80%. The paper is organized as follows: related work to this study is introduced in the next section. The overview of the approach and dataset are presented in the subsequent section. Then data preprocessing and feature engineering are described, followed by which CNN models are depicted. The details of the experiments and results with comparison are presented before the final section. Finally, the discussion and conclusion of this study are presented.

Related Work

Researchers have proposed many approaches for designing adaptable and adaptive interfaces [16, 38]. Gullà defines adaptable user interfaces (AdUIs) as systems where the activation and selection of user–computer interaction can be carried out by the end user via the choice of a tailor-made user profile from a predefined list. Adaptive systems can be defined as systems that alter facets of their structure over time in response to changing user interface requirements [5]. The research of adaptive user interfaces (AUIs) is an important direction in the field of HCI. The goal of AUI research is to furnish people with highly usable systems accommodating different needs and characteristics in different environments [16]. Eye tracking has become a powerful tool in usability studies to assess the usability of user interfaces. Problems caused by poor interface design can be identified through the particular pattern of eye movements [15]. Furthermore, application of eye tracking can assist people with disabilities. In one previous study, the researchers designed an intelligent tool called Vision Assistant to assist people with different disabilities [18]. The goal of this research was to use an adaptive eye-tracker system where a universal and easy-to-use human–computer interface can be set up to replace the mouse and keyboard. The technique they used could extract the shapes of the human eye and analyze them in real-time to obtain the position of the eye in an incoming image. The position was then interpreted to control the position of a mouse cursor on the user's screen. A study related to the automotive domain has suggested that adaptive interfaces in partially automated vehicles offer assistance to give drivers a method to monitor the automated system [45]. In another study, researchers used eye tracking to classify information usage changes over time according to driver experience in partially automated vehicles in order to propose design recommendations for future adaptive interfaces [55].

Eye tracking has been applied to studies in different fields such as biometrics, psychology, medicine, and human–computer interaction. In recent years, it has also been combined with deep supervised learning to solve various problems. The following studies applied deep supervised learning to different eye tracking data to perform a number of different classification tasks. Neural networks, including CNNs were successfully used in all the studies. They also played different roles such as a classifier or a feature extractor. Furthermore, in a number of studies, feature engineering was applied to raw eye tracking data before training a neural network model.

One study focused on the use of deep gaze velocity for biometric identification of radiologist expertise during the reading of mammographic imagery [58]. Individual radiologists were grouped by three levels of expertise including new radiology residents, advanced radiology residents, and expert radiologists. A CNN classifier was applied to eye tracking data collected from 10 radiologists who read 100 mammograms of various diagnoses. The results of the CNN was compared with a deep neural network [21], a deep belief network [22], and a hidden Markov model classifier. In the study, tenfold cross-validation was used to train each classification algorithm. The macro F1-score was used to evaluate and compare the classification methods. The results show that the CNN classifier had the best performance in the classification task and thus proved to be an effective approach of analyzing one-dimensional eye tracking data.

Another line of research focused on gaze-based detection of mind wandering [24]. In this study, investigators made an automated detector of mind wandering for students by using eye-gaze and contextual information to assist in interaction with an intelligent tutoring system for biology. A number of different classification models were compared including Bayesian networks, logistic regression, multilayer perceptrons (MLP), random forests, support vector machines (SVM), and NeuroEvolution of Augmenting Topologies (NEAT) to classify two types of responses of the detector. Feature engineering was applied to eye-gaze data along with contextual information to create a composite set of features. Fixations and saccades from the raw eye-gaze data were calculated by using an open source package for eye tracking analysis. Gaze features were calculated from six general measures such as fixation duration and saccade length regarding fixations and saccades. According to sizes of time window, ranging from 3 to 30 s, gaze features were generated from fixations and saccades along with eight contextual features. The F1 measure was used to evaluate the classification models and the NEAT classifier produced the best results.

In [7], researchers proposed to use CNNs as feature extractors combined with a SVM classifier to automatically identify the vision disease strabismus. Six different pre-trained CNN models were used in the study including AlexNet [30], VGG-F, VGG-M, VGG-S [6], VGG-16, and VGG-19 [51]. These models were trained on ImageNet, which is a large and well-known image database. The results of the study show that the proposed approach is promising to recognize strabismus and demonstrate that eye tracking data can be represented by the well transferred natural image features.

In [57], a modified LeNet5 CNN model combined with feature engineering model was used to determine whether a user was interacting a particular interface (Google News or NewsMap) to answer questions about current events. The resulting grayscale images were fed to train the CNN model and perform two classification tasks to identify web user interfaces and nationalities of users. The evaluation metrics used for assessing the model were accuracy (or recognition rate), recall (or sensitivity), specificity, precision and negative predictive value. Finally, the effect of a number of feature engineering and their impact on classification was analyzed. This study suggested that deep learning combined with feature engineering is promising in the classification of eye tracking results.

In addition, deep learning has been used for gaze tracking. In [39], researchers developed an eye detection and pupil tracking approach which can be used to capture eye movements of users. This study implemented eye-tracking using an inexpensive camera and applied deep learning for object recognition. Their prototype model achieved 85.5% accuracy which was comparable to the real-time system based on eye-tracking. CNNs have also been used to extract facial features for facial expression [28, 56]. These studies proposed their own CNNs including an appearance feature-based CNN structure and deep comprehensive multipatch aggregation convolutional neural networks. In [42], deep neural networks were used in combination with a low-cost eye-tracking system. Due to the speed and high accuracy required for gaze tracking tasks, only few suitable neural networks, such as YOLOv3 [44], SSD [34], Mask R-CNN [19], were considered. Among these three neural networks, the researchers selected YOLOv3 to detect objects in rectangular boxes and to obtain gaze position coordinates. Another related study detects eye landmarks using a novel weakly supervised learning approach with two modules including object detection and recurrent learning [23]. The first module associated with faster R-CNN is capable of detecting original eye positions and a bounding-box of facial components. The second module uses the initial shape of the eye to refine eye landmarks.

Based on the existing literature, different machine learning models and CNN models have been used to solve a number of eye tracking problems. In addition, feature engineering has been used to prepare features in order to make models perform better. In this study, four CNN models with different architectures were tested to solve a classification problem of recognizing information presentation methods. A process used to transform eye tracking data to be used in deep learning models is presented where different feature engineering methods were applied to convert eye tracking data used as input for CNN models. This study also shows effects of different CNN model architectures combined with different feature engineering methods in the classification of eye tracking data.

Overview of Approach and Dataset

Preliminaries and Problem Statement

Given a sequence of gaze points P, where P is an ordered set such that $P = \{p_1, p_2, \dots, p_n\}$. Each $p_i \in P$ is composed of a tuple (t_i, x_i, y_i), where t_i is a timestamp in sequential order such that $t_1 < t_2 < \dots < t_n$, and x and y are coordinates in Euclidean space that indicate the spatial position of the gaze point on the computer screen.

Problem statement and objectives Given a sequence of gaze points related to human eye movements, the first objective of the approach is to transform raw gaze point data into a set of subsequences that can then be converted into a set of images to be consumed by CNN models. Given this set of images, a secondary objective is to test the effectiveness of a number of feature engineering approaches to analyze how they affect classification accuracy. The result of which is a transformed set of images with enhanced eye tracking features. The final objective is to experimentally train and evaluate four CNN classifiers on the resulting sequence of images to classify three different types of information presentation interfaces.

Overview of Approach

Figure 1 shows an overview of the approach used in this study. In the first step of the approach, raw eye tracking data were preprocessed to retrieve gaze points based on active intervals of a number of areas of interest (AOIs) in each interface. In this step, several key attributes were selected as a subset from the raw data (Table 1). In the second and third steps, the resulting data were converted to images based on a time window. In addition, a number of feature engineering methods were applied to the images to generate scan paths. The resulting images were then used to train four convolutional neural networks. Finally, 28 experiments were conducted to compare the performance of the different feature engineering approaches and classification models. The details of these steps are explained and described in the following sections.

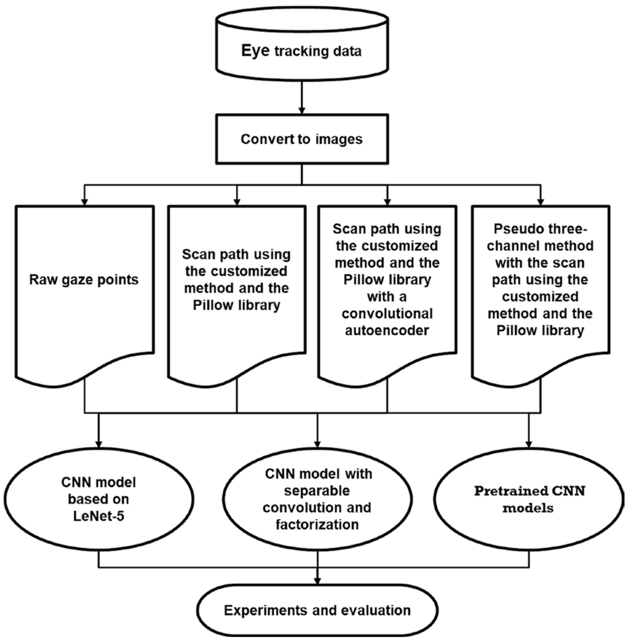

Eye tracking data
Convert to images
Raw gaze points
Scan path using the customized method and the Pillow library
Scan path using the customized method and the Pillow library with a convolutional autoencoder
Pseudo three-channel method with the scan path using the customized method and the Pillow library
CNN model based on LeNet-5
CNN model with separable convolution and factorization
Pretrained CNN models
Experiments and evaluation

Fig. 1 Overview of approach

Dataset Summary

The eye tracking data in this study were collected from two studies. The first study was designed to evaluate differences in visual information processing behavior between users from the United States and users from Saudi Arabia [2]. The second study compared information processing behaviors of users with Attention Deficit Hyperactivity Disorder (ADHD) and neurotypical users [3]. In both studies, the participants completed the same visual information processing tasks following the same procedure with the same hardware and system settings. Eye tracking data were collected as the users viewed information presented on a customized web page to answer questions. The questions were based on fictitious health-related information about smoking, high blood pressure, high cholesterol, and diabetes of four ethnic groups from four countries. The information was presented in three different formats: textual presentation, graphical presentation, and tabular presentation. To answer the questions, the users had to review the information presented on a web page to compare and contrast the data. The three interfaces used for each information display method are shown in Fig. 2. Twelve participants from the U.S. and twelve participants from Saudi Arabia took part in the first study. None of the participants had any documented disabilities. The age range of American users was between 20 and 30 years and the age range of Saudi users was between 20 and 37 years. In the second study, we examined the impact of ADHD from 12 participants from the U.S. who took part in the study. The age range of the participants was between 18 and 24. The data of the participants with ADHD were compared with the data collected of the participants from the U.S. in the first study to examine the difference between neurotypical users and users with ADHD. Eye tracking data were collected using a Tobii X2-60 eye tracker. The raw gaze point data were extracted using the Tobii Studio software. The attributes that were extracted are Recording Date, Participant Name, Local Timestamp, GazePointX (ADCSpx) and GazePointY (ADCSpx). The details for each attribute are listed in Table 1.

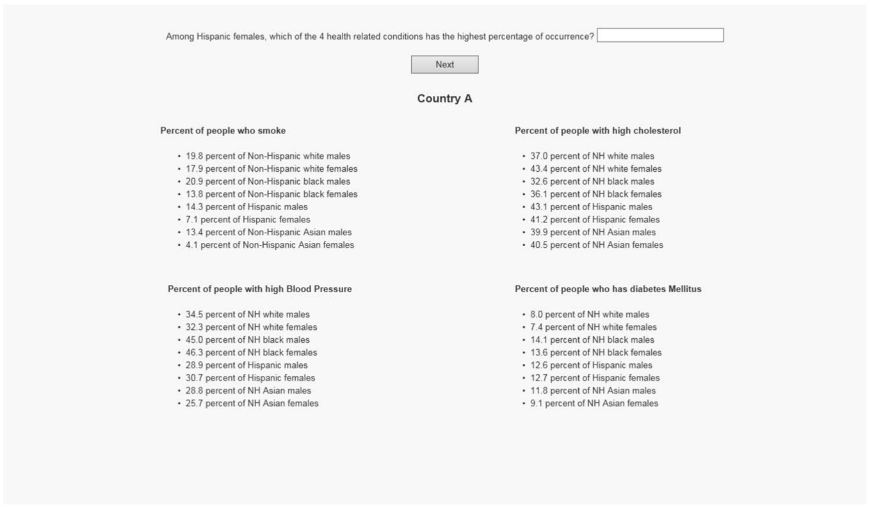
Data Preprocessing and Feature Engineering

In the following section, the data preprocessing step and feature engineering approaches are presented. The overall work flow path of this study is shown in Fig. 3.

Data Preprocessing

Eye tracking data were exported to two files in comma separated values (csv) format and the key attributes shown in Table 1 were extracted. For the classification task in this

Fig. 2 Three information presentation methods used in the study



Among Hispanic females, which of the 4 health related conditions has the highest percentage of occurrence?

