Visual Deep Learning Recommender System for Personal Computer Users

Daniel Shapiro*†, Hamza Qassoud*†, Mathieu Lemay† and Miodrag Bolic*
*School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, Ontario, Canada
Email: {dshap092, hqass076, mbolic}@eecs.uottawa.ca
†Clockrr Inc., and Lemay Solutions Consulting Inc., Ottawa, Ontario, Canada
Email: {daniel, matt}@lemaysolutions.com

***Abstract*—This work presents a new architecture for creating virtual assistants on personal computers, building upon prior work on deep learning neural networks, image processing, mixed-initiative systems, and recommender systems. Recent progress in virtual assistants enables them to converse with users and interpret what the user sees. These systems can understand the world in intuitive ways with neural networks, and make action recommendations to the user. The assistant architecture in this work is described at the component level. It interprets a computer screen image in order to produce action recommendations to assist the user. It can assist in automating various tasks such as genetics research, computer programming, engaging with social media, and legal research. The action recommendations are personalized to the user, and are produced without integration of the assistant into each individual application executing on the computer. Recommendations can be accepted with a single mouse click by the computer user.**

Keywords–Recommender systems; Image processing; Deep learning.

I. INTRODUCTION

Personal computers typically push the cognitive effort of solving a problem or discovering a task onto the user, requiring the user to take the initiative and tell the computer how to solve the problem based upon hints provided by the computer. For example, a search engine box presents a blank slate to the user, who must decide what to type into the box, and similarly, an error message can lead the user to search in a search engine for a solution, rather than the computer offering a solution to the problem.

Virtual assistants can take many forms, including assistance hardware appliances [1], cloud-based assistants [2], and application specific assistants [3]. Advances in artificial intelligence have enabled the user to express her intent using voice commands [4]–[12]. Additional recent advances in mobile computing such as Google Now on Tap enable a mobile phone to interpret a picture of the phone's screen and recommend relevant actions and information [13] [14]. This type of image processing recommender system represents a step in the right direction towards intelligent anticipatory computing. However, Google Now on Tap is not available for personal computers, where many users work, and it is not a proactive system. In other words, Google Now on Tap waits for the user to request recommendations by pressing a button. With the exception of reminders, personal computer assistants, such as [15] and [16], were designed to require the user to express intent in order to access a recommendation [17].

Anticipatory computing with artificial intelligence can assist users in real-time as they interact with personal computers [18]. However, the basic mismatch in initiative between the computer and user remains unresolved. Virtual assistants for personal computers can converse with users more naturally than ever, but cannot see what the user sees, or take the initiative to understand the computer screen in some intuitive way and provide action recommendations to the user. Current efforts revolve around extracting the intent of the user from the user's input, transforming the identified intent into a query, and then returning query results to the user. Some systems go further and attempt to support interactive dialog, reminders, and application launching. This command and dialog approach to virtual assistants continues to place the onus of specifying intent squarely onto the user. It is that cognitive pressure applied to user by the assistant - the need to have the user specify what the user wants - that is a limitation of existing works.

To reduce the cognitive pressure on the user, this work presents a virtual assistant called Automated Virtual Recommendation Agent (AVRA). AVRA follows a Mixed-Initiative (MI) approach to human-computer interaction, where a human and virtual agent work together to achieve common goals. The approach is to offload to AVRA some of the cognitive pressure of understanding onscreen problems and goals visible on the computer screen, and recommending actions to the user as solutions.

At the heart of AVRA is a Recommender System (RS) which provides action recommendations to the user on its own initiative. This approach is different from contemporary user assistance software, which typically provides a text box for the user to enter a command as in [19], voice recognition to speak commands to the computer as in [20], or other such user-driven computer interaction mechanisms. The difference in AVRA is that there is no dialog for user input. AVRA observes the visible information on the computer screen to produce recommendations on its own, and the user can accept or reject the advice from AVRA.

The contribution of this work is to describe the architecture of a virtual assistant with shallow application integration. Related work is discussed in Section III. The process of training AVRA to recognize contexts and keywords is described in Section IV. In Section V, AVRA is evaluated using execution time, recall, precision, precision-recall and Receiver Operating Characteristic (ROC) curves as metrics. Section VI contains a summary of this work and a discussion of future research directions.

II. SYSTEM OVERVIEW

AVRA follows a series of steps in order to produce recommendations. The primary goal of providing action recommendations based upon an image of the computer screen leads to several sub-tasks such as context identification (Section II-A),

 ISBN: 978-1-61208-577-7

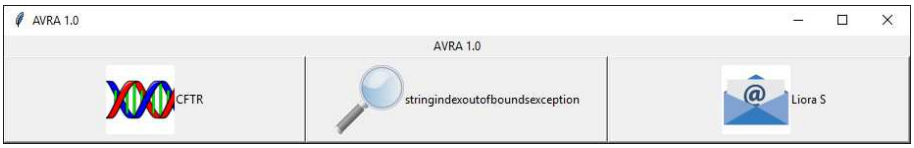

Figure 1. AVRA's graphical user interface showing recommendations to explore a specific gene name, a Java programming error, and compose an e-mail.

context-specific text filtering (Section II-B), context-specific character-level classification of onscreeen text (Section II-C), and objective ranking of all possible recommendations and re-ranking these recommendations according to the user history (Section II-D).

To provide the user with action recommendations, AVRA begins by detecting the particular graphical contexts appearing on the computer screen. This is followed by the detection of tasks within those contexts for which the RS knows how to help the user. Next, it ranks the recommendations for the user so that they are customized to the user's past behavior. Finally, the RS presents three action recommendations to the user via a Graphical User Interface (GUI), as shown in Figure 1. The buttons are decorated with meaningful icons, making it clear what action recommendations are being offered to the user, and reducing the footprint of the GUI on the screen. Further insight into why the recommendation was made is available as a tooltip when the user's mouse scrolls over a GUI button. Interaction with the GUI is very simple, limiting the scope of the interaction between the RS and the user. When the user clicks on one of the GUI buttons, the action corresponding to the solution recommendation in that button is executed. These actions may be email composition, opening a browser window to a particular website, opening a document, etc.

Action recommendations are presented as binary choices in the GUI, limiting the scope of user feedback to clicking or not clicking buttons. This simple interaction makes the user experience significantly more intuitive than a voice command system, because the user does not know with certainty when issuing a voice command that the computer will comply correctly with the command, whereas AVRA advertises what actions it proposes to perform prior to the user commanding the action to be performed. Furthermore, the GUI only updates the recommendations when the mouse is stationary and not hovering over the GUI. This way, if the user sees a recommendation in the GUI and begins to move the mouse toward the buttons, the recommendation will not be replaced, as this would likely annoy the user. Similarly, if the user is pondering a recommendation (perhaps reading the tooltip text while hovering the mouse over a button in the GUI), the action recommendation should not be replaced. This type of confidence building is important in MI system design and is discussed in [21] and [22].

