DeepBoard: A Smart Touch Keyboard with Private Prediction of User Photos using Deep Neural Networks

Richard Sicoli
Stony Brook University
richard.sicoli@stonybrook.edu

ABSTRACT
Most mobile operating systems and user applications support the ability to insert user photos from the device directly into the given document or message. The problem is that this often requires the user to perform several steps to insert the intended photo. DeepBoard solves this problem by automatically predicting which photo the user intends on inserting based on the semantics of the given document or message. This allows the user to insert a photo significantly faster compared to the default interface found on most mobile applications. Furthermore, DeepBoard runs locally on the device, which means a user does not need to sacrifice any privacy as there is no dependence on an internet connection.

Author Keywords
Smart touch keyboard; photo prediction; deep convolutional neural networks; privacy;

INTRODUCTION
Inserting photos into a message or document is a common task that users perform on their mobile device frequently. Yet users still need to perform multiple steps in order to complete this task, such as traversing through interface layers and searching through sometimes thousands of photos to find the intended photo. Furthermore, each application is usually delegated with handling its own photo insertion interface, which results in an inconsistent user experience across the device. And while there has been some related work on implementing keyboards with basic web-search support [2], none pertain to the problem of inserting user photos.

DeepBoard attempts to solve these issues by automatically predicting which photo or photos the user intends to insert based on the semantics of the given text message or document. The user can then select the desired photo through the use of a dynamic interface presented directly on the keyboard. This allows the user to type a message and insert a photo without ever needing to search for the photo manually. And since this interaction is tied directly to the keyboard, this action can be performed any time the user is typing across the entire operating system.

Several models are used to interpret the user's text and predict the intended photo or photos. DeepBoard takes advantage of the success of deep convolutional neural networks [6] to classify photos into categories and link the categories to the text semantics through various natural language processing (NLP) techniques. Furthermore, DeepBoard handles all processing completely local to the device, which means no external server is required for DeepBoard's functionality. This provides the added benefit of user privacy and for use without an internet connection. In the following sections, we will look at the exact details of the design as well as an early evaluation of the system.

RELATED WORK
Surprisingly, little to no work (to the best of our knowledge) has been done to develop a keyboard interface for user photo prediction. There has been previous work on text prediction and even emoji prediction [2], but none involving the prediction of user photos. Similar work has been done for photo search, where a user enters a query to search their photos in a photo application such as iOS Photos, or Google Photos. However, this is not a photo prediction based on the user's message or document, instead the user still needs to manually enter a query for the intended photo. Furthermore, this has not been done within the context of smart touch keyboard interfaces.

There exist smart touch keyboards such as Gboard which feature an integrated web-search directly into the keyboard [2]. An example of this is shown in figure 2. This allows a user to perform a web-search within the keyboard for a particular photo. However, as discussed previously, this is not a photo prediction based on the user's message or document, but rather a manual web query. Additionally, this does not search the user's photos, but instead only searches photos on the web.

DEEPBOARD
DeepBoard is a smart touch keyboard that is designed to solve the problems with photo insertion. The following three design principles are what make DeepBoard a novel system for this task.

- Predicts the photo a user intends on inserting based on the semantics of the message or document. No manual search or additional steps are needed.
- Provides a dynamic interface for smart touch keyboards for quickly selecting the intended photo.
- Works entirely local to the device, which means no privacy concerns and no internet connection required.