Country A

Percent of people who smoke

- 19.8 percent of Non-Hispanic white males
- 17.9 percent of Non-Hispanic white females
- 20.9 percent of Non-Hispanic black males
- 13.8 percent of Non-Hispanic black females
- 14.3 percent of Hispanic males
- 7.1 percent of Hispanic females
- 13.4 percent of Non-Hispanic Asian males
- 4.1 percent of Non-Hispanic Asian females

Percent of people with high cholesterol

- 37.0 percent of NH white males
- 43.4 percent of NH white females
- 32.6 percent of NH black males
- 36.1 percent of NH black females
- 43.1 percent of Hispanic males
- 41.2 percent of Hispanic females
- 39.9 percent of NH Asian males
- 40.5 percent of NH Asian females

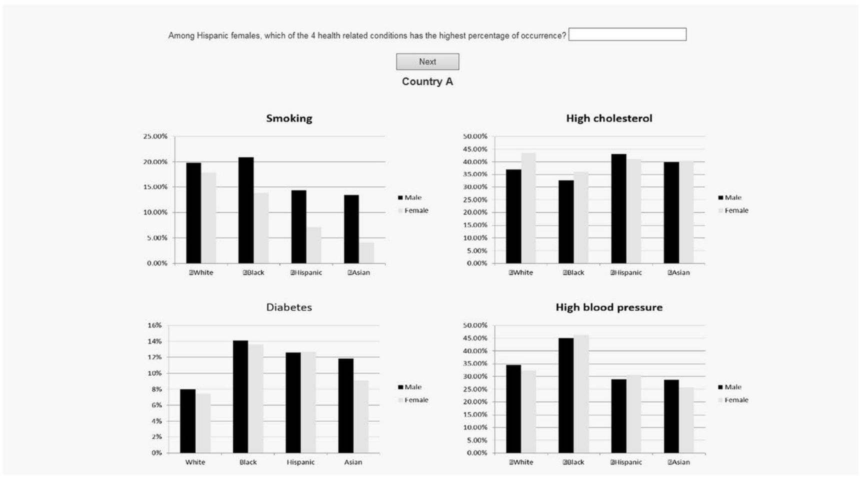
Percent of people with high Blood Pressure

- 34.5 percent of NH white males
- 32.3 percent of NH white females
- 45.0 percent of NH black males
- 46.3 percent of NH black females
- 28.9 percent of Hispanic males
- 30.7 percent of Hispanic females
- 28.8 percent of NH Asian males
- 25.7 percent of NH Asian females

Percent of people who has diabetes Mellitus

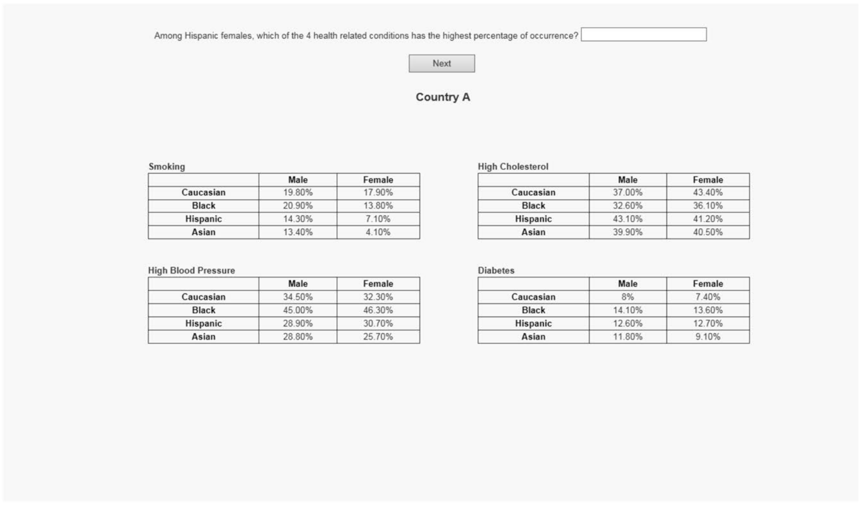
- 8.0 percent of NH white males
- 7.4 percent of NH white females
- 14.1 percent of NH black males
- 13.6 percent of NH black females
- 12.6 percent of Hispanic males
- 12.7 percent of Hispanic females
- 11.8 percent of NH Asian males
- 9.1 percent of NH Asian females

Textual presentation.



Among Hispanic females, which of the 4 health related conditions has the highest percentage of occurrence?

Graphical presentation.



Among Hispanic females, which of the 4 health related conditions has the highest percentage of occurrence?

Country A

Smoking

	Male	Female
Caucasian	19.80%	17.90%
Black	20.90%	13.80%
Hispanic	14.30%	7.10%
Asian	13.40%	4.10%

High Cholesterol

	Male	Female
Caucasian	37.00%	43.40%
Black	32.60%	36.10%
Hispanic	43.10%	41.20%
Asian	39.90%	40.50%

High Blood Pressure

	Male	Female
Caucasian	34.50%	32.30%
Black	45.00%	46.30%
Hispanic	28.90%	30.70%
Asian	28.80%	25.70%

Diabetes

	Male	Female
Caucasian	8%	7.40%
Black	14.10%	13.60%
Hispanic	12.60%	12.70%
Asian	11.80%	9.10%

Tabular presentation.

Table 1 Dataset attributes

Attribute	Description
Recording Date	Date of recording Format: mm/dd/yyyy
Participant Name	Name of the participant associated with the Tobii Studio recording
Local Timestamp	Recording computer local date time value timestamp. This provides the timestamp from the internal clock of the computer running Tobii Studio in the data format Hour: Minutes: Seconds, Milliseconds
GazePointX (ADCSpx)	Horizontal coordinate of the averaged left and right eye gaze point on the screen with pixel units. The origin is at the upper left corner of the active display area
GazePointY (ADCSpx)	Vertical coordinate of the averaged left and right eye gaze point on the screen with pixel units. The origin is at the upper left corner of the active display area

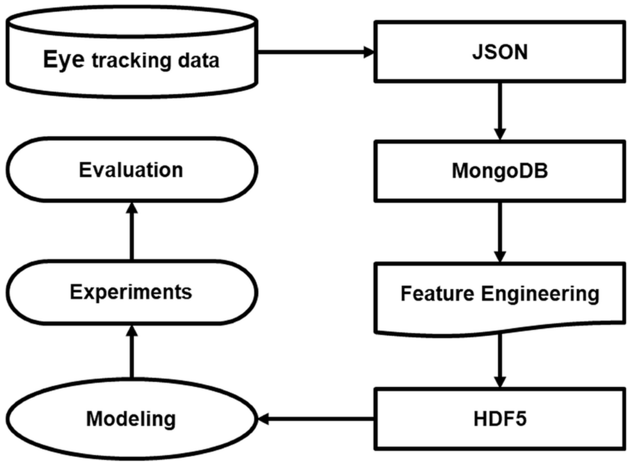


Fig. 3 Overall work flow path

study, eye movement patterns were observed in short time intervals when areas of interest (AOIs) of the user interface (UI) were active. AOIs were delineated for each area on the interface that held data. For example, given a table of information, an AOI would be drawn around the table to track gaze points within the AOI. The activation state of an AOI means that a particular table, graphic, or text box was being used to answer a question at that time. The AOI would then change from active to inactive when the information presentation method switched (e.g. from table to graphic). The beginning and end timestamps corresponding to active intervals of AOIs of three different information representation methods were recorded manually. The exported csv files contained all data including when an AOI was active or inactive. Because of this, some *x* or *y* coordinates of gaze points were missing. Thus, gaze point data corresponding to inactive intervals of AOIs and coordinates of gaze points which were not complete were removed. The rest of the data with the key attributes mentioned in Table 1 were kept.

The eye tracking data were sliced using a 10-s time window where every 10 seconds of gaze points were grouped for the three different information presentation methods. We chose a time window of 10 s to keep this study consistent with our previous study [57]. Each 10-s gaze point image was represented by a 2D array with a size of 1440×900 —which corresponds directly to the screen resolution of the computer used for the study. Two types of images were created including grayscale images and pseudo-three-channel images. Since a time window size of 10 s was used, each image contained 10 s of eye movements. Cells in each array were assigned a value of 1 to indicate that that cell had an active gaze point during the 10 second period. If a gaze point reoccurred in a cell, the value in that place would be increased by 1. This results in a 2D array where the cells represent pixels and the values represent gaze points for 10 s of eye movements.

Database and Data Formats

As was the case in our previous study [57], the JavaScript Object Notation (JSON) format was used to transform the eye tracking data because it can be easily constructed using pairs of name/value and an ordered list of values [52]. MongoDB was chosen to store the data because it directly maps to the JSON format. Another benefit of this model is that the data model and persistence strategies of MongoDB are built for high read-and-write throughput and the ability to scale easily with automatic failover [4].

The transformed data were stored in a single JSON document for each 10s time window. Each JSON document contains eight fields as shown in Fig. 4. The data_id field serves as a unique identifier for each JSON document. The gp_image field contains a raw gaze points image which is a 2D array filled with gaze points for 10 s. The gp_sequence field is a customized data structure that consists of a timestamp sequence for 10 s of gaze point coordinates recorded during the 10s interval (GazePointX and GazePointY). The timestamp_begin field and the timestamp_end field indicate the beginning and end timestamps for each 10s interval. The type_of_question field is the class label to indicate what information representation method corresponds to a particular set of gaze points. The values of this field are text, graph, or table. The last field is user_class which identifies

```
{
  'data_id': 1,
  'gp_image': array([[0, 0, 0, ..., 0, 0, 0],
                     [0, 0, 0, ..., 0, 0, 0],
                     [0, 0, 0, ..., 0, 0, 0],
                      ...,
                     [0, 0, 0, ..., 0, 0, 0],
                     [0, 0, 0, ..., 0, 0, 0],
                     [0, 0, 0, ..., 0, 0, 0]]),
  'gp_sequence': [['13:05:59.114', [1286, 99]],
                  ['13:06:00.316', [743, 141]],
                  ['13:06:00.333', [747, 154]],
                  ...,
                  ['13:06:09.067', [468, 388]],
                  ['13:06:09.084', [458, 392]],
                  ['13:06:09.100', [449, 368]]],
  'participant_name': 'U1',
  'timestamp_begin': '2018-03-12 13:05:59.114000-04:00',
  'timestamp_end': '2018-03-12 13:06:09.100000-04:00',
  'type_of_presentation': 'Text',
  'user_class': 'U'
}
```

Fig. 4 A sample JSON representation of a 10-s time window

the participant groups in the study U for American, S for Saudi Arabian, and AU for ADHD.

Using the above database structure, a sequence of gaze point coordinates with corresponding timestamps were retrieved from MongoDB and converted to grayscale images. Then a number of feature engineering methods were applied to the grayscale image. Additionally, pseudo-three-channel images were generated using a convolutional autoencoder to be tested as a potential feature engineering approach. In addition, HDF5 [12] was used to hold big numerical arrays that resulted from the feature engineering approaches. Finally, images were organized based on a batch size parameter. The resulting batches of images were used as input to the CNN models for training. The program was broken down to first create the different types of images which would be used for training, validating and testing the CNN models. All the images were represented using arrays and were stored in seven datasets separately in a single HDF5 file.

Feature Engineering

Feature engineering methods are sometimes applied to input data to enhance image features and allow machine learning models to more easily recognize features. As was mentioned in "Data preprocessing", both grayscale and pseudo-three channel images were used to train the CNN models. Figure 5 depicts the nine types of images. Six of the nine resulted from the feature engineering approaches. The title of each image indicates the feature engineering approach that was used. Each image represents 10 s of eye tracking data.

In the first row of Fig. 5, the sample images were generated using the raw gaze data without applying any feature engineering methods on them. In addition, min–max scaling was employed on the image arrays generated using processed raw gaze data to make the values scaled consistently and to help CNN models to make a prediction.

Scan Path Creation

In the second row of Fig. 5, the images were produced by using the gp_sequence field in the JSON representation stored in the database to connect the gaze points together to create an image of scan paths. This method is shown in Algorithm 1. The input to the algorithm is an ordered list of coordinates of x and y such that $[[x_1, y_1], [x_2, y_2], \ldots [x_n, y_n]]$, which is the *coordList* coming from the gp_sequence field of a JSON representation obtained from an ordered set of gaze points $P_{t\_w\_s=10s} \subset P$ where $t\_w\_s = 10s$ denotes that the time window size is 10 s. The output is a 1440-by-900 image represented by using a 2D array with 900 rows and 1440 columns. This algorithm does not connect the positions of two gaze points directly when they were not on the same main diagonal or anti-diagonal of the 2D array. Instead, they are linked indirectly such that a starting gaze point would stretch out on its main diagonal or anti-diagonal first based on the position of the gaze point with which it would connect. Then, the path continues until it reaches the row or column in the 2D array for the gaze point with which the starting gaze point would link. Finally, the path continues until it reaches the end gaze point. According to the same principle, the rest of the gaze points, in a 10-s interval, are then linked together. Figure 6 depicts a simple example of this problem based on a 4-by-6 2D array to demonstrate how the algorithm connects two gaze points together. It consists of 9 example cases. The first case shows that the starting gaze point and ending gaze point are in the same position of the 2D array. The second and third examples display that the starting gaze point and ending gaze point are in the same row or column of the 2D array. The fourth and fifth examples demonstrate that the starting gaze point and ending gaze point are on the same main diagonal or anti-diagonal in the 2D array. The sixth, seventh, eighth and ninth examples depict that the starting gaze point and ending gaze point are not on the same main diagonal or anti-diagonal in the 2D array, and the starting gaze point is at the top of the array and the ending gaze point is at the bottom of the array. When connecting two gaze points, the algorithm will assign 1 to the positions in the array. The remaining area of the array will be 0.