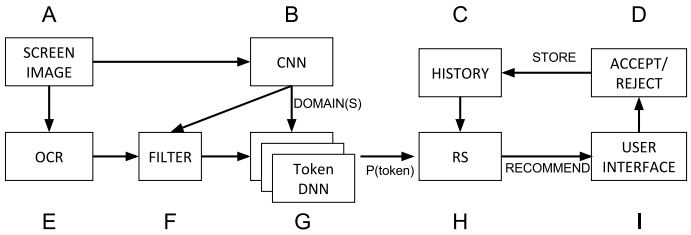

Figure 2. AVRA System Overview.

Like many artificial intelligence platforms, AVRA s implemented as a web service. Each client repeatedly captures an image of the screen which is then interpreted by a server-side program. The server processes the image by parsing and classifying the text and graphics on the computer screen. The server responds to the client's image submission with a set of 3 personalized action recommendations presented to the user.

The client consists of components A and I in Figure 2, while the server implements components $B - H$. The advantages of placing most system components on the server in a thin-client design include the ability to leverage server-side GPU acceleration, having the user profile accessible from multiple computers as a user moves from machine to machine, and having access to the user profiles of many users for future work on collaborative filtering of recommendations.

This approach to assisting the user generalizes to non-programming domains where the user researches items that appear onscreen. For example, when the user is browsing social media, the RS can detect the names of friends and recommend an action such as composing an email to the detected contact when that friend's name appears onscreen. Another example developed in AVRA is genetic research, where the user is reading a PDF document involving genetic research. The RS can detect this genetic research context, and propose to open a browser window to a web page detailing the relationship between a gene name recognized on the computer screen and other genes.

A. Context Identification

A computer screen image may contain graphical information regarding several topics. For example, a terminal window alongside a browser window conveys graphical information on these topic. Each topic detected in the computer screen image is called a context, and each context is recognized by a trained Convolutional Neural Network (CNN). The image features within a context can be the contents of a program such as a document format, or the look and feel of a program such as menus and color schemes. Figure 2 shows that each screen capture is parsed by a CNN (Figure 2 B) to identify what contexts should be explored for each image. The context recognition CNN that steers recommendation generation is not tightly integrated into each specific application producing onscreen information. Rather, an image of the computer screen is processed by the CNN classifier to detect contexts, and these detected contexts guide text filtering and classification units, which search the onscreen text for keywords based upon which AVRA can recommend actions. This shallow integration of an RS into the programs executing on a personal computer system is novel. Because of this shallow integration approach, AVRA has visibility over all programs visible to the user, and can therefore help with multitasking, whereas an isolated program restricted to text or voice commands has insufficient visibility into the workspace (i.e., the computer screen) to make such recommendations. Effectively, AVRA sees what the user sees.

In AVRA, the classifier can make more than one prediction per image. Multiple different contexts can be present on the

computer screen at the same time, and therefore consider that one image of the computer screen containing side-by-side windows of the Eclipse IDE and a console window could trigger two predictions which may both be accurate.

Through supervised learning, AVRA is initialized with knowledge of context-detection for certain domains, context-specific textual terms (called keywords), and action recommendations corresponding to each possible context-specific keyword detection.

B. Text Filtering

Optical Character Recognition (OCR) (Figure 2 E) converts the screen image (Figure 2 A) into a string of text to be parsed. The raw text from the OCR system (Figure 2 E) is cut into many small text segments, which are then very quickly filtered (Figure 2 F), selecting for a minimum similarity between the candidate text and keywords of each detected context (Figure 2 B). This filtering step is necessary to reduce the number of text segments reaching the relatively slow text classifier (Figure 2 G).

C. Character-level Text Classification

A Deep Neural Network (DNN) for each activated context (Figure 2 G) processes the text output of the text filtering stage (Figure 2 F) using character-level neural network classification to detect keywords within the text that are associated with recommendations stored in the RS database. Each DNN is trained by processing an image of the keyword text through OCR software (generated using a text-to-image generator), and then encoding the results into vectors that include letter frequency information. The training process is described in [23], and the goal of the approach is to robustly detect keywords even in the presence of OCR output spelling errors and frame-shifts in the text.

D. Recommender System

The recommendations generated by the DNNs are filtered (Figure 2 H) to identify the top 3 ranked recommendations to be presented in the user interface (Figure 2 I). The user may accept recommendations by clicking a button in the user interface, causing the corresponding action(s) to be executed on the computer (Figure 2 D) and the user history is also updated (Figure 2 C).

AVRA provides personalized recommendations that adapt over time to the changing behavior of the user. The RS learns from the history of user interactions with AVRA to adapt recommendations to the user's needs. Employing a virtual assistant in this way is a novel approach to helping the user with tasks. In the programming example, AVRA can share with the user the task of identifying the error message on the computer screen, looking up actions which may help the user with the error, and identifying the most relevant action recommendations. This approach puts little or no pressure on the mind of the user in the course of normal computer use.

AVRA includes a probabilistic hybrid RS model, where the RS model combines the classification confidence level associated with each action recommendation with the confidence level associated to the each context detected on the computer screen. These two confidence levels are further combined with the user's preferences (history of clicks and rejections of action recommendations). Following this adaptive RS approach, the recommendations generated by a given image of the computer screen will change over time based upon the actions of the user.

III. Related Work

Using reinforcement learning, specifically a deep Q-network, [24] developed a deep learning artificial intelligence system that can play many different video games using computer screen images as input and joystick positions as the output. The breadth of different applications played by the same neural network was an impressive validation of the power of reinforcement learning. The overlap between AVRA and [24] is the use of a CNN to process the image of the screen, and then using fully connected layers of a DNN to make a decision. In the case of AVRA, the DNN output is a recommendation to be ranked, whereas in [24] the DNN outputs represent joystick positions. Also, in [24] the goal of the CNN in the system was regression, and the CNN did not perform pooling (down-sampling), while in contrast AVRA's CNN does perform down-sampling via pooling, as the AVRA CNN is outputting a classification rather than a target score. Furthermore, unlike [24], AVRA's CNN is connected to multiple DNNs in a low resolution way (either activating a DNN or not), while the CNN from [24] were connected directly into the fully connected layers of the DNN.

The shallow integration between AVRA and the applications executing on the user's computer relates to prior work on high-level query processing proposed in [25]. The concept in [25] is that shallow clues in a query can hint at the correct databases to search. Therefore, a query to many databases need not be written for each database; rather, a high-level intermediate query engine can dynamically steer the query to the right databases. In AVRA, the shallow search is performed by a CNN to detect contexts, and then DNNs, along with filtering algorithms and OCR, perform deeper context-driven analysis. The key information to understand from [25] in this work is that the low-level context by context search was bypassed, and instead a dynamic high-level context search was achieved.

A. Recommender Systems

An RS must score and rank recommendations in order to present high-quality options to the user. Content-Based Filtering (CBF) is an approach where the recommendation score is increased if related items were rated positively in the past. CBF makes it more likely for items and topics preferred in the past to be recommended in the future [26] [27]. As in [28], this work takes the approach to modify the score of a recommendation based on both topic and item similarity. Similar to [29], in which the user's tagging history informs the likelihood of recommending an item, AVRA uses the user's history of accepting and rejecting recommendations to modify the likelihood of new recommendations over time. Specifically, action recommendations in AVRA are biased toward actions and contexts where the user clicked to accept a recommendation. To a smaller extent action recommendations are negatively biased toward recommendations that were not accepted by the user. AVRA includes a probabilistic hybrid RS model similar to [30].