Photo prediction and insertion

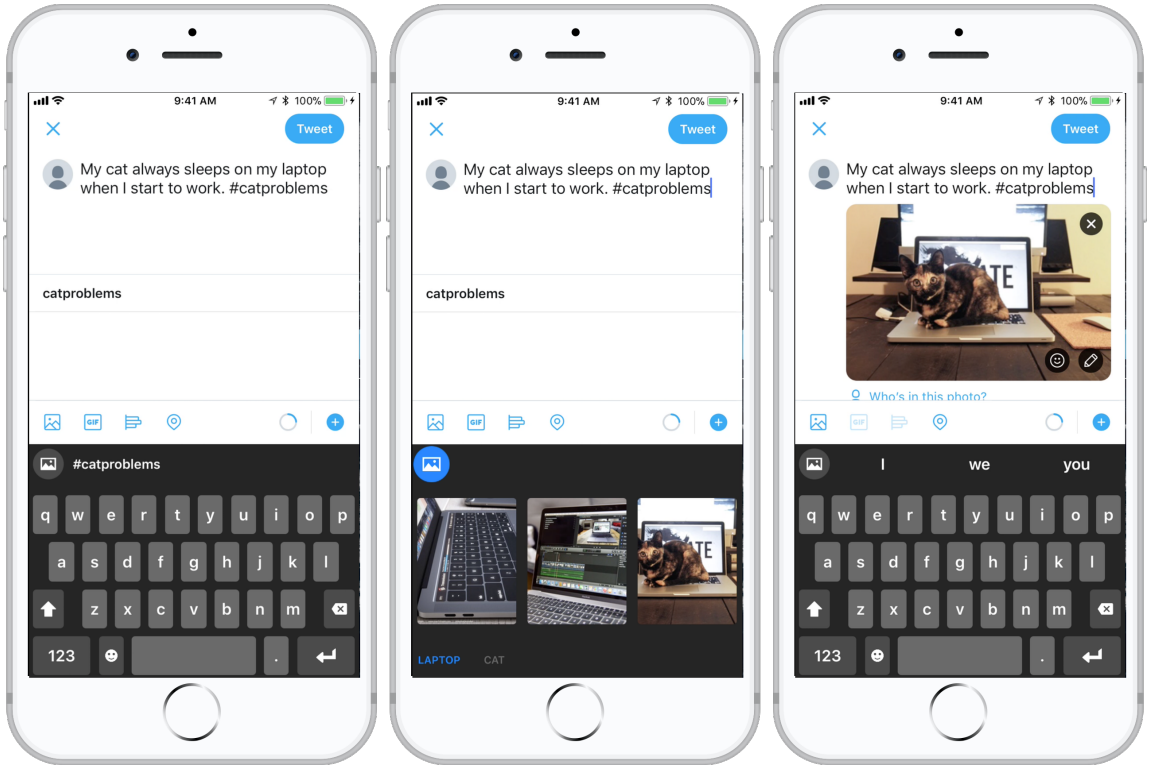

Figure 1. The three step process to use DeepBoard. From left to right, first the user enters a message. Then they press the "Predict Photos Button" in the upper left corner. This will return a list of predicted photos based on the context of the message. In this case, photos for the words "Laptop" and "Cat" were returned. The user can choose to tap on a word to switch between photo categories. Lastly the user taps on the intended photo and inserts it.

Photo Prediction and insertion is performed by the user in three steps. Figure 1 shows an example of this three step process.

1. A user first types some text into a rich text view. This could be an email, message, document etc.
2. Then the user presses the "Predict Photos" button found in the top left corner of figure 1. This will present the user with an interface directly within the keyboard (as shown in figure 1) for selecting the desired photo. DeepBoard will show the photos for each dominant words it has detected. The user can choose to tap a word to reveal the corresponding photos.
3. When the user is ready, they simply select and insert the desired photo directly into the message or document.

Note that photos may be tagged as multiple words if the photo can be classified into multiple categories. This is especially useful for users to find their intended photo regardless of the words they use to describe it.

For example, in figure 1, the message is classified under both "laptop" and "cat". The user could choose either one to reach their desired photo. Also note that words which there are no photos for are not displayed to reduce interface clutter.

Indexing

It should be noted that, before DeepBoard can begin to predict photos, it first needs to process and index every photo on the device. This allows DeepBoard to quickly process all future predictions which is essential when developing a responsive keyboard. The indexing stage may take a few seconds to a minute depending on how many photos the user has and the speed of the device processor. The indexing stage only needs to be completed once, and then again for each new photo the user adds to their library. However, the time needed for indexing new photos is virtually instantaneous.

It should also be noted that the user can still interact with DeepBoard as the photos are indexed as this process happens asynchronously. If the user issues a photo prediction command while the system is indexing, only the currently indexed photos will appear. The user can see if DeepBoard is currently indexing by the progress bar at the bottom of the keyboard, as shown in figure 3. Furthermore, during testing, we found that no noticeable loss of responsiveness occurred during the indexing phase.

IMPLEMENTATION AND DESIGN

Keyboard layout

DeepBoard was developed for all supported iOS devices and works on all sizes and orientations. No feature is dependent on iOS which makes an Android version entirely possible as well. DeepBoard was developed as a standard QWERTY keyboard with several baseline "smart" features that users expect from a smart touch keyboard, such as auto-correct and predictive text. The auto-correct and predictive text are based on