In the last row of Fig. 5, the images were also generated

Algorithm 1 Scan path creation using gp_sequence field

```
1: Input: $coordList$
2: Output: $\mathcal{M} \in \mathbb{R}^{900\times 1440}$
3: $\mathcal{M} \leftarrow zeros((900, 1440))$    ▷ Create a 900-by-1440 2D array filled with 0s
4: Swap $x$ and $y$ coordinates in the $coordList$
5: if $(len(coordList) > 1)$ then
6:     for ($r$ from 0 to $len(coordList) - 1$) do
7:         if $(r + 1 \neq len(coordList))$ then
8:             $currentY \leftarrow coordList[r][0]$
9:             $currentX \leftarrow coordList[r][1]$
10:            $nextY \leftarrow coordList[r + 1][0]$
11:            $nextX \leftarrow coordList[r + 1][1]$
12:            if $(currentY = nextY$ and $currentX = nextX)$ then
13:                $\mathcal{M}[currentY, currentX] \leftarrow 1$
14:            else if $(currentY = nextY)$ then
15:                if $(currentX < nextX)$ then
16:                    Fill 1s at the row $currentY$ from column $currentX$ to column $nextX$ in $\mathcal{M}$
17:                else if $(currentX > nextX)$ then
18:                    Fill 1s at the row $currentY$ from column $nextX$ to column $currentX$ in $\mathcal{M}$
19:                end if
20:            else if $(currentX = nextX)$ then
21:                if $(currentY < nextY)$ then
22:                    Fill 1s at the column $currentX$ from row $currentY$ to row $nextY$ in $\mathcal{M}$
23:                else if $(currentY > nextY)$ then
24:                    Fill 1s at the column $currentX$ from row $nextY$ to row $currentY$ in $\mathcal{M}$
25:                end if
26:            else if $((currentY < nextY$ and $currentX < nextX)$ or $(currentY > nextY$ and $currentX > nextX))$ then
27:                $\mathcal{M}_{sub} \leftarrow$ select a sub-matrix in $\mathcal{M}$ using coordinates of $currentY$ and $currentX$, and of $nextY$ and $nextX$ as endpoints    ▷ The $\mathcal{M}_{sub}$ has access to $\mathcal{M}$.
28:                $indicesList \leftarrow$ main_diagonal $(\mathcal{M}_{sub})$    ▷ Obtain the indices related to entries to access the main diagonal of $\mathcal{M}_{sub}$
29:                Fill 1s in $\mathcal{M}_{sub}$ using the indices in the $indicesList$
30:                if (the bottom right endpoint of the main diagonal of $\mathcal{M}_{sub}$ is right above the bottom right corner of $\mathcal{M}_{sub}$) then
31:                    Fill 1s from the endpoint to the corner in $\mathcal{M}_{sub}$
32:                else if (the bottom right endpoint of the main diagonal of $\mathcal{M}_{sub}$ is on the left side of the bottom right corner of $\mathcal{M}_{sub}$) then
33:                    Fill 1s from the endpoint to the corner in the $\mathcal{M}_{sub}$
34:                end if
35:            else if $((currentY < nextY$ and $currentX > nextX)$ or $(currentY > nextY$ and $currentX < nextX))$ then
36:                $\mathcal{M}_{sub} \leftarrow$ select a sub-matrix in $\mathcal{M}$ using coordinates of $currentY$ and $currentX$, and of $nextY$ and $nextX$ as endpoints    ▷ The $\mathcal{M}_{sub}$ has access to $\mathcal{M}$.
37:                $indicesList \leftarrow$ antidiagonal $(\mathcal{M}_{sub})$    ▷ Obtain the indices related to entries to access the antidiagonal of $\mathcal{M}_{sub}$
38:                Fill 1s in $\mathcal{M}_{sub}$ using the indices in the $indicesList$
39:                if (the bottom left endpoint of the antidiagonal of $\mathcal{M}_{sub}$ is right above the bottom left corner of $\mathcal{M}_{sub}$) then
40:                    Fill 1s from the endpoint to the corner in the $\mathcal{M}_{sub}$
41:                else if (the bottom left endpoint of the antidiagonal of $\mathcal{M}_{sub}$ is on the right side of the bottom right corner of $\mathcal{M}_{sub}$) then
42:                    Fill 1s from the endpoint to the corner in $\mathcal{M}_{sub}$
43:                end if
44:            end if
45:        end if
46:    end for
47: end if
48: Return $\mathcal{M}$
```

using a sequence of 10-s timestamps and the corresponding pairs of gaze point coordinates from the gp_sequence field. This feature engineering approach uses the Pillow Python library [40] to yield the scan paths. The input to Algorithm 2 is an ordered list of coordinates of x and y such that $[[x_1, y_1], [x_2, y_2], \dots [x_n, y_n]]$, which is the *coordList* coming from the gp_sequence field of a JSON representation obtained from an ordered set of gaze points $P_{t\_w\_s=10s} \subset P$ where $t\_w\_s = 10s$ denotes a time window size of 10 s. The output is a 900-by-1440 image represented using a 2D array. The goal of the method was the same as the approach presented in Algorithm 1. This approach improves the problem associated with linking two gaze points which were not on the same main diagonal or anti-diagonal. In this method, the line function from the ImageDraw module of the Pillow Library was used to connect different positions of gaze points based on the timestamp. The function attempts to

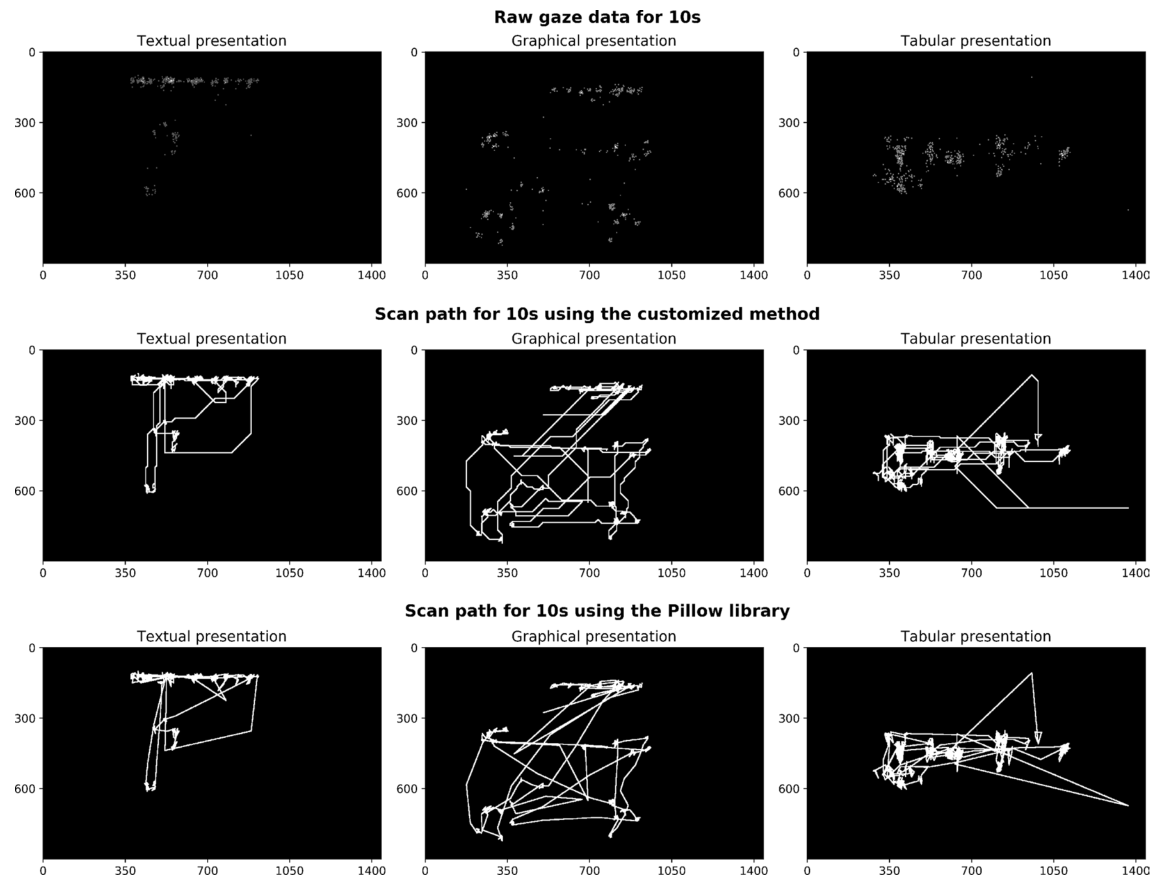


Fig. 5 Three different types of grayscale sample images generated from three sample JSON representations within a 10-s time window for each based on three different information presentation methods: textual presentation, graphical presentation, and tabular presentation. The images in the first row were generated using raw gaze points. The images in the second and third row were derived using Algorithm 1 and Algorithm 2, respectively

directly link two gaze points that are not on the same main diagonal or anti-diagonal. The decimal code for black and white colors were applied in the line function. Therefore, min-max scaling was exploited to process the images in order to scale the values in the 2D array to a fixed range. The images of this row illustrate the scan paths of three different information presentation methods produced using this approach. In comparison with the last rows, the difference between directly linking two gaze points that are not on the same main diagonal or anti-diagonal and indirectly linking them can be easily discovered. The following algorithm displays steps of the method in details.

Algorithm 2 Scan path using the Pillow library

```
1: Input: $coordList$
2: Output: $\mathcal{M} \in \mathbb{R}^{900\times 1440}$
3: $\mathcal{M} \leftarrow$ Initialize a black image with a size of width and height (1440, 900) using 8-bit pixels
   indicating a black and white mode
4: $\mathcal{M} \leftarrow$ Call the line function with an ordered list of coordinates of $x$ and $y$ in $coordList$, white
   as the filled color, and "curve" as the joint type between a sequence of lines
5: Return $\mathcal{M}$
```

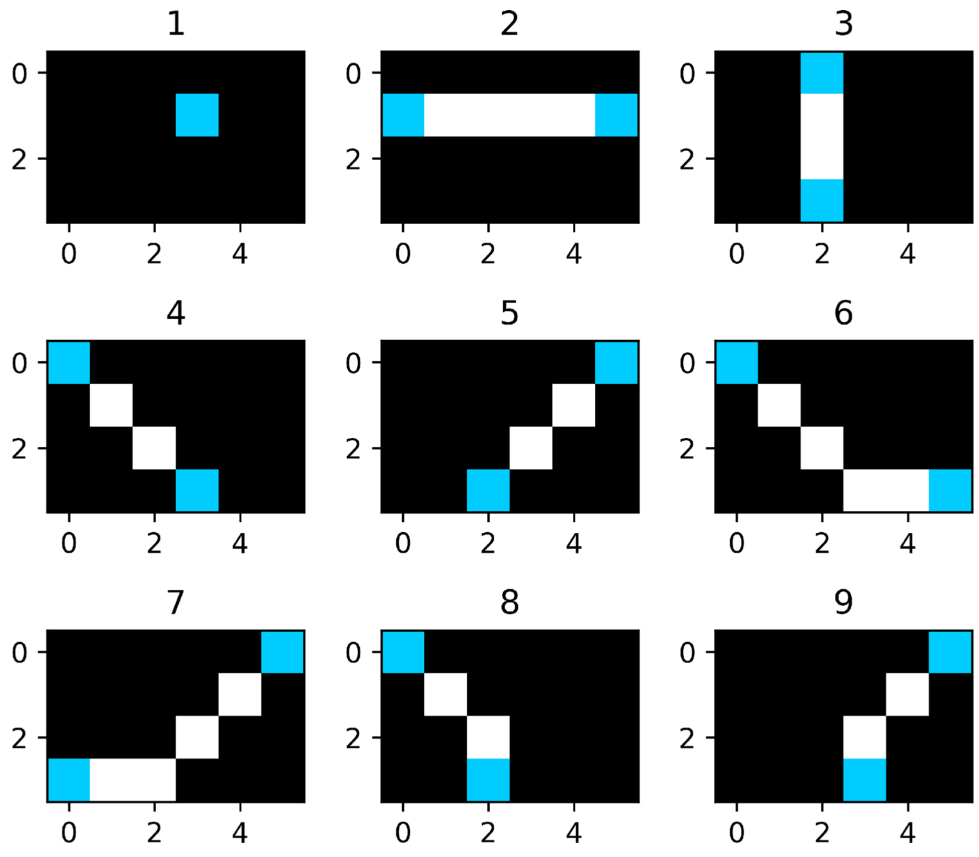


Fig. 6 Cases of scan paths and representation using Algo. 1

Autoencoder Models

An autoencoder is a type of neural network that is trained to duplicate its input as its output [13]. There are different kinds of autoencoders, such as fully-connected autoencoders, convolutional autoencoders, sequence-to-sequence autoencoders, and variational autoencoders. Autoencoders consist of three components including an encoder, a decoder, and a loss function utilized to evaluate the degree of difference modeled by the algorithm between the compressed representation generated from the encoder and the reconstructed representation yielded from the decoder. The purpose of creating a reconstructed representation using an autoencoder model is to have a different distribution of the weights of each gaze point and the relevant scan path to allow CNN models to easily recognize image features. In this study, a convolutional autoencoder [8] was selected and the decompressed representation produced from the decoder of the model was used as input for feature engineering. The same model architecture was kept, but the number of feature maps and the number of strides chosen to use in some convolutional layers as well as some of the max pooling layers were customized. Same padding was applied to the convolutional layer. In the convolutional autoencoder, the encoder portion consists of three 3-by-3 convolutional layers with same padding and a ReLU activation function in each layer. Each convolutional layer is followed by a max pooling layer. The feature map size of the first two max pooling layers is 3-by-3, and the feature map size of the third max pooling layer is 2-by-2. The decoder portion of the convolutional autoencoder is composed of three 3-by-3 convolutional layers, each using same padding and the ReLU activation function. The final 3-by-3 convolutional layer in the decoder uses same padding and the sigmoid activation function. Each

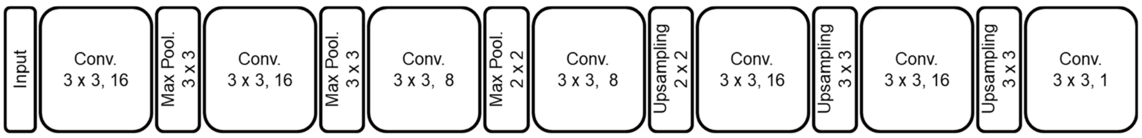


Fig. 7 Convolutional autoencoder model. Conv. = Convolution and Max Pool. = Max Pooling. Kernel sizes for convolution, max pooling and upsampling are denoted as $n \times n$. The number of filters is denoted after a comma in each box having convolution operations