Combining CBF with a taxonomic record of user preferences improves the quality of recommendations [31]. AVRA

 ISBN: 978-1-61208-577-7

follows this approach by scoring recommendations based upon context, while taking into account user preferences. Similarly, [32] combines CBF with knowledge of the domain. Ontology and taxonomy approaches to modeling recommendation scores encapsulates entity relationships. An ontology-based approach could be incorporated into AVRA's RS in addition to the described context-based CBF approach.

Context-aware recommender systems can focus on using the context of the user (as in [33] in which the user profile can guide the recommendation score) and/or the context of the recommendation (as in [34]) to modulate recommendation scores. In this work, context is approached from the perspective of [34], where recommendations can be annotated with additional situation-based information called the context. Rather than interpreting the computer screen in every possible context, this work is about narrowing down the search for onscreen meaning to a select few contexts.

The cold start problem is a situation where the RS has insufficient information about a user to make high-quality recommendations [35]. This quality problem is less pronounced in AVRA as the recommendations are strongly driven by onscreen activity, which leads naturally to an initial set of recommendations that are adjusted over time based upon clicks or recommendation rejections. The review article [35] provides an overview of the RS state of the art. RS recommendation quality is discussed in the literature, and some key ideas are repeated here. Reverb [36] is an IDE plug-in that recommends previously visited web pages related to code being written by a programmer in the IDE. The idea is to reduce false-positive recommendations when offering the programmer a recipe to look up as she is writing a program. This increase in quality is accomplished by only recommending pages that have been previously visited by the user. A predecessor of Reverb is Fishtail [37], which recommends web pages for the same purpose, but without restricting recommendations using the user history as Reverb does. The consequence of recommending pages based only upon keywords is lower quality recommendations [38]. Another interesting example from application-specific RS is [3], an MI system that works in tandem with the programmer to expose programming recipes and other useful programming tips. Whereas [3] [37] [36] and others integrate into specific programs, AVRA was designed to process images of the computer screen as a whole, loosening the integration between the applications and RS.

An RS for personal computers that learns and predicts actions was described in [39]. In that work, a distinction is made between types of information extracted by the RS: action features describe items that happened recently on the computer, while state features describe the current state of the machine. Examples of action features are a history of program calls, the stream of keyboard characters, and onscreen streaming video. Examples of state features are a list of programs that are currently running, the current directory, and the current language settings. In [40], the goal of the RS is prediction of the next command to be typed into a terminal program. Recommendations from this type of RS are based upon previous input (action features), but cannot see the current or past output resulting from the execution of these commands. AVRA seeks out information from many different programs visible onscreen, while command prediction only detects the command history within one program. An unrelated feature in [40] that influenced the design of AVRA is the mapping of recommendations to function keys. In [40] the keyboard keys F1 to F5 were mapped to five different command recommendations presented onscreen to the user. Picking up on this interface design, AVRA uses three onscreen buttons to expose recommendations.

The RS user interface design and behavior is a major driver of user adoption, and is a factor in the user's trust of the RS recommendations [38]. Furthermore, exposing the capability to explore and comprehend recommendations in the GUI can be a negative influence on the user experience [38]. The virtual assistant "Clippy" from Microsoft Word is an example of good RS design combined with bad GUI design [38]. Clippy was considered by many users to be too intrusive, even though the recommendations provided were typically useful [38]. In contrast, less intrusive features such as spellcheck, autocomplete and autocorrect are now widely adopted in many technologies. Two independent factors influencing GUI design in RS are obviousness of the recommendations and cognitive effort required to use the interface [38]. Cognitive effort in AVRA is reduced by limiting the number of buttons in the GUI, restricting the action recommendation type, and restricting the complexity of the action recommendations. The obviousness of the recommendations in the GUI is enhanced by the graphical icons used to symbolize various tasks. For example, a mail icon next to a person's name (e.g., "Daniel") is more obvious than the sentence "Compose an e-mail to Daniel". The former contains only one word (the name), and can be interpreted at a glance without reading into the text of the recommendation with full focus.

AVRA was designed to continuously prompt the user. Similarly, [41] presents a smart home application for assisting the elderly that involves prompting the user many times. To limit the negative effects of repeated prompting, the system learns rules that define when activities normally occur and utilizes these rules to automate prompting. Whereas [41] focuses on waiting to prompt until the notification is needed, AVRA focuses on recommending quickly based upon the current content on the computer screen, or not at all.

B. Mixed Initiative

Iterative sensemaking is a process of working with data sets of semi-reliable structure to produce intermediate results along the way to a conclusion [42] [43]. Each sensemaking iteration involves two steps: foraging and synthesis. These steps are easier to perform quickly with assistance from a predictive intelligent system, such as an MI system. An MI system assists a human analyst to derive and take advantage of insights into data. In MI systems, the breakdown of work between the computer and the human focuses on the strength of each participant in the iterative problem solving activity [44]. The goal of the collaboration between human and virtual agent is producing insights leading to one or more conclusions regarding the data [22]. AVRA's design was based upon recent work that recommends the desired characteristics for MI systems [45] [22].

Graphics-based interaction with the MI system helps the user to more effectively concentrate on decision making [46]. AVRA's 3 button UI design was influenced by similar interface designs, including the "Smart Reply" feature of Google Inbox, which offers up to 3 candidate email message replies as draft

 ISBN: 978-1-61208-577-7

responses [47]. Unlike Smart Reply, AVRA's UI needs to specify multiple items per button: an action, a short snippet of text describing the data used by the action, and a longer tooltip message about the recommended action. The representation of these 3 items per button is accomplished using action icons in the GUI rather than action description text, saving real estate in the GUI that is better utilized by the action data description text. The tooltip hides the longer full-text description of the recommended action. The tooltip text feature allows the user to glean additional insight by reading a longer text explanation for the proposed action recommendation. Having compacted the form factor of the GUI, there is a tendency to increase the number of GUI buttons. However, including too many buttons in the GUI would create a paradox of choice where the interface begins adding cognitive pressure onto the user by forcing her to think about many possible options, rather than achieving the goal this system was designed for, which is to reduce that cognitive pressure [48].

IV. Recommendation Learning

AVRA contains processes for supervised learning of new contexts through CNN training, and for learning new keywords for existing contexts through DNN training. The RS associates each DNN keyword with one or more recommendation actions. AVRA can also learn new information from unsupervised learning of contexts and keywords.

A. Supervised Context Learning

One expects that a CNN recognizing activities on the computer screen requires thousands of classes to represent the wide variety of activities taking place on the computer. Training the CNN through supervised learning to recognize a new context requires many example images. Typically, several hundred representative images of a context are required to train the CNN. To acquire images that look like a particular context (e.g., the Eclipse IDE) an Internet image scraper was used to pull many high-resolution images from online image search engines based upon DNN keywords within the context, and adjusted these keywords manually to improve the quality of the scraped images (e.g., "eclipse IDE java programming"). The scraper was implemented in nodejs using the images-scraper library [49], and filtered results by file type and file name to exclude unhelpful or damaged images. The resulting many thousands of images for each context were reduced with automated duplicate image deletion, followed by further manual inspection. Images were selected that best represent the context to the person selecting the images. A second source of training data was locally generated training images. In order to test out the feasibility of unsupervised learning, AVRA's image capture program was executed for several days in the background as the computer was used for normal work activities, in order to generate sufficient data to browse and extract relevant images. Once the data set of representative images was finalized, the supervised training was ready to proceed.