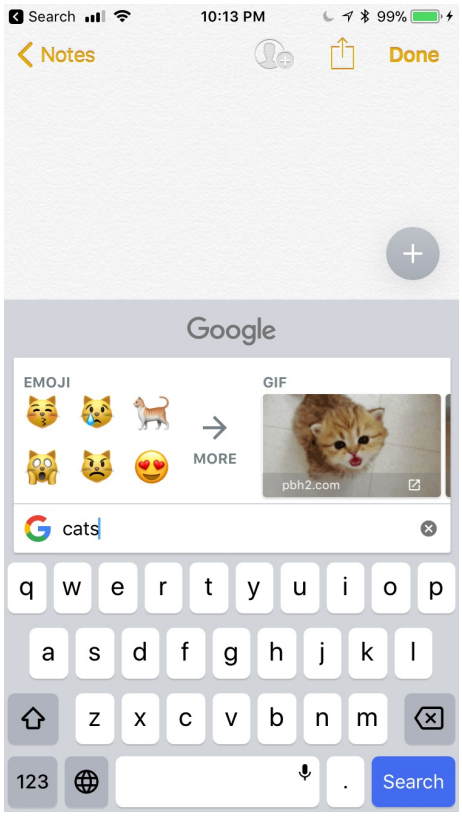

Figure 2. Screenshot of the web search feature of Gboard.

edit distance and augmented with an n-gram language model trained on a Twitter corpus. While the primary focus was on the development of the photo prediction, it was important to develop a sufficient baseline keyboard for users to interact with during the evaluation phase.

Photo retrieval

DeepBoard uses a photo identifier for every photo for efficient retrieval of user photos. This means DeepBoard itself does not need to store any copies of user photos but instead an identifier used to fetch photos whenever necessary.

This is essential to save both storage space as well as memory as photos can be efficiently fetched from the device using system APIs. Photos are fetched during the indexing phase when the photos propagate through the deep learning model and during the prediction phase where photos are fetched based on the semantic analysis of the text.

The implementations of the deep learning model and the semantic analysis are detailed in the next sections.

Deep convolutional neural networks

DeepBoard takes advantage of deep convolutional neural networks for photo classification. Convolutional neural networks have seen great success in image classification tasks and are state of the art for this task [6]. DeepBoard uses a pre-trained model, MobileNets, for classifying the user's photos into 1000 categories. MobileNets was selected for this task because it is specifically developed for mobile and embedded

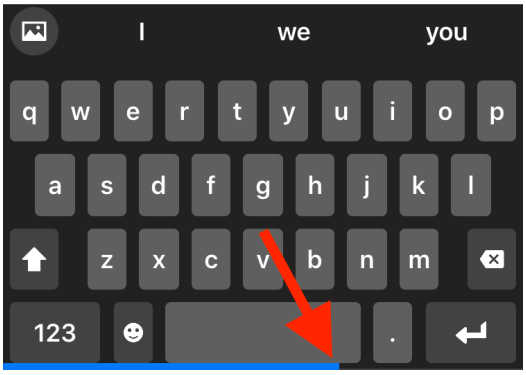

Figure 3. The DeepBoard indexing phase.

devices [5]. MobileNets features a low latency and relatively small model size which is essential when running a responsive keyboard service on a mobile device.

Figure 4 shows the architecture of the MobileNets network. The input to the model is a multi-channel image (re-sized to 224x224) and, after a series of primarily convolutional layers, the model outputs probabilities for the 1000 classes using a softmax classifier. These probabilities are then used by DeepBoard to map user photos to distinct common categories so that the intended photos can be retrieved based on the user text.

It should also be noted that photos are only ever fed into the model during the indexing stage and for any future photos that the user takes. As discussed above, this process takes anywhere from a few seconds to a minute depending on the number of photos, however once complete the model will not need to process existing photos and the prediction can occur near instantaneously.

Semantic analysis

We have seen how the convolutional neural networks are used to classify the user photos. Now we will look at how they are mapped to the semantics of the text.

When a user presses the "Predict Photos" button, DeepBoard first starts by extracting the text in the current text container. Priority is given to text nearest to the cursor in cases when a large amount of sentences are present, such as in a document. This text will serve as the key contextual information for predicting the corresponding photos.

A naive approach would be to then directly compare the text to the image classifier labels, and if there is a match we present the relevant photos for that category. However, this has two major problems. First, this will only work for words which match the classifier labels exactly. So words which are semantically similar (e.g. synonyms, hypernyms) will not map to the intended category. Secondly, there will be too much noise from just extracting all words. We want to be able to prune the input sentence and extract only the key words which the user is interested in.

We solve these issues with the following approach. First, after the text is extracted, it is tokenized and passed through a

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Figure 4. The MobileNets architecture.

part-of-speech tagger for extracting the dominant nouns from the text. The dominant nouns are then compared to a custom lexical database of semantically linked words to determine which classes are relevant to the input text. This database is developed from portions of the WordNet database [4]. The WordNet database is a popular lexical database which links words by semantical relationships. Figure 5 shows an example of this.