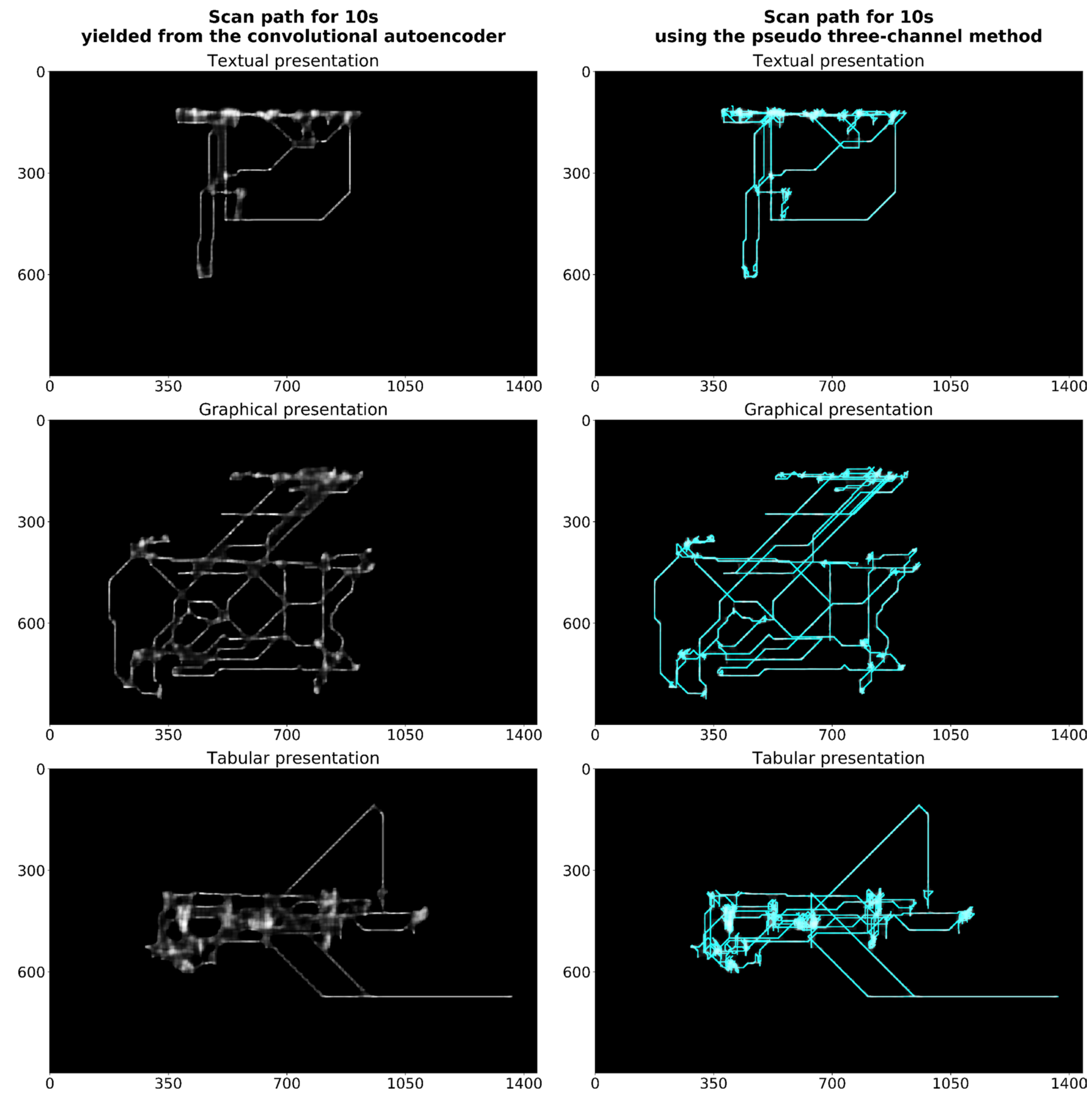


Fig. 8 Three different types of sample images generated from three sample JSON representations within a 10-s time window for each based on three different information presentation methods: textual presentation, graphical presentation, and tabular presentation. The images corresponding to images yielded using Algorithm 1 in the left column were generated from the convolutional autoencoder, and the images in the right column were yielded by using Algorithm 3

convolutional layer in the decoder is also followed by an upsampling layer which is the reverse of the pooling layers in the encoder portion of the model. The filter size of first upsampling layer is 2-by-2, and the filter size of the last two upsampling layers is 3-by-3. In addition, the autoencoder uses binary cross-entropy as the loss function and the Adam optimizer with a learning rate 0.0001. The complete model architecture is illustrated in Fig. 7.

The left columns of Figs. 8 and 9 demonstrate the reconstructed representation generated from the decoder of the

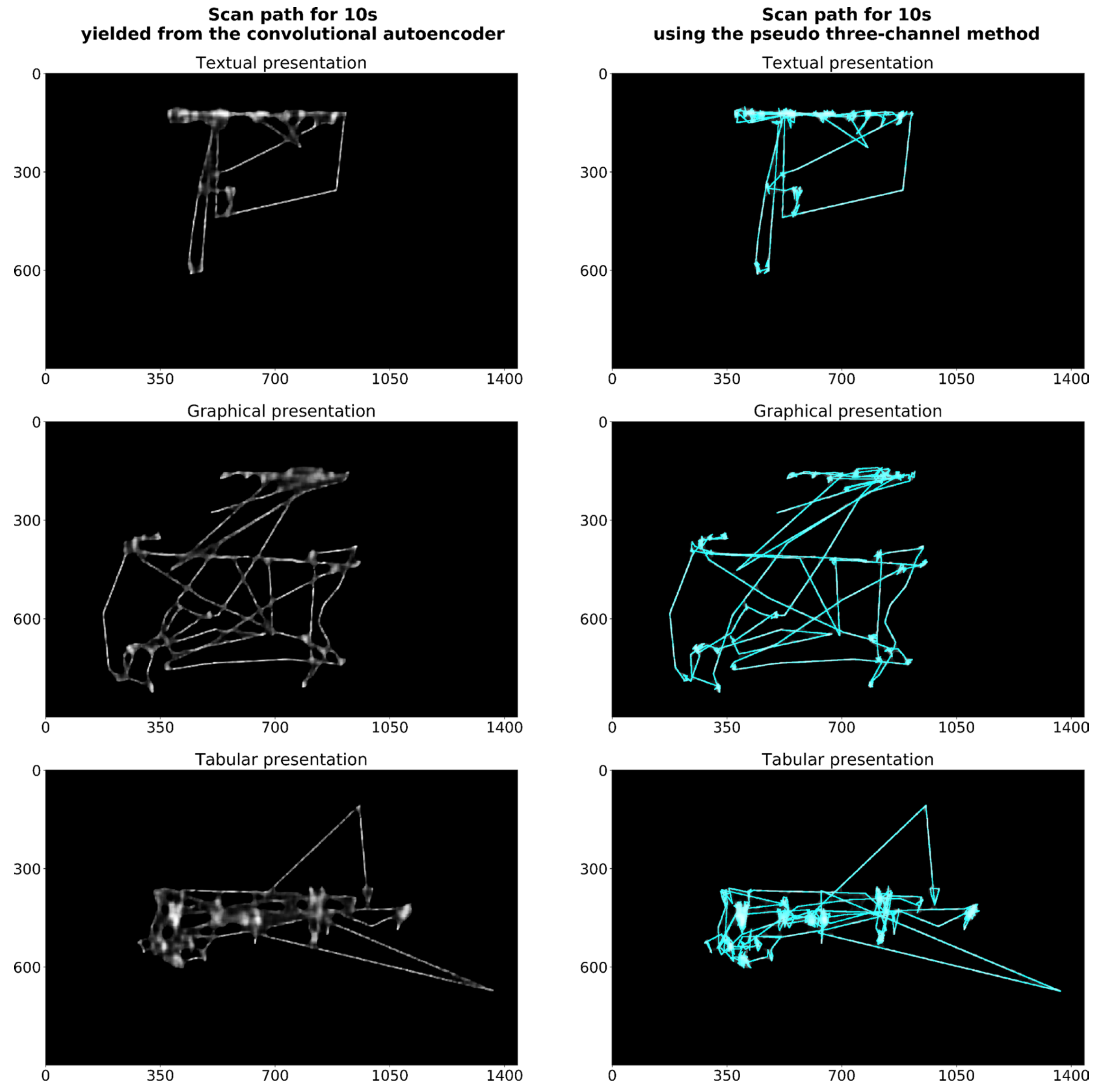


Fig. 9 Three different types of sample images generated from three sample JSON representations within a 10-s time window for each based on three different information presentation methods: textual presentation, graphical presentation, and tabular presentation. The images corresponding to images yielded using Algorithm 2 in the left column were generated from the convolutional autoencoder, and the images in the right column were yielded by using Algorithm 3

convolutional autoencoder for each scan path image for each feature engineering method. The images in this figure directly correspond to the images presented in Fig. 5. In order to train the convolutional autoencoder, the 70/30 holdout method was used to split the eye tracking dataset in our database into a training set and a testing set. In the dataset, there were three categories including textual presentation, graphical presentation, and tabular presentation. For the number of samples of each class in the training set and testing set, the same ratio 70/30 was also used. Two autoencoders were trained using two different types of grayscale images generated by applying the two feature engineering approaches. Each autoencoder model was trained using 100 epochs with a batch size of 8. The model converged quickly and had a very small loss during both training and testing. Each autoencoder model was stored as a JSON file, and the weights of each was saved in HDF5 format.

Pseudo-three-channel Images

Natural three-channel images are composed of red, green and blue channels. In this study, another feature engineering method was used to yield pseudo-three-channel images. The pseudo-three-channel images represent a combination of raw gaze points, results of scan path methods, and results of autoencoder models. The input to Algorithm 3 is a 900-by-1440 2D array stored in the gp_image field of the JSON representation, an ordered list of coordinates of x and y such that $[[x_1, y_1], [x_2, y_2], \dots [x_n, y_n]]$, which is the *coordList* in the gp_sequence field of JSON representation obtained from an ordered set of gaze points $P_{t\_w\_s=10s} \subset P$ where $t\_w\_s = 10s$ denotes that the time window size is 10 seconds, a *scan_path_option* indicating the scan path algorithm, and the *autoencoder*, which is an autoencoder model related to the *scan_path_option* with pretrained weights. The output of the algorithm consists of three 900-by-1440 arrays representing three grayscale channels of a pseudo-three-channel image. These three grayscale images were an output from the convolutional autoencoder, an image generated using processed raw gaze data, and an image produced using information from the gp_sequence field of an JSON representation with applying one of the scan path methods. The architectures of each of the convolutional autoencoders were loaded from the corresponding JSON files with their weights loaded from the related HDF5 files. During the process of creating a pseudo-three-channel image, an autoencoder with its weights received an input of a grayscale image by applying a related scan path feature engineering method. Then, a reconstructed representation for the given input was created. The reconstructed representation was also a grayscale image which was then stacked at the top with an image generated using processed raw gaze data in the middle and a scan path image at the bottom. For the three grayscale images to yield a pseudo-three-channel image, they were produced using the same sequence 10 s slices and the corresponding coordinate pairs of gaze points. In Figs. 8 and 9, the title of each sample image shows the type of information presentation. The right column of each figure illustrates the relevant pseudo-three-channel sample images directly correspond to the sample images presented in Fig. 5. The following algorithm shows the steps for creating the pseudo-three-channel images:

Algorithm 3 Pseudo three-channel method

```
1: Input: An $\mathcal{I} \in \mathbb{R}^{900\times1440}$, $coordList$, $scan\_path\_option$, $autoencoder$
2: Output: $\mathcal{M} \in \mathbb{R}^{900\times1440\times3}$
3: $\mathcal{I}_{normalized} \leftarrow$ Min-max normalization($\mathcal{I}$)
4: if $scan\_path\_option$ = 'Algorithm 1' then
5:     $\mathcal{S\_P\_I} \leftarrow$ use Algorithm 1
6: else if $scan\_path\_option$ = 'Algorithm 2' then
7:     $\mathcal{S\_P\_I} \leftarrow$ use Algorithm 2
8:     $\mathcal{S\_P\_I} \leftarrow \mathcal{S\_P\_I}$ / 255
9: end if
10: $\mathcal{S\_P\_I}_{expanded} \leftarrow$ Expand_dims($\mathcal{S\_P\_I}$)    ▷ Make the dimension as the input dimension of the $autoencoder$.
11: $\mathcal{D\_I} \leftarrow autoencoder$.predict($\mathcal{S\_P\_I}_{expanded}$)
12: $\mathcal{D\_I}_{reshaped} \leftarrow \mathcal{D\_I}$.reshape((900, 1440))
13: $\mathcal{M} \leftarrow$ Stack(($\mathcal{D\_I}_{reshaped}, \mathcal{I}_{normalized}, \mathcal{S\_P\_I}$))
14: Return $\mathcal{M}$
```

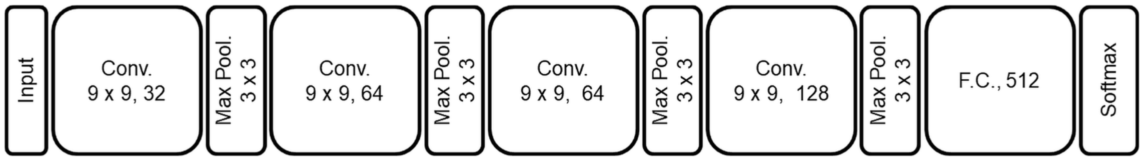


Fig. 10 CNN model based on LeNet-5. Conv. = Convolution, Max Pool. = Max Pooling and F.C. = Fully Connected. Kernel sizes for convolution and max pooling are denoted as $n \times n$. The number of filters is denoted after a comma in each box having convolution operations and in the box showing F.C

Deep Learning Models

LeNet-5 CNN Model

Originally designed to recognize hand-written digits, the LeNet-5 model [32, 33] is the most well-known CNN model. It originally consisted of three convolutional layers, two average pooling layers, one fully connected layer, and one output layer. The key component in this model is the convolutional layer which can be thought of as a feature detector. A feature, also referred to as a receptive field, is represented as a n-by-n filter where n is usually no more than 7. This filter slides over the entire original image with element-wise computation. The result of which is used as input to the next layer and then fed into a nonlinear activation unit. Historically, the sigmoid function or tanh function was chosen as the nonlinear activation unit. Most recently, the rectified linear unit (ReLU) is the method of choice. The ReLU function simply sets negative values to 0 and keeps the positive values the same. Pooling layers are used to merge similar features among a local region like a receptive field. The pooling layer is defined by a m-by-m matrix where m is a small positive integer no more than 10. In LeNet-5, this number was set to 2 so the total image size will be reduced to 1/4 after each pooling layer. Features can then be merged by calculating an average or maximum value. Recent research has found that maximum pooling works better than average pooling. The fully connected layer works like a multi-layered feed forward neural network to classify targets. Every input data to the fully connected layer will result in a predicted value that will be compared with the correct value. Then the loss function is computed and back propagation is used to optimize weight parameters to minimize loss thus train the classifier.