Cross-validation is the fitness measure used for supervised context learning during CNN training. Transfer learning was used to decrease CNN training time and increase classification accuracy [50]. The transfer learning was implemented using the inception v3 CNN [51], taking advantage of feature detection capabilities of image recognition software trained on large sets of images.

TABLE I. CNN CONFUSION MATRIX FOR K=1% RESULTED IN HIGH RECALL WITH LOW PRECISION.

		Predicted											Recall
		A	B	C	D	E	F	G	H	I	J	K	
Actual	A	5	5	5	0	4	4	0	0	0	0	0	1
	B	2	5	5	2	5	5	0	0	0	0	0	1
	C	0	3	5	0	5	5	0	0	0	0	0	1
	D	0	0	0	5	2	0	1	0	1	0	1	1
	E	0	4	1	2	5	0	0	0	0	0	0	1
	F	0	0	5	5	5	5	0	0	0	0	0	1
	G	0	0	0	0	0	0	5	0	0	0	4	1
	H	0	0	0	0	0	0	0	5	0	1	0	1
	I	0	0	0	0	0	0	3	1	5	4	0	1
	J	0	0	0	0	0	0	2	3	2	5	0	1
	K	0	0	0	0	0	0	3	0	0	0	5	1
Precision		.71	.29	.24	.36	.19	.26	.36	.56	.63	.50	.50	

To provide a motivating example of AVRA in action, the CNN was trained on 9654 images in total, covering 11 different contexts. A graph of the training and cross-validation accuracy at each epoch during the CNN training is presented in Figure 3, produced with inception's retraining code [51]. The retraining configuration included 4000 training steps. Figure 3 shows that validation accuracy lags training accuracy, as expected. The shape of the learning graph is also as expected, with an initial high rate of learning followed by slower incremental improvements in accuracy. The performance of the CNN was measured in terms of recall and precision when trained to identify several contexts. The testing images (5 images per class for 11 classes) were not included in the training dataset. The testing data had 1 class per image. The results are presented in a confusion table (Table I).

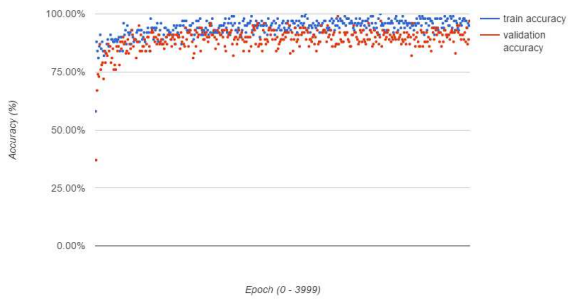

Figure 3. Training and validation classification accuracy during the 4000 epochs of the CNN training process. The final test accuracy was 92.20%

The testing dataset used to produce the confusion matrix of Table I only contained images with one class per image. Hyperparameter K of the CNN is a classification threshold that tunes AVRA to be more conservative (higher threshold) or more open to evaluating hypotheses (lower threshold). The recall observed in Table I is high at the expense of precision. There is a fundamental trade-off between recall and precision determined by the threshold K. This is observed in Table II, where K=95%. With such a high requirement for certainty that a context has been detected, there is a higher precision, but only when the context is detected. The cost of increasing K is missing the context completely, costing the algorithm on both the precision and the recall metrics. Because false positive context detection is generally less damaging to the recommendation quality than false negative context detection, AVRA is configured with a low K threshold as in Table I, rather than a high value as in Table II.

 ISBN: 978-1-61208-577-7

TABLE II. CNN CONFUSION MATRIX FOR K=95% RESULTED IN LOW RECALL WITH HIGH PRECISION.

	Predicted											Recall
		A	B	C	D	E	F	G	H	I	J	K	
Actual	A	0	0	0	0	0	0	0	0	0	0	0	0
	B	0	0	0	0	0	0	0	0	0	0	0	0
	C	0	0	0	0	0	0	0	0	0	0	0	0
	D	0	0	0	3	0	0	0	0	0	0	0	0.6
	E	0	0	0	0	1	0	0	0	0	0	0	0.2
	F	0	0	0	0	0	0	0	0	0	0	0	0
	G	0	0	0	0	0	0	2	0	0	0	0	0.4
	H	0	0	0	0	0	0	0	5	0	0	0	1
	I	0	0	0	0	0	0	0	0	1	0	0	0.2
	J	0	0	0	0	0	0	0	0	0	2	0	0.4
	K	0	0	0	0	0	0	0	0	0	0	3	0.6
Precision		0	0	0	1	1	0	1	1	1	1	1	

B. Supervised Keyword Learning

The DNN for each context was trained to recognize keywords within filtered OCR output text. The purpose of the DNN is to detect keywords in the presence of OCR transcription errors. The classification task is accomplished through a process of learning the mistakes made by the OCR system. More specifically, AVRA generates test images for each keyword, feeding the images through the OCR software to obtain string data. Next, AVRA encodes the error-laden OCR string output as binary ASCII characters into numerical vectors, which include letter frequency information in the encoding scheme [23]. The input vectors for DNN training were produced using this approach, while the index of each keyword within the RS is the output vector corresponding to a particular recommendation.

C. Unsupervised Keyword and Context Learning

Generating recommendations from unsupervised learning involves learning CNN contexts as well as DNN keyword sets. AVRA's unsupervised learning function is to search for an onscreen keyword that the user searched for in the past (verbatim) after seeing it on the screen. This approach will be discussed in upcoming work.

V. Performance Evaluation

Execution time for the OpenMP and multicore approaches are reported here. Next, GP-GPU acceleration with CUDA is presented. Finally, recall, precision, precision-recall and ROC curves are presented.

1) OpenMP and Multicore DNN Training Acceleration: Each DNN in AVRA was trained for 150 epochs, and each epoch required approximately 250 seconds to complete on a low cost VM. This execution time is related to the size of the training and testing sets, and so these figures are different for each DNN. Assuming 50 DNNs with 150 epochs each, and an average epoch execution time of 250 seconds, the training time on a single VM would be: (50 DNNs) x (150 epochs) x (250 seconds) which is approximately 520 hours. This slow training and classification did not provide a reasonable response time at scale. Even if every DNN is trained on a different dedicated VM, the execution time becomes (150 epochs) x (250 seconds) which is approximately 10.5 hours. Reducing the number of epochs would reduce the accuracy of the DNN classification. This leaves the epoch execution time as the variable to be reduced.

OpenMP is a shared memory parallel programming API suitable for parallelizing applications on a multiprocessor computer [52]. Theano is compatible with OpenMP and ships with a benchmark for testing the speedup provided by OpenMP [53]. This benchmark was used to generate the data in Table III which is a generic view of accelerating theano with OpenMP.