The process of creating the custom database using WordNet for the classifier labels is as follows:

1. Iterate over all labels in the classifier.
2. For each label, extract its synonyms, hypernyms and hyponyms using WordNet.
3. Repeat this process a few times on the extracted words to build a tree of the semantically linked words for the given label.
4. Store the result in the custom database for the associated label.

The end result will be a portion of the WordNet database that is relevant only to the classifier labels. This makes the custom database very small and efficient for DeebBoard to use for class look-ups.

CHALLENGES AND LIMITATIONS

Developing any system for a mobile device has its challenges, but developing a keyboard service for the said device comes with even greater constraints. A custom smart touch keyboard has both memory and processing limits. We found these limits to be especially true on iOS, where the OS would kill the process with sometimes no warning. Furthermore, a smart touch keyboard needs to be responsive enough for the user to feel comfortable using.

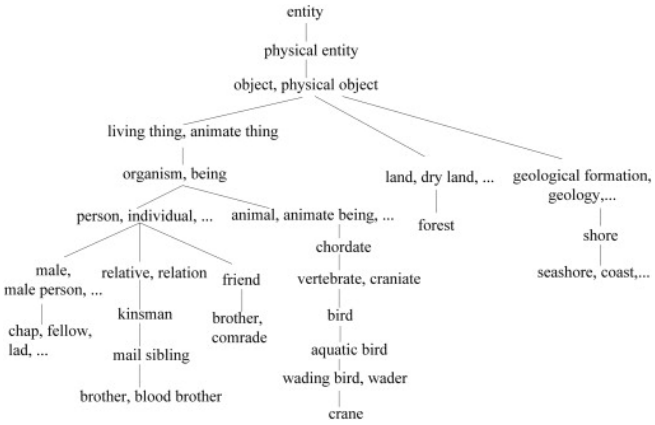

Figure 5. Structure of the WordNet lexical database.

We could have solved some of these issues by performing the work on a server, but this would violate the privacy design principle of DeepBoard. Instead, multiple device-level optimizations are used to guarantee responsiveness and efficient handling of resources.

Model sizes

Every model that DeepBoard uses internally is designed to consume as little memory and processing as possible. The pre-trained MobileNets model was specifically designed for mobile and embedded devices, so it consumes a relatively small amount and offers the benefit of low latency processing. This was essential for fast and efficient processing of user photos during the indexing phase. The custom semantic relational database also consumes little space as it was only designed from a small portion of the WordNet database.

It is possible to integrate larger models or additional models for improved accuracy and expanded coverage, but the device performance quickly becomes a limiting factor. Even with the deep learning model we currently use, it can take upwards of a minute to process every photo on the device. Additionally, the memory usage of DeepBoard already consumes roughly 80% of the available memory.

Hardware acceleration

One of the primary reasons for the deep learning model's efficiency is the use of hardware accelerated APIs available on iOS devices. DeepBoard uses Apple's recently developed Core ML API [1] for efficient on-device performance and low memory footprint.

Privacy

Privacy challenges and limitations exist when developing systems that process some form of user data. As discussed earlier, DeepBoard handles all processing local to the device, but even with this approach, there are still privacy challenges we

encounter. For example, we were unable to automatically insert the user photos directly into the rich text view due to iOS privacy restrictions. Instead the user must perform this action through the system clipboard via a copy-and-paste action.

Other privacy restrictions exist for third party keyboards on iOS as well. For example, no custom keyboard is given access to the microphone [3].

EARLY EVALUATION

In this section, we will look at an early evaluation of the DeepBoard keyboard using a small informal user study. A large formal user study is still needed, but this early evaluation provides some preliminary data on how well DeepBoard is currently performing.

Experiment considerations

It was a challenge designing an experiment to evaluate our system. There are currently no other keyboards that do photo prediction, and there are only certain times when a user plans on inserting a photo, so it's not a continuous process. We could have done a systematic evaluation of the model on pairs of text and photos, but developing a dataset for this task would take too much time and the results may not translate well to real user interactions. It was decided that a real-world experiment of the system with a few users would make for the best initial evaluation.

For the experiment, we decided to compare DeepBoard to the default iOS messaging app, iMessage. There are two reasons for this decision.

Consistency

The first reason is because there is no mechanism to insert photos using the default keyboard. Instead, each application handles its own insertion mechanism. So for consistency reasons, it makes sense to have a common application to compare to. The iMessage app is also a fair (and challenging) comparison because it is designed for users to send photos quickly.