CNN Model Based on LeNet-5

In our previous study [57], we used an adapted version of the LeNet-5 model which consisted of 4 convolutional layers each with the ReLU activation function, 4 max pooling layers, one fully connected layer with the ReLU activation function, and one output layer containing the softmax activation function. The kernel size of each convolutional layer was 9-by-9, and the number of strides was 1. Same padding was used to pad the input so as to make the values of the dimensions of the output shape have the same numbers as the original input. Each max pooling layer had a kernel size of 3-by-3 and a stride of 3 to make sure pooled areas do not overlap. Dropout regularization was applied after the second, third, and fourth max pooling layers as well as the first fully connected layer to avoid the overfitting problem. The dropout rate is the fraction of the features that are zeroed out which is usually set between 0.2 and 0.5 [9]. The dropout rate for the dropout layers after max pooling layers in this study was set to 0.2. The dropout rate for the dropout layer after the first fully connected layer was set to 0.5. The loss function used in the model was categorical cross-entropy. Cross-entropy is a metric from the field of Information Theory that measures the distance between probability distributions or, in this case, between the ground-truth distribution and the predictions [9]. Finally, the Adam optimizer was used with a learning rate 0.0001. The overall model architecture is shown in Fig. 10. The figure also indicates the size and the number of feature maps for each convolutional layer, the window size of each max pooling layer, and the number of neurons in the fully connected layer.

CNN Model with Separable Convolution and Factorization

The second model that was tested included separable convolution and convolution layers. In addition, factorization was applied to the some of convolutions in this model. This model was inspired by [51, 54] where researchers took advantage of the configuration of a 3-by-3 receptive window size called a convolution kernel. In this model, a number of 3-by-3 convolutions are stacked together to replace a single larger convolution. For example, a stack of two 3-by-3 convolutions can substitute for a single 5-by-5 convolution or a stack of three 3-by-3 convolutions can be equal to an effective receptive field whose size is 7-by-7. There are two advantages in applying multiple smaller convolutions instead of a single bigger convolution. First, increased nonlinearity allows the model to better make fine distinctions between features. Second, the computational cost is less since the resulting number of weights is reduced. According to principles of factorization to convolutions presented in the above studies, the 5-by-5 or 7-by-7 convolutions can be replaced with a stack of two and three 3-by-3 convolutions. Therefore, if an input has N channels and a stack of two or three 3-by-3 convolutions have F filters or feature maps for each convolution, then the total number of weights will be $2 \times [(3^2 \times N + 1) \times F]$ or $3 \times [(3^2 \times N + 1) \times F]$, which is $18NF + 2F$ or $27NF + 3F$ with a bias contained in every feature map. Conversely, for a single 5-by-5 or 7-by-7 convolution, the sum of weights will be $(5^2 \times N + 1) \times F$ or $(7^2 \times N + 1) \times F$, which is $25NF + F$ or $49NF + F$ with a bias included in each filter. Thus, a single convolution has many more weights than multiple 3-by-3 convolutions based on the same size of an effective receptive field. An operation called depthwise separable convolution was introduced and used in neural network design in the work [50]. In open-source deep learning libraries, such as Tensorflow and Keras, the depthwise separable convolution is also known as separable convolution. In XceptionNet [10], depthwise separable convolutions were used to substitute Inception

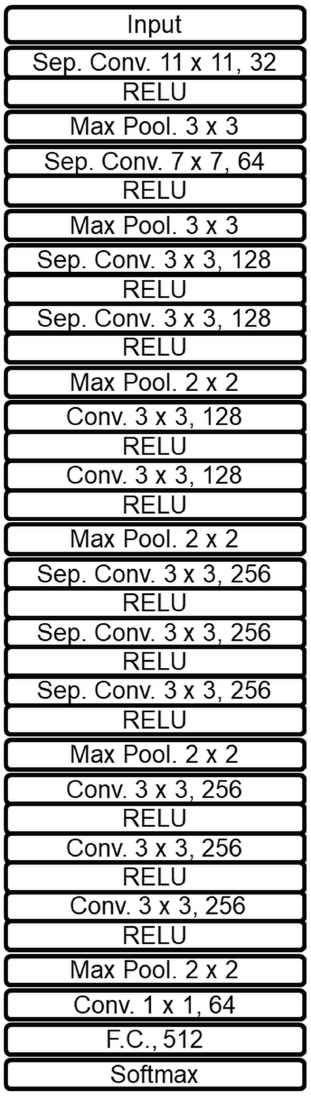


Fig. 11 CNN model with separable convolution and factorization. Sep. Conv. = Separable Convolution, Conv. = Convolution, Max Pool. = Max Pooling and F.C. = Fully Connected. Kernel sizes for separable convolution, convolution and max pooling are denoted as $n \times n$. Notice that all separable convolutional and convolutional layers are followed by batch normalization [25] (not shown in the figure) except for the 1-by-1 convolutional layer. The number of filters is denoted after a comma in each box having separable convolution and convolution operations and in the box showing F.C

modules. A depthwise separable convolution is composed of two stages. The first stage performs spatial convolution on each channel of an input independently instead of throughout the entire depth of an input at one time in a regular convolution. The following stage applies a pointwise convolution such as 1-by-1 convolution with a certain number of filters to the stacked outputs from the first stage. If there is a 13-by-13 input without applying the padding and with N channels, and a 3-by-3 convolution and a 3-by-3 depthwise separable convolution have F filters or feature maps for each convolution, then the total number of multiplications of a regular convolution is $(13-3+1)^2 \times 3^2 \times N \times F$, which is $1089NF$, and the total number of multiplications of a depthwise separable convolution is $(13-3+1)^2 \times 3^2 \times N \times 1 + (13-3+1)^2 \times 1^2 \times N \times F$, which is $121N(9+F)$. Therefore, the benefits of depthwise separable convolutions are less computation time and number of parameters.

Table 2 Data partition

Classification of information presentation types		
	Training set	Testing set
Number of textual presentation	588	148
Number of graphical presentation	360	90
Number of tabular presentation	353	88
Total number of each set	1301	326
Total number of the dataset	1627	

In the CNN model with separable convolution and factorization, the first two separable convolutional layers had kernel sizes of 11-by-11 and 7-by-7, and a stride of one. Each layer was followed by a max pooling layer with a 3-by-3 pooling window size with a stride of three. The next phase of the model consists of four blocks where each block is followed by a max pooling layer with a 2-by-2 pooling window size with a stride of two. The first block had two 3-by-3 depthwise separable convolutions. The second block had two 3-by-3 regular convolutions. The third block had three 3-by-3 depthwise separable convolutions. The fourth block had three 3-by-3 regular convolutions. The number of stride for convolutions of each block was one as well. Separable convolutional and regular convolutional layers from the beginning to these blocks were applied same padding and followed by batch normalization [25] before the output fed into the ReLU activation function. After that, there was a linear layer using 1-by-1 regular convolution without an activation function. The layer can reduce the number of feature maps to have dimensionality with no loss in accuracy [47]. Then, the output of the layer was flattened and fed into a fully connected layer with the ReLU activation function. Dropout regularization was applied to the fully connected layer with a dropout rate that was set to 0.2. The softmax activation function was applied in the output layer. The Adam optimization function with a learning rate of 0.0001 was used and the optimization score function was categorical cross-entropy. The entire model architecture is displayed in Fig. 11.

Using Pretrained CNN Models

The third and fourth models that were tested included two pretrained CNN models combined with classifier layers from the above CNN models. The pretrained CNN models used in this experiment were VGG16 [51] and ResNet50 [20]. Both of these models were trained on the ImageNet dataset [46], which included more than 1 million labeled images with 1000 different categories. For this experiment, the models were used as feature extractors and the dense layers were not used. The purpose of using these pretrained CNNs was to examine whether the spatial hierarchy of features learned by the pretrained models would be useful to help solve our classification problems. The representation of image arrays generated from our eye tracking data were completely different from the photographic images in the ImageNet dataset. Another reason to choose these two pertained models were related to the number of layers which were 16 and 50, respectively, and their depth was not very deep in comparison with the many current neural networks with more than 100 layers. Conversely, the proposed CNN models only have 6 layers and 15 layers, respectively, when the layers including trainable parameters were counted only. As was mentioned before, the densely connected classifiers for these two pretrained networks were excluded because they were related to the 1000 categories from ImageNet. To have a relatively fair comparison with our CNN models, the last three layers in the second CNN model presented in Fig. 11 were attached to the end of the pretrained models as the classifier portion. Also, the last two layers of the first model presented in Fig. 10 were the same as the last two layers in the second model with the classifier portion attached to the pretrained models. In addition, the optimization score function of categorical cross-entropy and the Adam optimization function with the same learning rates as two our CNN models were used.

Experiments and Results

The machine learning platform used in this study was TensorFlow [1, 14], which is end-to-end, open source, and developed by Google. Keras [11, 27] was also used as an API for configuring and running models on top of TensorFlow. The classification task of this study was to classify the three different types of information presentation methods. The models were trained and tested on a single NVIDIA TITAN V GPU with a memory size of 12 GB. In addition, different experiments related to the classification task with different feature engineering models were conducted. In the following, the data partitioning method, the evaluation metrics, details about our experiments, and relevant results will be introduced and described.

Data Partitioning Method

Instead of using the traditional 70/30 training to testing ratio, the 80/20 stratified shuffle split method was used to split the processed dataset in our database into a training set and a testing set for our classification task. This allowed us to use more samples for training and validation. The data partitioning method for this experiment is detailed in Table 2. The class values for each type of information presentation method in our dataset were not perfectly balanced. Because of this, the 80/20 stratified shuffle split method allowed us to ensure that the training and testing sets were split evenly. The training set which contained 80% of the data in our dataset was used to train and evaluate the models through experiments with the stratified tenfold cross-validation method. The final evaluation was performed on the test set that included the 20% of the data.

Evaluation Metrics and Method

A number of metrics were used to evaluate the classification results such as accuracy, precision, recall and F_1 score. Classification accuracy, also known as recognition rate, is the percentage of correct predictions made by the classifier on a given test set. Precision measures how often the prediction result is true positive when a positive value is predicted. Recall measures how sensitive the classifier is at capturing true positives. The F_1 score measures the harmonic mean of precision and recall, and it provides the same weight to precision and recall [17]. A confusion matrix is typically used to calculate the metrics based on the number of true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN). The evaluation metrics can be calculated using 1, 2, 3, and 4, respectively. P refers to the sum of TP and FN. In 2, 3, and 4, the subscript c denotes an associated class label for which the evaluation metric is computed. The ranges of accuracy, precision, recall, and F_1 score are between 0 and 1. 0 is the worst score and 1 is the perfect score.

$$\text{Accuracy} = \frac{\text{number of correct predictions}}{\text{total number of predictions}} \tag{1}$$

$$\text{Precision}_c = \frac{\text{TP}_c}{\text{TP}_c + \text{FP}_c} \tag{2}$$

$$\text{Recall}_c = \frac{\text{TP}_c}{\text{TP}_c + \text{FN}_c} = \frac{\text{TP}_c}{P_c} \tag{3}$$

$$(F_1)_c\ \text{score} = \frac{2 \times \text{Precision}_c \times \text{Recall}_c}{\text{Precision}_c + \text{Recall}_c} \tag{4}$$

Table 3 Test results for the CNN model based on LeNet-5

Classification of information presentation types			
Batch size	8		
1. Raw gaze points			
Test accuracy	0.8650		
	Precision	Recall	F_1 score
Textual presentation	0.8919	0.8919	0.8919
Graphical presentation	0.8427	0.8333	0.8380
Tabular presentation	**0.8427**	0.8523	**0.8475**
2. Scan path using the customized method			
Test accuracy	**0.8712**		
	Precision	Recall	F_1 score
Textual presentation	0.8874	0.9054	**0.8963**
Graphical presentation	0.8916	0.8222	0.8555
Tabular presentation	0.8261	0.8636	0.8444
3. Scan path using the Pillow library			
Test accuracy	0.8528		
	Precision	Recall	F_1 score
Textual presentation	0.8693	0.8986	0.8837
Graphical presentation	0.8824	0.8333	0.8571
Tabular presentation	0.7955	0.7955	0.7955
4. Scan path using the customized method with a convolutional autoencoder			
Test accuracy	0.8589		
	Precision	Recall	F_1 score
Textual presentation	0.8951	0.8649	0.8797
Graphical presentation	**0.9036**	0.8333	**0.8671**
Tabular presentation	0.7700	0.8750	0.8191
5. Scan path using the Pillow library with a convolutional autoencoder			
Test accuracy	0.8681		
	Precision	Recall	F_1 score
Textual presentation	0.8973	0.8851	0.8912
Graphical presentation	0.8556	0.8556	0.8556
Tabular presentation	0.8333	0.8523	0.8427
6. Pseudo-three-channel method with the scan path using the customized method			
Test accuracy	0.8436		
	Precision	Recall	F_1 score
Textual presentation	0.8385	**0.9122**	0.8738
Graphical presentation	0.8554	0.7889	0.8208
Tabular presentation	0.8415	0.7841	0.8118
7. Pseudo-three-channel method with the scan path using the Pillow library			
Test accuracy	0.8558		
	Precision	Recall	F_1 score
Textual presentation	**0.9308**	0.8176	0.8705
Graphical presentation	0.8081	**0.8889**	0.8466
Tabular presentation	0.8041	**0.8864**	0.8432