TABLE III. THEANO'S OPENMP BENCHMARK, TIMED WITH A VECTOR OF 200,000 ELEMENTS AND 4GB RAM

OpenMP Threads	CPU Cores	Operation Type	Without OpenMP (s)	With OpenMP (s)	Speedup
1	2	Fast	0.000113	0.000099	1.14
2	2	Fast	0.000110	0.000066	1.67
3	2	Fast	0.000114	0.000137	0.83
4	2	Fast	0.000111	0.000128	0.87
1	2	Slow	0.006590	0.006091	1.08
2	2	Slow	0.006208	0.003060	2.03
3	2	Slow	0.006107	0.004127	1.48
4	2	Slow	0.006253	0.003738	1.67

TABLE IV. DNN TRAINING TIME PER EPOCH. THE AVERAGE BASELINE EXECUTION TIME WAS 257.

OpenMP Threads	CPU Cores	RAM (GB)	Epoch	Training Time (s)	Speedup from Average Baseline
1	2	4	0	263	N/A
1	2	4	1	251	N/A
2	2	4	0	196	1.3
2	2	4	1	220	1.2
3	2	4	0	151	1.7
3	2	4	1	173	1.5

The results from theano's benchmark indicate that configuring theano with 2 OpenMP threads is the best option for reducing epoch execution time. Perhaps this result is a product of the VM having 2 CPUs and OpenMP mapping one OpenMP thread to each CPU. However, testing with AVRA's DNN code reported in Table IV indicates that 3 OpenMP threads is best. This disagreement between the generic benchmark and the AVRA code itself is a reminder that often benchmarks are a poor proxy for testing performance improvements in computer architecture [54]. Note that even though a 2x speedup was achieved by using OpenMP, this only reduced the per-DNN execution time from 10.5 hours down to 5.25 hours ($> 150s$ / training epoch), which is still impractically slow. In the next section GP-GPU acceleration is leveraged to further reduce the per-epoch execution time of DNN training.

2) DNN Training Acceleration using GPU and CUDA: An AWS instance of type g2.2xlarge with a GRID K520 GPU was configured with an existing AMI containing theano for cuda 6.5 [55]. The system was tested with the same DNN code and data used to produce the data in Table IV, although OpenMP was not enabled.

CUDA is a parallel computing programming model which enables GPU hardware acceleration of computations [56]. The per-epoch execution time was reduced from 151 seconds without the using CUDA and the GPU, to 6.5 seconds when the GPU is used. This represents a 23x speedup relative to the best OpenMP result in Table IV. With a 6.5 second epoch execution time, training a DNN can take (6.5 seconds) x (150 epochs) which is approximately 16 minutes and 15 seconds. Note that the epoch execution time is highly variable based upon the size of the training data, and the number of classes trained into the DNN. With OpenMP, multicore, and GPU acceleration, the time required to interpret the computer screen with AVRA did not scale as the number of DNNs increased. With one DNN the latency from changes on the screen to updates in the GUI was approximately 40 seconds. Execution time scaled linearly so that 10 DNNs required 400s to analyze the screen and update the GUI. This challenge led to a change in approach, leaving

 ISBN: 978-1-61208-577-7

behind the slow but precise DNNs as future work, and instead using the text filter as the main text classifier in AVRA.

3) Execution Time Measurement for AVRA: The design is shown in Figure 4, removing the DNNs and connecting the output of the text filter to the RS. The image capturing and OCR in AVRA were modified as well. Instead of capturing the OCR of the whole screen, the fullscreen image was processed into text in slices, with each slice processed in a separate thread. The advantage of this approach was much faster OCR, but the downside was that this approach would miss text sliced at the lines between slices where the image was cut. To counter this, a second set of image slices offset by half of the slice height was also processed by OCR as well. This ensured that no onscreen text was missed by the OCR process. Figure 5 shows how a screen can be cut into $2n-1$ slices. Each thread is responsible for extracting text from its image slice and uploading the text to the server. Testing various settings for n yielded that $n=6$ had the lowest execution time on average for a computer screen of size $4800x3600$ pixels using an Intel Core i7 3.6GHz CPU, 16GB DDR3 RAM, an SSD hard disk, and 2 GeForce GTX 770 GPUs. For 5 trials with fullscreen color images, $n=6$ (11 image slices) led to an OCR execution time for a given slice of 4.4 ± 1.5 seconds per slice, and an overall OCR execution time of 8.5 ± 1.4 seconds per image. For 5 trials with fullscreen grayscale images, $n=6$ (11 image slices) led to an execution time of 2.9 ± 0.5 seconds per slice, and an overall execution time of 5.4 ± 0.9 seconds per image. The advantage of using color images was increased CNN precision and recall. To balance the response time and scalability with high recall, AVRA was set to process color images to maintain the recall level at the expense of execution time.

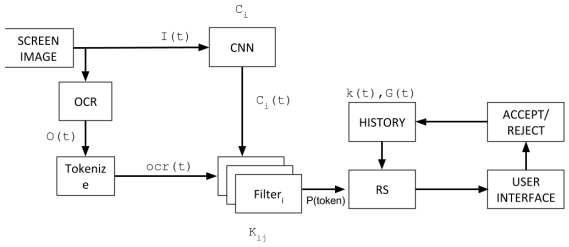

Figure 4. AVRA System Overview, after removing DNN to reduce execution time.

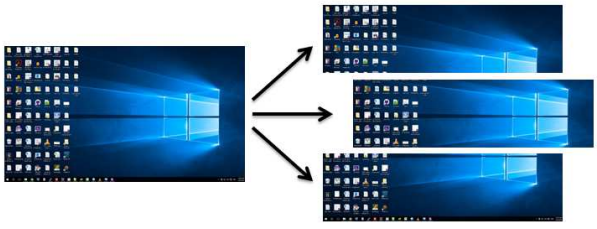

Figure 5. OCR capture process, splitting fulscreen images into $2n-1$ slices for executing overlapping subtasks in parallel threads to reduce OCR execution time.

Consider an example where AVRA is trained to recognize 11 contexts containing 2,103 $keywords$ overall. Tracking the flow of information through the color image processing design described above, and rounding to the nearest second, an image captured at time 0 passed enough information for the CNN to complete context recognition after 4 seconds, and the first context-filtered OCR results emerge from the text filter one second later. The RS results were available 4 seconds later, with another 2 seconds required to update the GUI. The minimum time between information appearing onscreen and a recommendation displaying on the GUI was therefore 11 seconds.

Improving on this design, the RS and filter were moved into the same file, obviating the database communication between these two modules. Tracking the flow of information through the new design and again rounding to the nearest second, an image captured at time 0 passed enough information for the CNN to complete context recognition after 4 seconds, and the first context-filtered OCR results emerged from the text filter combined with the RS after 2 seconds, with an average of less than one second required to update the GUI. The minimum time between information appearing onscreen and a recommendation displaying on the GUI was therefore 6 seconds.