The values in bold represent the best results of the evaluation metrics of the corresponding CNN model

Table 4 Test results for the CNN model with separable convolution and factorization

Classification of information presentation types			
Batch size	8		
1. Raw gaze points			
Test accuracy	**0.8926**		
	Precision	Recall	F_1 score
Textual presentation	**0.9247**	**0.9122**	**0.9184**
Graphical presentation	0.8864	**0.8667**	**0.8764**
Tabular presentation	**0.8478**	**0.8864**	**0.8667**
2. Scan path using the customized method			
Test accuracy	0.8712		
	Precision	Recall	F_1 score
Textual presentation	0.9000	**0.9122**	0.9060
Graphical presentation	**0.8902**	0.8111	0.8488
Tabular presentation	0.8085	0.8636	0.8352
3. Scan path using the Pillow library			
Test accuracy	0.8466		
	Precision	Recall	F_1 score
Textual presentation	0.8693	0.8986	0.8837
Graphical presentation	0.8642	0.7778	0.8187
Tabular presentation	0.7935	0.8295	0.8111
4. Scan path using the customized method with a convolutional autoencoder			
Test accuracy	0.8466		
	Precision	Recall	F_1 score
Textual presentation	0.8725	0.8784	0.8754
Graphical presentation	0.8605	0.8222	0.8409
Tabular presentation	0.7912	0.8182	0.8045
5. Scan path using the Pillow library with a convolutional autoencoder			
Test accuracy	0.8558		
	Precision	Recall	F_1 score
Textual presentation	0.8912	0.8851	0.8881
Graphical presentation	0.8554	0.7889	0.8208
Tabular presentation	0.8021	0.8750	0.8370
6. Pseudo-three-channel method with the scan path using the customized method			
Test accuracy	0.8681		
	Precision	Recall	F_1 score
Textual presentation	0.9172	0.8986	0.9078
Graphical presentation	0.8588	0.8111	0.8343
Tabular presentation	0.8021	0.8750	0.8370
7. Pseudo-three-channel method with the scan path using the Pillow library			
Test accuracy	0.8313		
	Precision	Recall	F_1 score
Textual presentation	0.8591	0.8649	0.8620
Graphical presentation	0.8434	0.7778	0.8092
Tabular presentation	0.7766	0.8295	0.8022

The values in bold represent the best results of the evaluation metrics of the corresponding CNN model

Table 5 Test results for using pretrained CNN models

Classification of information presentation types						
Batch size	8					
1. Raw gaze points						
Method	VGG16 base			ResNet50 base		
Test accuracy	**0.8926**			0.7730		
	Precision	Recall	F_1 score	Precision	Recall	F_1 score
Textual presentation	**0.9060**	**0.9122**	**0.9091**	0.8800	0.7432	0.8059
Graphical presentation	0.8764	**0.8667**	**0.8715**	0.6066	0.8222	0.6981
Tabular presentation	**0.8864**	**0.8864**	**0.8864**	**0.8608**	0.7727	**0.8144**
2. Scan path using the customized method						
Method	VGG16 base			ResNet50 base		
Test accuracy	0.8589			0.7515		
	Precision	Recall	F_1 score	Precision	Recall	F_1 score
Textual presentation	0.8742	0.8919	0.8829	**0.8929**	0.6757	0.7692
Graphical presentation	0.8642	0.7778	0.8187	0.5906	**0.8333**	0.6912
Tabular presentation	0.8298	0.8864	0.8571	0.8046	0.7955	0.8000
3. Scan path using the Pillow library						
Method	VGG16 base			ResNet50 base		
Test accuracy	0.8374			0.7761		
	Precision	Recall	F_1 score	Precision	Recall	F_1 score
Textual presentation	0.8333	0.9122	0.8710	0.8611	0.8378	**0.8493**
Graphical presentation	0.8375	0.7444	0.7882	0.6737	0.7111	0.6919
Tabular presentation	0.8452	0.8068	0.8256	0.7471	0.7386	0.7429
4. Scan path using the customized method with a convolutional autoencoder						
Method	VGG16 base			ResNet50 base		
Test accuracy	0.7945			**0.7945**		
	Precision	Recall	F_1 score	Precision	Recall	F_1 score
Textual presentation	0.8155	0.9257	0.8671	0.8521	0.8176	0.8345
Graphical presentation	0.8667	0.5778	0.6933	0.8421	0.7111	**0.7711**
Tabular presentation	0.7143	0.7955	0.7527	0.6852	**0.8409**	0.7551
5. Scan path using the Pillow library with a convolutional autoencoder						
Method	VGG16 base			ResNet50 base		
Test accuracy	0.7945			0.7914		
	Precision	Recall	F_1 score	Precision	Recall	F_1 score
Textual presentation	0.8212	0.8378	0.8294	0.8289	0.8514	0.8400
Graphical presentation	0.7949	0.6889	0.7381	0.7619	0.7111	0.7356
Tabular presentation	0.7526	0.8295	0.7892	0.7556	0.7727	0.7640
6. Pseudo-three-channel method with the scan path using the customized method						
Method	VGG16 base			ResNet50 base		
Test accuracy	0.8620			0.7883		
	Precision	Recall	F_1 score	Precision	Recall	F_1 score
Textual presentation	0.8808	0.8986	0.8896	0.7964	0.8986	0.8444
Graphical presentation	0.8488	0.8111	0.8295	**0.9091**	0.5556	0.6897
Tabular presentation	0.8427	0.8523	0.8475	0.7115	**0.8409**	0.7708
7. Pseudo-three-channel method with the scan path using the Pillow library						
Method	VGG16 base			ResNet50 base		
Test accuracy	0.8620			0.7853		
	Precision	Recall	F_1 score	Precision	Recall	F_1 score
Textual presentation	0.8599	**0.9122**	0.8852	0.8667	0.7905	0.8269
Graphical presentation	**0.8987**	0.7889	0.8402	0.6931	0.7778	0.7330
Tabular presentation	0.8333	0.8523	0.8427	0.7667	0.7841	0.7753

The values in bold represent the best results of the evaluation metrics of the corresponding CNN model

When a dataset is divided into three sets containing a training set, a validation set, and a test set, the number of observations that are used for training the classifier will decrease. Consequently, a given random selection of training and validation sets can determine the performance of the classifier. To solve this problem, cross-validation was used to evaluate the performance of the classifiers in a less biased manner. We employed the stratified k-fold cross-validation, which is a variation of k-fold cross-validation. Stratified k-fold cross-validation divides the data into k folds and can guarantee that the same proportions of the given number of types of target classes can be approximately contained in each fold. In addition, stratification is less biased and has slightly lower variance when it is contrasted with regular cross-validation based on both bias and variance [29]. The number of folds was 10, since it had been discovered by experimentation to generate test error rate estimate without undergoing overly high bias or very high variance [26].

Classification of Information Presentation Methods

In the task of classifying three different information presentation methods, seven experiments were conducted for each CNN model. The three target classes were encoded by using the one-hot encoding method such as [1, 0, 0] for textual presentation, [0, 1, 0] for graphical presentation, and [0, 0, 1] for tabular presentation. A number of feature engineering methods were examined in these experiments and the results were compared using the previously discussed evaluation metrics. During the experiments, the tenfold stratified cross-validation method was used to evaluate the performance with respect to the CNN models without the pretrained portion. The same seed values were utilized for the stratified shuffle split partition method and the stratified 10-fold cross-validation method. For each group of 10-fold stratified cross-validation applied in each experiment, a CNN model was trained for 30 epochs with a designated batch size. Then the mean and the standard deviation of the corresponding classification accuracy of the model, precision, recall and F_1 score of each target class in the classification task were computed across the test folds in order to assess the performance of the model. As was mentioned in "Data partitioning method", the 80/20 stratified shuffle split method was used to split the dataset. Before the final evaluation was carried out on the test set containing 20% of the dataset, the whole training set containing 80% of the data was used to train our CNN models, and then the final evaluation was conducted. For the two networks with pretrained portions, all the layers of the pretrained portions were frozen first so as to keep the pretrained weights unchangeable. Then we used the same training set including 80% of the data to train the classifier portion with the same number of training epochs which we did for our two CNNs without pretrained portions. Then the same test set containing 20% of the dataset was used to perform the final evaluation to the models with pretrained portions. Tables 3, 4 and 5 illustrate results with respect to the final evaluation on the test set for all the experiments. Each table contains one experiment without using any feature engineering approaches and six experiments where the feature engineering methods mentioned in "Feature engineering" were applied. In addition, the value range of each evaluation metric listed in the table is between 0 (worst score) and 1 (perfect score).

In the test results for the CNN model based on LeNet-5 with a batch size of 8 (Table 3), experiment 2 using the customized method to create scan paths yielded the best results for test accuracy and the F_1 score of textual presentation. Experiment 5 had the second highest test accuracy, and experiment 4 showed the best F_1 score for graphical presentation, and experiment 1 had the best F_1 score for tabular presentation. In experiments 2 and 3, the feature engineering methods for creating scan paths were different. Experiment 2 had better results for all of the evaluation metrics except for the recall and F_1 score of graphical presentation. However, experiment 3 had better recall and F_1 score of graphical presentation than experiment 2. In experiments 2, 4, and 6, the scan path using the customized method was used in the three different feature engineering approaches. Experiment 2 had the highest test accuracy and F_1 scores of textual presentation and tabular presentation, and experiment 4 had the highest F_1 score of graphical presentation. In experiments 3, 5, and 7, the scan path using the Pillow library was applied to the three different feature engineering approaches. Of the three, experiment 5 had the highest test accuracy and F_1 score of textual presentation. Experiment 3 had the highest F_1 score of graphical presentation, and experiment 7 had the highest F_1 score of tabular presentation. In experiments 4 and 5, a convolutional autoencoder was used with different scan path methods. Experiment 5 had better test accuracy and F_1 scores of textual presentation and tabular presentation than experiment 4. Experiment 4 had better F_1 score of graphical presentation than experiment 5. In experiments 6 and 7, a pseudo-three-channel method was used with different scan path methods. Experiment 7 had better test accuracy and F_1 scores of tabular presentation and graphical presentation scores than experiment 6. Experiment 6 had better F_1 score of textual presentation than experiment 7.

In the test results for the CNN model with separable convolution and factorization with a batch size of 8 (Table 4), the first experiment of using images of raw gaze points as input for the model yielded the best results for all of the evaluation metrics except for the precision of identifying graphical presentation. Overall, the results were promising across all presentation methods with metrics greater than 0.8. In the second experiment using the customized method to create scan paths, it had the same recall metric value as the first experiment. In addition, it had the highest precision metric value of identifying graphical presentation among all the 7 experiments. In experiments 2 and 3, the featured engineering methods for creating scan paths were different. Experiment 2 had better results than experiment 3 for all the evaluation metrics. In experiments 2, 4, and 6, the scan path using the customized method was used in the three different feature engineering approaches. Experiment 2 with the customized method had the best test accuracy and F_1 score of graphical presentation. However, the F_1 scores of textual presentation and tabular presentation for experiment 6 was slightly higher than for experiments 2 and 4. Experiments 3, 5, and 7 were based on the scan path created using the Pillow library. In this experiment, the scan path created with the Pillow library combined with the autoencoder had the highest test accuracy and F_1 scores for classifying all class labels. In experiments 4 and 5, a convolutional autoencoder was used with different scan path methods. Experiment 4 had a better F_1 score of graphical presentation than experiment 5. Experiment 5 had better test accuracy and F_1 scores of textual presentation and tabular presentation than experiment 4. In experiments 6 and 7, the pseudo-three-channel method was used in both with different scan path methods. Experiment 6 had better test accuracy and F_1 scores of all the class labels than experiment 7.

In the test results for the neural network model using the pretrained VGG16 base with a batch size of 8 (Table 5), the first experiment using images of raw gaze points as input for the model yielded the best results for all of the evaluation metrics except for the precision of identifying the graphical presentation method. Overall, the results were promising across all presentation methods with metrics greater than 0.8. In the seventh experiment using the pseudo-three-channel method with the scan path using the Pillow library, it had the same recall metric value as the first experiment. In addition, it had the highest precision metric value of identifying graphical presentation among all the 7 experiments. In experiments 2 and 3, the feature engineering methods for creating scan paths were different. Experiment 2 had better results of most of the evaluation metrics than experiment 3 except for the recall of identifying textual presentation and the precision of identifying tabular presentation. In experiments 2, 4, and 6, the scan path using the customized method was used in the three different feature engineering approaches. Experiment 6 had the best test accuracy and F_1 scores of textual and graphical presentation. However, the F_1 scores of tabular presentation for experiment 2 was slightly higher than experiment 6 and 10% higher than experiment 4. Experiments 3, 5, and 7 were based on the scan path created using the Pillow library. Experiment 7 had the highest test accuracy and F_1 scores for classifying all class labels. In experiments 4 and 5, a convolutional autoencoder was used with different scan path methods. Both had the same test accuracy. Experiment 4 had a better F_1 score of textual presentation than experiment 5. Experiment 5 had better F_1 scores of graphical presentation and tabular presentation than experiment 4. In experiments 6 and 7, the pseudo-three-channel method was used in both with different scan path methods. Both had the same test accuracy as well. Experiment 6 had better F_1 scores of textual and tabular presentation than experiment 7. Experiment 7 had a better F_1 score of graphical presentation than experiment 6.