The end-to-end execution time from detection onscreen to recommendation in a button was measured. To collect data from AVRA, the search text "looking for restaurant English text" was inserted into the Google Images API along with the restriction that the dimensions of the image results be exactly $1080x1920$ pixels, corresponding to the dimensions of the mobile phone used for testing. The first 50 results were saved as a dataset of images referred to as $SMALL\_IMAGES$.

To simulate the latency of image capture, a region of the desktop $1080x1920$ pixels was captured into a file for each processed image. After this simulated image capture delay, the CNN and OCR processes of AVRA were passed one of the static images from $SMALL\_IMAGES$. The total execution time required to fully process all 50 images of $SMALL\_IMAGES$ through the OCR, CNN, text filter, RS, and GUI was 176.0 seconds. The average execution time per image was 3.52 ± 1.51 seconds.

To ensure that AVRA can execute relatively quickly when many contexts have been trained into the CNN, ten latency samples were recorded for AVRA CNNs trained with 5, 50, 100, 200, and 400 contexts. Each sample was obtained by recording the CNN output after processing an image with dimensions $4800x3600$ pixels using a the same Intel Core i7-based system described above. The results, shown in Figure 6, reveal that the execution time grows with the number of added contexts. However, the incremental cost of adding contexts decreases with the number of contexts added, as shown in Figure 7. Furthermore, the execution time at 400 contexts remained low, at 3.97 ± 0.10 seconds. In Figure 6, the vertical axis shows the execution time required to process one $4800x3600$ pixel image using AVRA's CNN. Results for 5, 50, 100, 200, and 400 contexts are shown with error bars indicating the standard deviation for each measurement. A moving average line is added to the figure, revealing with a decreasing slope that the latency cost of adding more contexts is decreasing. In Figure 7, the vertical axis shows the execution time required to process one image, divided by the number of contexts trained into the CNN. Results for 5, 50, 100, 200, and 400 contexts are shown.

4) Recall, Precision, Precision-Recall and ROC Curve: It is interesting to observe the capabilities of AVRA regarding overlapping images because a program like AVRA must recognize onscreen items such as program windows that may overlap

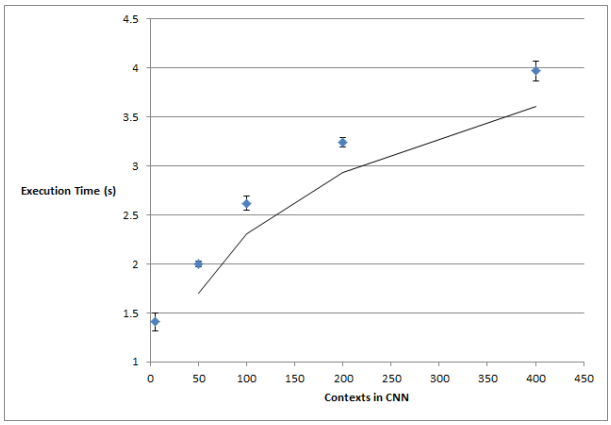

Figure 6. CNN latency as number of context increases.

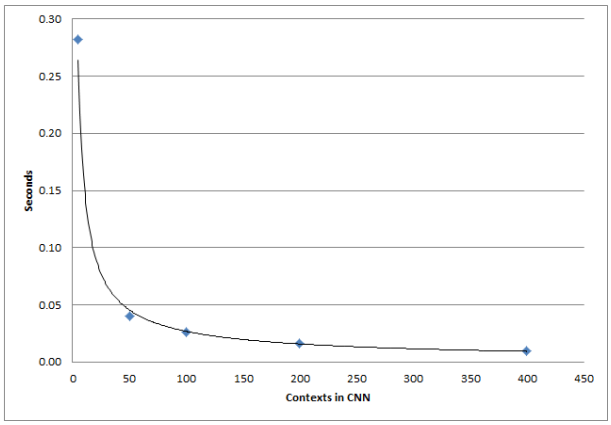

Figure 7. CNN latency per context.

when observed. AVRA recognizes multiple items in one image. For example, as shown in Figure 8, AVRA analyzing an image containing both a terminal window and an Eclipse IDE window leads to the recognition of both by the CNN. AVRA sometimes recognizes both contexts when they are partially occluded, but when overlap is high the occluded context (eclipse) was not recognized. However, in some cases the occluded context (in this case a terminal window) was recognized by the CNN. When two windows appeared side by side, the CNN usually recognized the context associated with each one.

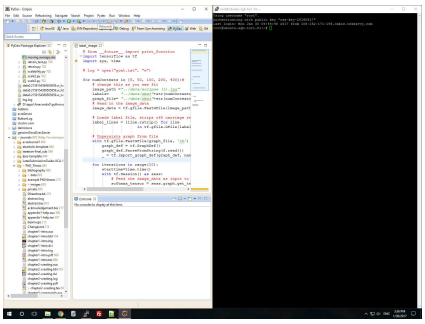

Figure 8. AVRA's CNN recognizing content in side by side windows. CNN output score was: eclipse (79.64%), console (12.73%), facebook (3.74%).

To observe the precision and recall for AVRA, it was tested by displaying to the system images on the full screen area one at a time. There were 50 $4800x3600$ pixel fullscreen desktop images, which contained one of two contexts (eclipse or console) and one specific $keyword$ known to the CNN for that context. The images were collected from real examples, and

TABLE V. CONFUSION MATRIX FOR ASSESSING AVRA.

		Predicted		Recall
		$C_A \wedge K_B$	$\neg C_A \vee \neg K_B$	
Actual	$C_A \wedge K_B$	36	14	0.7200
	$\neg C_A \vee \neg K_B$	0	50	
Precision		1.0000		

so they sometimes contained several other $keywords$ trained into AVRA for the context in question. For the eclipse context, the $keyword$ was `import`, and for the console context the $keyword$ was `apt-get`. Note that in eclipse the text is blue on a white background, and in a terminal the text is white on a black background. Neither of these scenarios is the typical OCR situation of black text on a white background. These examples were selected because they are a more realistic sample of text on a desktop computer than the obvious case of black text on a white background. To test for TN results 50 additional images containing no relevant context or keyword in them were also presented to AVRA one at a time. These 50 $4800x3600$ images were collected from Google Images using the keywords "my pictures" and the photo type filter was set to "photo". No FP examples were recorded for AVRA during the experiment, as the chance that a context and keyword are incorrectly selected by AVRA at the same time is very small.

The confusion matrix for the recorded observations is presented in Table V, after AVRA was trained on 11 classes. Samples either contained context C_A and keyword K_B, or they did not ($\neg C_A \vee \neg K_B$). The testing data had 25 images per class, with 1 class per testing image, one relevant $keyword$ per image, and classification threshold K=1% To generate an ROC curve for the collected AVRA data, a binary classification measure was used. Any sample with a TP results was associated with the ground truth state $[1, 0]$, and any sample with no TP results was associated with the ground truth state $[0, 1]$. The classifier guess was set to $[1, 0]$ in cases where TP or FP was observed. Otherwise the classifier guess for the sample was set to the state $[0, 1]$. The ground truth and classifier guesses were used to create the ROC curve of Figure 9, and the Precision-Recall curve of Figure 10. In 5 of the 50 images containing recommendable information, the recommendation was picked up by the text filter but was eliminated by the RS and 3 or more items the RS could recommend ranked higher than the $keyword$ of interest.

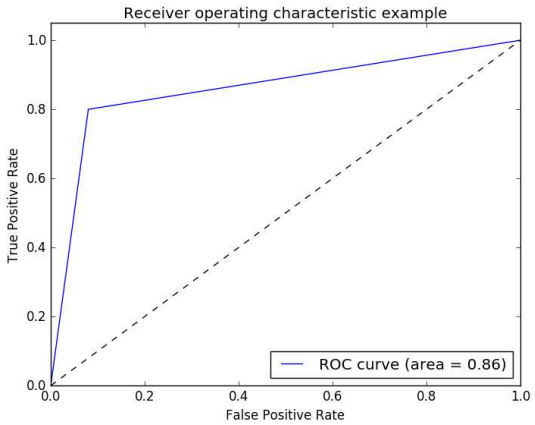

Figure 9. ROC curve for AVRA binary classification experiment

The following is an example of the weaknesses of AVRA's

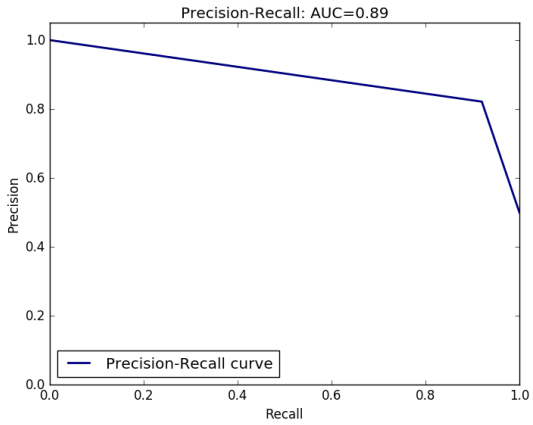

Figure 10. Precision-Recall curve for AVRA binary classification experiment

strict unsupervised learning approach described above. Due to the strict nature of the learning algorithm, when AVRA observes the user subtracting numbers in a calculator, the exact sequence of operations (e.g., $25,000 * 4$) is stored as a pattern to be recommended later on, rather than being learned symbolically as the multiplication of two numbers. Another limitation of the current approach to unsupervised learning is the keyword lexicon. At this time, only keywords already present in the word embedding model can be added as keywords in one of AVRA's DNNs. AVRA sometimes misses the context or keyword information, or has higher confidence in unhelpful recommendations than detected helpful recommendations. Two overall challenges in developing AVRA were poor classification of keywords with very short text length (e.g., the terminal command "ls"), and low context detection confidence (e.g., 2% confidence in the correct class). These cases were rare but noticeable. Perhaps the short keyword recognition could be resolved by modifying the DNN input filter hyperparameters. The low confidence context detection cases may be mitigated by collecting additional image data for context training.

VI. Conclusion

This work presented the overall design of AVRA, a virtual assistant for personal computer users. Measures of performance and measures of effectiveness were defined, and the design of AVRA's Mixed Initiative qualities was discussed. The performance of AVRA and a similar closed source commercial tool were recorded, and these two tools were compared in the context of the aforementioned requirements. The thesis statement claimed that a deep learning artificial intelligence can provide action recommendations related to onscreen messages. This work explained at a high level how action recommendations can be provided within a reasonable response time, and how these recommendations can be acted upon with a single mouse click.

An architecture for a deep learning recommender system for personal computer users was described in this work. Action recommendations produced by this design are personalized to the user and are generated in real-time. The AVRA system mines information from screen capture data, rather than interface with individual applications. Recommendations are presented to the user in an intuitive button-based user interface. The architecture described in this work can provide the foundation for further research into recommender system for personal computer users.

Future work planned for AVRA includes collaborative filtering and related privacy considerations, the expansion of AVRA's input processing and modeling capabilities, and unsupervised learning. Applying content-based image recognition and semantic segmentation of images for face and object classification within a context (and generating related recommendations) is an interesting area to explore as well.

References

[1] Amazon.com, Inc., "Alexa," https://developer.amazon.com/public/solutions/alexa, [retrieved: 2016-07].

[2] IBM, "IBM Watson Developer Community," https://developer.ibm.com/watson/, [retrieved: 2016-07].

[3] A. Smith, "Kite - Programming copilot," https://kite.com/, 2016, [retrieved:2016-05].

[4] Apple Inc., "iOS - Siri - Apple (CA)," http://www.apple.com/ca/ios/siri/, [retrieved: 2016-07].

[5] D. Kittlaus, "The team behind Siri debuts its next-gen AI Viv at Disrupt NY 2016," https://www.youtube.com/watch?v=MI07aeZqeco, 2016, [retrieved: 2016-07].

[6] V. Labs, "Viv," http://viv.ai/, [retrieved: 2016-07].

[7] SoundHound Inc., "SoundHound - Hound App," http://www.soundhound.com/hound, [retrieved: 2016-07].

[8] API.ai, "Assistant by Api.ai - Assistant.ai," https://assistant.ai/, [retrieved: 2016-07].

[9] N. Satt, "Hands-on with facebook m: the virtual assistant with a (real) human touch," http://www.theverge.com/2015/10/26/9605526/facebook-m-hands-on-personal-assistant-ai, 2015, [retrieved: 2016-07].

[10] J. Hempel, "Facebook launches m, its bold answer to siri and cortana," http://www.wired.com/2015/08/facebook-launches-m-new-kind-virtual-assistant/, 2015, [retrieved: 2016-07].

[11] K. Bell, "Facebook messenger chief: It will be years before everyone has m," http://mashable.com/2016/04/18/facebook-m-not-ready-for-years/#rnO6Nl3m1sqg, 2016, [retrieved: 2016-07].

[12] Y. LeCun, "EmTech MIT 2015: Augmented Knowledge Teaching Machines to Understand Us," https://youtu.be/a3DzL7Ad4PU?t=17m10s, 2015, [retrieved: 2016-07].

[13] Google, "Google Developers - Google Now," https://developers.google.com/schemas/now/cards, [retrieved: 2016-07].

[14] Q. Guo and Y. Song, "Large-scale analysis of viewing behavior: Towards measuring satisfaction with mobile proactive systems," in CIKM 2016, 2016.

[15] Microsoft, "Cortana - Meet your personal assistant - Microsoft - Global," https://www.microsoft.com/en/mobile/experiences/cortana/#, [retrieved: 2016-07].

[16] NextOS, "NextOS - Denise - Creating Virtual Life," http://www.nextos.com/, [retrieved: 2016-07].

[17] A. M. Elkahky, Y. Song, and X. He, "A multi-view deep learning approach for cross domain user modeling in recommendation systems," in Proceedings of the 24th International Conference on World Wide Web. ACM, 2015, pp. 278–288.

[18] M. Lynlet, "Google unveils google assistant, a virtual assistant thats a big upgrade to google now," https://techcrunch.com/2016/05/18/google-unveils-google-assistant-a-big-upgrade-to-google-now/, 2016, [retrieved: 2016-07].

[19] J. Karlin, "Launchy," https://www.launchy.net/, 2016, [retrieved:2017-05].