In the test results for the neural network model using the pretrained ResNet50 base with a batch size of 8 (Table 5), experiment 4, using the customized method to create scan paths with a convolutional autoencoder, yielded the best results for test accuracy and the F_1 score of graphical presentation. Experiment 1 and 3 had the best F_1 scores for tabular and textual presentation respectively. In experiments 2 and 3, the feature engineering methods for creating scan paths were different. Experiment 2 had a better F_1 score of tabular presentation than experiment 3. Experiment 3 had better F_1 scores of textual and graphical presentation than experiment 2. In experiments 2, 4, and 6, the scan path using the customized method was used in the three different feature engineering approaches. Experiment 4 had the highest test accuracy and F_1 score of graphical presentation. Experiment 2 had the best F_1 score of tabular presentation, and experiment 6 had the best F_1 score of textual presentation. In experiments 3, 5, and 7, the scan path using the Pillow library was applied to the three different feature engineering approaches. Of the three, experiment 5 had the highest test accuracy and F_1 score of graphical presentation. Experiment 3 had the highest F_1 score of textual presentation, and experiment 7 had the highest F_1 score of tabular presentation. In experiments 4 and 5, a convolutional autoencoder was used with different scan path methods. Experiment 4 had better test accuracy and F_1 score of graphical presentation. Experiment 5 had better F_1 scores of textual presentation and tabular presentation. In experiments 6 and 7, a pseudo-three-channel method was used with different scan path methods. Experiment 6 had better test accuracy and F_1 score of textual presentation. Experiment 7 had better F_1 scores of tabular presentation and graphical presentation.

In the test results of Tables 3, 4 and 5, the architectures of the neural networks were different, but the batch size used for training the models was the same. In addition, two neural networks had pretrained portions as feature extractors. Experiment 1 for the CNN model with separable convolution and factorization yielded the best results of test accuracy and F_1 scores for identifying the textual and graphical presentation. Also, experiment 1 for the neural network model using pretrained VGG16 base had the same test accuracy as the CNN model with separable convolution and factorization and had the highest F_1 score for identifying the tabular presentation. In addition, the values of the recall metric were the same for these two models. Under this situation, the CNN model with separable convolution and factorization had the highest precision for identifying the textual and graphical presentation, while the neural network model using pre-trained VGG16 base had the highest precision for identifying tabular presentation. The CNN model based on LeNet-5 turned out to have four better results of test accuracy among the experiments which applied feature engineering. The CNN model with separable convolution and factorization had the best test accuracy in experiment 6 among four neural network models. The neural network model using pre-trained VGG16 base had the best accuracy in experiment 7 among four neural network models. Overall, two of our CNN models without using pretrained models had accuracy greater than 80% for all 7 experiments. The neural network model using pretrained VGG16 base had accuracy greater than 80% for 5 experiments. The neural network model using pretrained ResNet50 base had accuracy greater than 75% for all 7 experiments. In addition, the model had better test accuracy for most of experiments where feature engineering approaches were applied in comparison with the experiment 1 without applying any feature engineering approaches.

Discussion and Conclusion

In this study, four CNN models were trained and tested to identify three different types of information presentation methods. Furthermore, a number of feature engineering methods were evaluated with each CNN model. One of the goals of this study was to explore the use of neural networks with a lower cost in terms of GPU resources. Therefore, we took into consideration how neural networks with a small number of convolutional layers could solve our problem with satisfactory results since the number of convolutional layers in the neural networks could influence the level of generalizability of the extracted feature representations. The local and generic feature representations were usually extracted in the earlier layers of the model, and more abstract feature representations were extracted in subsequent layers. Therefore, the feature representations would be more abstract when the depth of the model increases. The image arrays used in this study contained gaze points or scan paths, not real objects such as a cat or a dog. Therefore, more abstract feature representations might not be necessary in our case to improve the accuracy of the classification task. If we compare the number of layers of the two CNN models without using the portions from pretrained CNNs in the classification task, the CNN model with separable convolution and factorization is deeper than the CNN model based on LeNet-5. However, the two CNN models were not deep in comparison with the neural network model using the pretrained ResNet 50 base. Considering the computational cost, the total number of parameters would be a metric to examine. The total number of parameters of the CNN model with separable convolution and factorization had about 4 million units. The CNN model based on LeNet-5 had about 13 million units. Comparing these two models, about 9 million units were reduced, and the relevant computational cost was also lowered. For the neural networks using pretrained VGG16 and ResNet50, the total numbers of parameters were about 56 million and 66 million units, respectively. For these two models excluding the pretrained parameters, the numbers of trainable parameters of both models were about 41 million and 42 million units respectively after we applied a convolutional layer to reduce the number of feature maps. There were two reasons that the numbers of trainable parameters for these models were large. First, the input size of the eye tracking images was 1440-by-900 which is large by common deep learning standards. Second, the output tensor shapes of the last layer of pretrained portions of both of the models were 512 28-by-45 feature maps and

Table 6 Results about misclassified images in all the seven experiments for the CNN model with separable convolution and factorization

Classification of information presentation types								
Test image ID	Ex. 1	Ex. 2	Ex. 3	Ex. 4	Ex. 5	Ex. 6	Ex. 7	True class
199	2	0	2	2	2	2	2	1
240	2	2	0	2	0	2	2	1
270	0	0	0	0	0	0	0	1
394	1	2	2	2	2	2	2	0
649	2	2	2	2	0	2	2	1
978	1	2	1	1	2	1	1	0
1143	1	2	1	1	2	1	1	0
1208	0	0	0	0	0	0	0	2
1238	2	2	2	2	2	2	2	0
1319	0	0	0	0	0	0	0	2
1517	2	2	2	2	2	2	2	1

0, textual; 1, graphical, 2, tabular

2048 29-by-45 feature maps, respectively. Because of this, a 1-by-1 convolutional layer was added to reduce the number of features at the end of the pre-trained model. Otherwise, the size of the output flattened to one-dimensional shape would cause the followed fully connected layer to have many more parameters.

We also found that there was a need to apply feature engineering methods in our study, although the deeper CNN model with separable convolution and factorization trained using images of raw gaze points as input with a batch size of 8 yielded the best results on test accuracy and F_1 scores for classifying the textual and graphical presentation methods. The neural network model using the pretrained VGG16 base applied on this experiment also had the same accuracy as the CNN model with separable convolution and factorization and had the best F_1 score for classifying the tabular presentation method. When experiment 1 was conducted on the CNN model based on LeNet-5 without applying any feature engineering methods, the test accuracy and F_1 scores of classifying textual presentation and graphical presentation were lower than experiment 2 and experiment 5 using the same architecture. Furthermore, the test results of the neural network model using the pretrained ResNet50 base showed that all the experiments using feature engineering approaches except for experiment 2, had better test accuracy than experiment 1 without using any feature engineering approach. Comparison among results of the CNN model based on LeNet-5 and the CNN model with separable convolution and factorization on test accuracy and F_1 scores of classifying three different types of information presentation methods showed an interesting finding. The neural network model with a small number of layers and a large number of parameters had better results among the experiments when applying feature engineering. Using feature engineering methods made features enhanced and complicated, but it may not be beneficial to a neural network model with a small number of parameters. However, in comparison with the neural networks using the pretrained VGG16 and RetNet50 bases, the CNN model with separable convolution and factorization had similar and in most cases, better test results on test accuracy and F_1 scores. Furthermore, the pretrained models had many more parameters and in the case of VGG16, many more layers. Another interesting finding was that feature representations learned from the ImageNet dataset were reusable for our classification problem, since the network model using the pretrained VGG16 base had test accuracy greater than 80% for 5 out of 7 experiments. This finding was surprising to some extent because the images generated from our eye tracking dataset were completely different from the photographic images in the ImageNet dataset.

The misclassified images in the test set were explored to see if any patterns emerged that might explain the reason for misclassifications. The total number of misclassified images was large across all seven experiments. Because of this, the CNN with separable convolution was selected because it had the best results overall. Table 6 shows the misclassifications for the CNN model with separable convolution and factorization. This table shows the ID of the misclassified image, the predicted class labels corresponding to each experiment along with the true class label where 0 indicates textual, 1 indicates graphical, and 2 indicates tabular presentation methods. The table shows that there were five scenarios where the majority of the experiments misclassified the images. This includes textual misclassified as graphical (images 978 and 1143), textual misclassified as tabular (images 394 and 1238), graphical misclassified as tabular (images 199, 240, 649, and 1517), graphical misclassified as textual (image 270), and tabular misclassified as textual (images 1208 and 1319). Images 270, 1208, 1238, 1319, and 1517 were universally misclassified as the same class. However, it must be noted that there were some misclassifications that were mixed where a minority of the experiments resulted in a different misclassification. For example, image 199 was misclassified as textual in experiment 2, image 240 was misclassified as textual in experiments 3 and 5, image 394 was misclassified as graphical in experiment 1, image 649 was misclassified as textual in experiment 5, image 978 was misclassified as tabular in experiments 2 and 5, and image 1143 was classified as tabular in experiments 2 and 5.

Recall that Fig. 2 shows the three information presentation methods from the experiment. Each method influences the eye movements of a user when searching for information to find answers to questions displayed on the user interfaces. Based on this premise, a number of misclassified images were selected to see if any patterns emerge. Figure 12 shows examples for each type of misclassification shown in Table 6. The images were generated using raw gaze points and scan path created using the customized method because these methods had the first- and second-best overall accuracy. Each image contains gaze points for 10 seconds of eye movements. The title of each image shows the true class labels and misclassified labels for the five misclassifications shown in Table 6. The information presented in each interface followed a left to right direction. Because of this, the arrangement of each display could cause a gaze pattern that could confuse the classifier. However, when taking the nationality of each participant into consideration, a pattern for misclassified images emerges. As was mentioned earlier, the eye tracking study from which the data in this experiment was collected consisted of users from North America and Saudi Arabia. A number of the misclassified images came from eye tracking results of the Saudi Arabian participants. In fact, only the third row in Fig. 12 represents a gaze point pattern for a North American participant. The rest of the rows represent gaze point patterns from Saudi Arabian

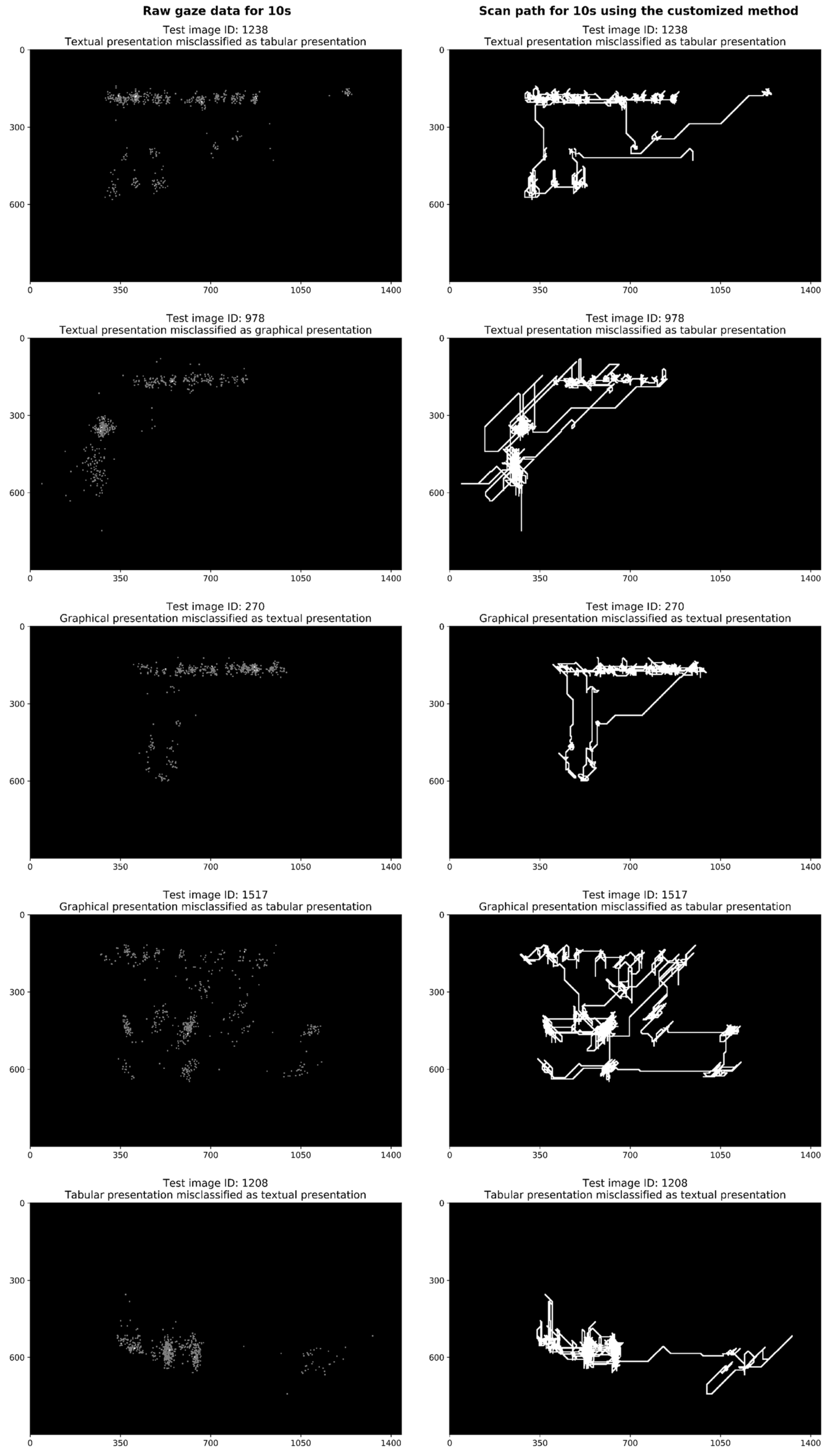


Fig. 12 Ten image samples in all the seven experiments were misclassified by the CNN model with separable convolution and factorization

participants. Notice that the images in the first and second rows of Fig. 12 follow the pattern of an upside-down L. This is an unexpected pattern since the all the text information follows a left to right direction. It is well known that people who read Arabic as their native language, such as the Saudi Arabian participants, typically read in a right to left direction. Because of this, one explanation for the misclassified images is that the reading pattern of a user's native language can influence the gaze pattern and confuse the classification algorithm. The results of this study can only suggest this as a phenomenon and therefore further research is required to verify this conclusion.