[20] Google Inc, "Google Chrome," https://www.google.com/chrome/, 2016, [retrieved:2017-05].

[21] C. D. Wickens, H. Li, A. Santamaria, A. Sebok, and N. B. Sarter, "Stages and levels of automation: An integrated meta-analysis," in Proceedings of the Human Factors and Ergonomics Society Annual Meeting, vol. 54. SAGE Publications, 2010, pp. 389–393.

 ISBN: 978-1-61208-577-7

[22] S. Makonin, D. McVeigh, W. Stuerzlinger, K. Tran, and F. Popowich, "Mixed-initiative for big data: The intersection of human+ visual analytics+ prediction," in 2016 49th Hawaii International Conference on System Sciences (HICSS). IEEE, 2016, pp. 1427–1436.

[23] D. Shapiro, N. Japkowicz, M. Lemay, and M. Bolic, "Fuzzy string matching with character-level deep neural network classification," Applied Artificial Intelligence, submitted 2016.

[24] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski et al., "Human-level control through deep reinforcement learning," Nature, vol. 518, no. 7540, 2015, pp. 529–533.

[25] K. C.-C. Chang, B. He, and Z. Zhang, "Metaquerier over the deep web: Shallow integration across holistic sources," in In Proceedings of the VLDB Workshop on Information Integration on the Web, 2004.

[26] M. Balabanović and Y. Shoham, "Fab: content-based, collaborative recommendation," Communications of the ACM, vol. 40, no. 3, 1997, pp. 66–72.

[27] M. J. Pazzani, "A framework for collaborative, content-based and demographic filtering," Artificial Intelligence Review, vol. 13, no. 5-6, 1999, pp. 393–408.

[28] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen, Collaborative Filtering Recommender Systems. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 291–324. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-72079-9_9

[29] J. Gemmell, T. Schimoler, M. Ramezani, L. Christiansen, and B. Mobasher, "Resource recommendation for social tagging: a multi-channel hybrid approach," Recommender Systems & the Social Web, Barcelona, Spain, 2010.

[30] S. Maneeroj and A. Takasu, "Hybrid recommender system using latent features," in Advanced Information Networking and Applications Workshops, 2009. WAINA'09. International Conference on. IEEE, 2009, pp. 661–666.

[31] L. T. Weng, Y. Xu, Y. Li, and R. Nayak, "Exploiting item taxonomy for solving cold-start problem in recommendation making," in 2008 20th IEEE International Conference on Tools with Artificial Intelligence, vol. 2, Nov 2008, pp. 113–120.

[32] L. Martinez, L. G. Perez, and M. J. Barranco, "Incomplete preference relations to smooth out the cold-start in collaborative recommender systems," in Fuzzy Information Processing Society, 2009. NAFIPS 2009. Annual Meeting of the North American. IEEE, 2009, pp. 1–6.

[33] K. Verbert, N. Manouselis, X. Ochoa, M. Wolpers, H. Drachsler, I. Bosnic, and E. Duval, "Context-aware recommender systems for learning: a survey and future challenges," IEEE Transactions on Learning Technologies, vol. 5, no. 4, 2012, pp. 318–335.

[34] G. Adomavicius and A. Tuzhilin, "Context-aware recommender systems," in Recommender systems handbook. Springer, 2011, pp. 217–253.

[35] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez, "Recommender systems survey," Knowledge-Based Systems, vol. 46, 2013, pp. 109–132.

[36] N. Sawadsky, G. C. Murphy, and R. Jiresal, "Reverb: Recommending code-related web pages," in Proceedings of the 2013 International Conference on Software Engineering. IEEE Press, 2013, pp. 812–821.

[37] N. Sawadsky and G. C. Murphy, "Fishtail: from task context to source code examples," in Proceedings of the 1st Workshop on Developing Tools as Plug-ins. ACM, 2011, pp. 48–51.

[38] E. Murphy-Hill and G. C. Murphy, "Recommendation delivery," in Recommendation Systems in Software Engineering. Springer, 2014, pp. 223–242.

[39] O. Madani, H. H. Bui, and E. Yeh, "Efficient online learning and prediction of users' desktop actions." in IJCAI, 2009, pp. 1457–1462.

[40] B. Korvemaker and R. Greiner, "Predicting unix command lines: adjusting to user patterns," in AAAI/IAAI, 2000, pp. 230–235.

[41] L. B. Holder and D. J. Cook, "Automated activity-aware prompting for activity initiation," Gerontechnology: international journal on the fundamental aspects of technology to serve the ageing society, vol. 11, no. 4, 2013, p. 534.

[42] Y.-a. Kang and J. Stasko, "Characterizing the intelligence analysis process: Informing visual analytics design through a longitudinal field study," in Visual Analytics Science and Technology (VAST), 2011 IEEE Conference on. IEEE, 2011, pp. 21–30.

[43] P. Zhang and D. Soergel, "Towards a comprehensive model of the cognitive process and mechanisms of individual sensemaking," Journal of the Association for Information Science and Technology, vol. 65, no. 9, 2014, pp. 1733–1756.

[44] J. Allen, C. I. Guinn, and E. Horvtz, "Mixed-initiative interaction," IEEE Intelligent Systems and their Applications, vol. 14, no. 5, 1999, pp. 14–23.

[45] K. Cook, N. Cramer, D. Israel, M. Wolverton, J. Bruce, R. Burtner, and A. Endert, "Mixed-initiative visual analytics using task-driven recommendations," in Visual Analytics Science and Technology (VAST), 2015 IEEE Conference on. IEEE, 2015, pp. 9–16.

[46] P. Pu and D. Lalanne, "Design visual thinking tools for mixed initiative systems," in Proceedings of the 7th international conference on Intelligent user interfaces. ACM, 2002, pp. 119–126.

[47] G. Corrado, "Computer, respond to this email." https://research.googleblog.com/2015/11/computer-respond-to-this-email.html, November 2015, [retrieved: 2017-05].

[48] B. Schwartz, The paradox of choice: why more is less. New York: Ecco, 2004.

[49] Peter Evers, "images-scraper," https://github.com/pevers/images-scraper, 2016, [retrieved: 2016-07].

[50] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" CoRR, vol. abs/1411.1792, 2014. [Online]. Available: http://arxiv.org/abs/1411.1792

[51] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 1–9.

[52] B. Chapman, G. Jost, and R. Van Der Pas, Using OpenMP: portable shared memory parallel programming. MIT press, 2008, vol. 10.

[53] LISA lab, "Multi cores support in Theano," http://deeplearning.net/software/theano/tutorial/multi_cores.html, [retrieved: 2016-07].

[54] J. L. Hennessy and D. A. Patterson, Computer architecture: a quantitative approach. Elsevier, 2011.

[55] M. Beissinger, "How to install Theano on Amazon EC2 GPU instances for deep learning," http://markus.com/install-theano-on-aws/, 2015, [retrieved: 2016-07].

[56] NVIDIA Corporation, "CUDA FAQ — NVIDIA Developer," https://developer.nvidia.com/cuda-faq, [retrieved: 2016-07].

 ISBN: 978-1-61208-577-7