There are a number of ways in which this work can be extended. For example, in this paper, positions of gaze points were used for the classification task and the time dimension was not completely exploited. Since sequences of gaze points were transformed into static images based on a time window size, this could be a limitation. Relevant evaluation measures can be affected if we exploit additional feature engineering methods. One of the biggest challenges in using CNN models is how to tune the hyperparameters since there are a lot of combinations of hyperparameters such as different learning rates and number of hidden layers and units. Furthermore, different combinations of parameters may affect the performance of the CNN models directly. In addition, the size of the array generated from gaze points is large (900-by-1440) and sparse with few non-zero entries. Also, the number of data samples is limited because of the small number of users in each study. Overfitting also existed on the CNN models utilized to perform the classification task in this study. If the image size can be reduced to a smaller size by using a compression method, the performance of CNN models may be further improved. As was mentioned before, the time dimension was not fully utilized. Future research could propose a sequence classification problem with eye tracking data if the time dimension can be completely exploited. This would require further study to determine the proper method to create sequences from eye tracking data. Finally, other deep learning models such as recurrent neural networks (RNNs) can be explored to solve the problem. Feature representations learned by our CNN models and generated by the network models using pretrained bases will be analyzed in the future study. The analysis will be beneficial to adjust our CNN model architectures in order to improve our CNN model performance. In addition, the analysis may provide insight about how users interacted with the interfaces corresponding to these three information representation methods, allowing us to propose recommendations on how to design efficient and reliable user interfaces that support users with diversified needs and capabilities.

Acknowledgements This research was partially supported by the Towson University School of Emerging Technologies. We also gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan V GPU used for this research.

Compliance with Ethical Standards

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

Research involving human participants and/or animals The study protocol was approved by the Towson University IRB (Exemption number: 14-X145).

Informed consent Informed consent was obtained from all individual participants included in the study.

References

1. Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, Davis A, Dean J, Devin M, Ghemawat S, Goodfellow I, Harp A, Irving G, Isard M, Jia Y, Jozefowicz R, Kaiser L, Kudlur M, Levenberg J, Mane D, Monga R, Moore S, Murray D, Olah C, Schuster M, Shlens J, Steiner B, Sutskever I, Talwar K, Tucker P, Vanhoucke V, Vasudevan V, Viegas F, Vinyals O, Warden P, Wattenberg M, Wicke M, Yu Y, Zheng X. Tensorflow: large-scale machine learning on heterogeneous distributed systems. 2016.
2. Alqahtani Y, Chakraborty J, McGuire M, Feng JH. Understanding visual information processing for American vs. Saudi Arabian users. In: Di Bucchianico G, editor. Advances in design for inclusion. Cham: Springer International Publishing; 2020. p. 229–38.
3. Alqahtani Y, McGuire M, Chakraborty J, Feng JH. Understanding how ADHD affects visual information processing. In: Antona M, Stephanidis C, editors. Universal access in human-computer interaction. multimodality and assistive environments. Cham: Springer International Publishing; 2019. p. 23–31.
4. Banker K, Bakkum P, Verch S, Garrett D, Hawkins T. MongoDB in action. 2nd ed. Shelter Island: Manning Publications Co.; 2016.
5. Benyon D, Innocent P, Murray D. System adaptivity and the modelling of stereotypes. In: Bullinger HJ, Shackel B, editors. Human-computer interaction-INTERACT '87. Amsterdam: North-Holland; 1987. p. 245–53. https://doi.org/10.1016/B978-0-444-70304-0.50047-9.
6. Chatfield K, Simonyan K, Vedaldi A, Zisserman A. Return of the devil in the details: delving deep into convolutional nets. In: BMVC 2014—proceedings of the British machine vision conference 2014; 2014. https://doi.org/10.5244/C.28.6.
7. Chen Z, Fu H, Lo WL, Chi Z. Strabismus recognition using eye-tracking data and convolutional neural networks. J Healthc Eng. 2018;2018:1–9. https://doi.org/10.1155/2018/7692198.
8. Chollet F. Building autoencoders in keras. 2016. https://blog.keras.io/building-autoencoders-in-keras.html. Accessed 15 Jun 2020.
9. Chollet F. Deep learning with python. Shelter Island: Manning Publications Co.; 2017.
10. Chollet F. Xception: deep learning with depthwise separable convolutions; 2017. pp. 1800–1807. https://doi.org/10.1109/CVPR.2017.195.
11. Chollet F, et al. Keras. 2015. https://github.com/fchollet/keras. Accessed 15 Jun 2020.
12. Collette A. Python and HDF5. Sebastopol: O'Reilly; 2013.

13. Goodfellow I, Bengio Y, Courville A. Deep Learning. Cambridge: MIT Press; 2016. http://www.deeplearningbook.org. Accessed 1 Jul 2020.
14. Google. Tensorflow. 2020. https://www.tensorflow.org/. Accessed 15 Jun 2020.
15. Groen M, Noyes J. Using eye tracking to evaluate usability of user interfaces: is it warranted? IFAC Proc Vol. 2010;43(13):489–93. https://doi.org/10.3182/20100831-4-FR-2021.00086 11th IFAC/IFIP/IFORS/IEA Symposium on Analysis, Design, and Evaluation of Human-Machine Systems.
16. Gullà F, Cavalieri L, Ceccacci S, Germani M, Bevilacqua R. Method to design adaptable and adaptive user interfaces. In: Stephanidis C, editor. HCI international 2015—posters' extended abstracts. Cham: Springer International Publishing; 2015. p. 19–24.
17. Han J, Kamber M, Pei J. Data mining concepts and techniques. 3rd ed. Waltham: Morgan Kaufmann; 2011.
18. Hardzeyeu V, Klefenz F, Schikowski P. Vision assistant: a human–computer interface based on adaptive eye-tracking; 2006. pp. 175–182.https://doi.org/10.2495/DN060171.
19. He K, Gkioxari G, Dollár P, Girshick R. Mask r-cnn. In: 2017 IEEE international conference on computer vision (ICCV); 2017. pp. 2980–2988. https://doi.org/10.1109/ICCV.2017.322.
20. He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR); 2016. pp. 770–778. https://doi.org/10.1109/CVPR.2016.90.
21. Hinton G, Li Deng, Yu D, Dahl G, Rahman Mohamed A, Jaitly N, Senior A, Vanhoucke V, Nguyen P, Sainath T, Kingsbury B. Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. Sig Process Mag IEEE. 2012;29(6):82–97. https://doi.org/10.1109/MSP.2012.2205597.
22. Hinton G, Osindero S, Teh YW. A fast learning algorithm for deep belief nets. Neural Comput. 2006;18(7):1527–54. https://doi.org/10.1162/neco.2006.18.7.1527.
23. Huang B, Chen R, Zhou Q, Xu W. Eye landmarks detection via weakly supervised learning. Pattern Recogn. 2020;98:107076. https://doi.org/10.1016/j.patcog.2019.107076.
24. Hutt S, Mills C, White S, Donnelly P, Mello S. The eyes have it: Gaze-based detection of mind wandering during learning with an intelligent tutoring system. In: 9th international conference on educational data mining, 2016.
25. Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: Bach F, Blei D, editors. Proceedings of the 32nd International Conference on Machine Learning, Proceedings of Machine Learning Research. Lille: PMLR; vol. 37, 2015. pp. 448–456.
26. James G, Witten D, Hastie T, Tibshirani R. An introduction to statistical learning: with applications in R. 1st ed. New York: Springer; 2013.
27. Keras. Keras. 2020. https://keras.io/. Accessed 15 Jun 2020.
28. Kim J, Kim B, Roy PP, Jeong D. Efficient facial expression recognition algorithm based on hierarchical deep neural network structure. IEEE Access. 2019;7:41273–85. https://doi.org/10.1109/ACCESS.2019.2907327.
29. Kohavi R. A study of cross-validation and bootstrap for accuracy estimation and model selection. In: International joint conference on artificial intelligence, vol. 14; 1995.
30. Krizhevsky A, Sutskever I, Hinton G. Imagenet classification with deep convolutional neural networks. Neural Inf Process Syst. 2012;25:1097–105. https://doi.org/10.1145/3065386.
31. LeCun Y, Bengio Y, Hinton G. Deep learning. Nature. 2015;521:436–44. https://doi.org/10.1038/nature14539.
32. Lecun Y, Boser B, Denker J, Henderson D, Howard R, Hubbard W, Jackel L. Handwritten digit recognition with a back-propagation network. Neural Inf Process Syst. 1989;2:396–404.
33. Lecun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. Proc IEEE. 1998;86:2278–324. https://doi.org/10.1109/5.726791.
34. Liu W, Anguelov D, Erhan D, Szegedy C, Reed S, Fu CY, Berg A. Ssd: Single shot multibox detector. In: ECCV; 2016.
35. Minar MR, Naher J. Recent advances in deep learning: an overview; 2018. https://doi.org/10.13140/RG.2.2.24831.10403.
36. Minsky ML. Steps toward artificial intelligence. In: Proceedings of the institute of radio engineers. 2000;49.
37. Naqshbandi K, Gedeon T, Abdulla U. Automatic clustering of eye gaze data for machine learning. 2016;001239–44. https://doi.org/10.1109/SMC.2016.7844411.
38. Norcio AF, Stanley J. Adaptive human-computer interfaces: a literature survey and perspective. IEEE Trans Syst Man Cybern. 1989;19(2):399–408. https://doi.org/10.1109/21.31042.
39. Park SJ, Kim BG. Development of low-cost vision-based eye tracking algorithm for information augmented interactive system. J Multimed Inf Syst. 2020;7:11–6. https://doi.org/10.33851/JMIS.2020.7.1.11.
40. Pillow. Pillow. 2020. https://pillow.readthedocs.io/en/stable/. Accessed 15 Jun 2020.
41. Poole A, Ball LJ. Eye tracking in hci and usability research. In: Ghaoui C, editor. Encyclopedia of human computer interaction; 2005. pp. 211–219.: Idea Group Reference.
42. Rakhmatulin I, Duchowski AT. Deep neural networks for low-cost eye tracking. Proc Comput Sci. 2020;176:685–94. https://doi.org/10.1016/j.procs.2020.09.041 Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 24th International Conference KES2020.
43. Redmon J, Divvala S, Girshick R, Farhadi A. You only look once: unified, real-time object detection. In: The IEEE conference on computer vision and pattern recognition (CVPR); 2016. pp. 779–788. https://doi.org/10.1109/CVPR.2016.91.
44. Redmon J, Farhadi A. Yolov3: an incremental improvement; 2018. ArXiv:1804.02767.
45. Riener A, Boll SC, Kun AL. Automotive user interfaces in the age of automation (dagstuhl seminar 16262). Dagstuhl Rep. 2016;6:111–59.
46. Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S, Huang Z, Karpathy A, Khosla A, Bernstein M, Berg AC, Fei-Fei L. ImageNet large scale visual recognition challenge. Int J Comput Vis (IJCV). 2015;115(3):211–52. https://doi.org/10.1007/s11263-015-0816-y.
47. Sainath TN, Peddinti V, Kingsbury B, Fousek P, Ramabhadran B, Nahamoo D. Deep scattering spectra with deep neural networks for lvcsr tasks. In: INTERSPEECH, 2014.
48. Schall A, Bergstrom JR. Eye tracking in user experience design. Waltham: Elsevier Inc.; 2014.
49. Schmidhuber J. Deep learning in neural networks: an overview. Neural Netw. 2014;61:85–117. https://doi.org/10.1016/j.neunet.2014.09.003.
50. Sifre L. Rigid-motion scattering for image classification. Ph.D. thesis 2014.
51. Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. In: International conference on learning representations, 2015.
52. Smith B. Beginning JSON. New York: Apress; 2015.
53. Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A. Going deeper with convolutions. In: The IEEE conference on computer vision and pattern recognition (CVPR); 2015. pp. 1–9. https://doi.org/10.1109/CVPR.2015.7298594.
54. Szegedy C, Vanhoucke V, Ioffe S, Shlens J, Wojna Z. Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE conference on computer vision and pattern recognition

(CVPR); 2016. pp. 2818–2826. Piscataway: IEEE. https://doi.org/10.1109/CVPR.2016.308.

55. Ulahannan A, Jennings P, Oliveira L, Birrell S. Designing an adaptive interface: using eye tracking to classify how information usage changes over time in partially automated vehicles. IEEE Access. 2020;8:16865–75. https://doi.org/10.1109/ACCESS.2020.2966928.
56. Xie S, Hu H. Facial expression recognition using hierarchical features with deep comprehensive multipatches aggregation convolutional neural networks. IEEE Trans Multimed. 2019;21(1):211–20. https://doi.org/10.1109/TMM.2018.2844085.
57. Yin Y, Juan C, Chakraborty J, McGuire MP. Classification of eye tracking data using a convolutional neural network. In: 17th IEEE international conference on machine learning and applications (ICMLA); 2018. pp. 530–535. Orlando, FL: IEEE. https://doi.org/10.1109/ICMLA.2018.00085.
58. Yoon HJ, Alamudun F, Hudson K, Morin-Ducote G, Tourassi G. Deep gaze velocity analysis during mammographic reading for biometric identification of radiologists. J Hum Perform Extrem Environ. 2018;. https://doi.org/10.7771/2327-2937.1088.
59. Zemblys R, Niehorster D, Komogortsev O, Holmqvist K. Using machine learning to detect events in eye-tracking data. Behav Res Methods. 2017;. https://doi.org/10.3758/s13428-017-0860-3.