Data collection

The second reason for using iMessage is because we could gather the most data as the users in the experiment used iMessage the most for sending photos. If we had picked a less popular application for sending photos, we would not have received enough data to make any interesting observations.

Experiment procedure

The experiment was conducted as follows. Three users were asked to install DeepBoard on their iOS devices. Each time a user sent a photo in iMessage using the "Predict Photos" feature, DeepBoard recorded the time it took for the user to insert the photo. The user would then manually time how long it took to insert the same photo using the default keyboard in iMessage. The user would also record if their intended photo was not predicted by DeepBoard.

Experiment assumptions

- As discussed above, the test was done only with the iMessage application.
- Photos which DeepBoard failed to predict did not have the time recorded. This is because it is useful to understand how well DeepBoard performs when it's working.
- Only the photo categories DeepBoard supports were used. This is because the model can be trained on any arbitrary categories, so it is more useful to understand how well DeepBoard performs on the given categories.

	DeepBoard (seconds)	Default Keyboard (seconds)
Count	20	20
Min	3.8	3.3
Max	9.8	19.3
Med	5.4	6.9
Avg	5.6	8.9
Std Dev	1.3	5.1

Table 1. The results for the time taken to insert a photo in iMessage using DeepBoard vs the default keyboard.

Results

The results of the experiment are shown below in table 1. The total number of predictions collected across all three users was 20. The time recorded is the amount of time (in seconds) it took for the user to insert the photo into the message. DeepBoard performed faster on average compared to the default iMessage keyboard. The iMessage keyboard also had a larger variance in time. Both keyboards have roughly the same best case insertion time, while the default iMessage keyboard has a poor worst case insertion time. The photo prediction accuracy for DeepBoard was defined as the number of times the intended photo was predicted divided by the total number of times the user requested a prediction. We report a photo prediction accuracy of **0.77**.

Feedback

The feedback we received for DeepBoard was overall positive. Users reported that when DeepBoard found their photo, the experience felt really satisfying. Users found the dynamic list of available categories very useful for quickly selecting the intended photo. Users reported that for recent photos, the experience on both keyboards was roughly the same, but for non-recent photos, DeepBoard saved them a large amount of time. Some users said that they wished this functionality was built into the standard keyboard.

Users were also asked about the responsiveness of DeepBoard, and if they noticed any latency issues, particularly during the indexing and prediction phases. All users reported that there were no noticeable responsiveness issues at any point.

There were also some criticisms of DeepBoard. Some users reported that sometimes the intended photo did not appear and/or the category did not appear. Other users reported that they wanted a way to view all photos for cases when the prediction process fails. Users also expressed that they wanted a way to correct missing or wrong classifications. One user said that it would be nice to have a "recents" section in addition to the prediction.

Discussion

While this is only an early evaluation and not a formal user study, we can still draw some interesting information from the results and the user feedback. The initial feedback and results suggest that DeepBoard indeed helps solve the problems with user photo insertion. DeepBoard allowed for faster photo insertion times on average and received positive feedback from the users.

One interesting outcome of the experiment is the effect of recent photos on the data. It appears that in cases when the photo is recent, both DeepBoard and the default iMessage keyboard perform similarly. This makes sense because the default iMessage keyboard has an interface for recent photos which doesn't require the user to traverse their photos. However in cases where the intended photo was not recent, the user then must search through all of their photos which could take potentially a lot of time. This makes photo prediction much more useful in these scenarios.

The data agrees with these findings and user feedback as well. We see that the default iOS keyboard had a greater variance and greater worst case which could result from the skew in time when photos were not recent.

Lastly, it is important to consider not just how fast DeepBoard performs at predicting but how accurate it is. As shown earlier, DeepBoard scored an accuracy of 0.77, meaning it correctly predicted the intended photo 77% of the time in our experiment.

This is a promising number for a few reasons. First, this is not just a classification problem, but instead there are multiple layers of models the text input must propagate through successfully in order for the intended photo to be predicted. For example, a photo may be classified correctly but the semantic analysis could fail for a particular word and the photo would not appear. Secondly, the results are similar to the expected classification accuracy in existing work [5].

PROFILING

We have seen how DeepBoard performs in a real-world experiment. Now we will look at the exact memory and CPU requirements of DeepBoard during normal use.

To do this, we have used the device profiler in Apple's Xcode to measure device performance while using DeepBoard. We profiled DeepBoard on an iPhone 6 running iOS 11.

Note that the device was not in the indexing phase when we profiled. The reason for this is because the bulk of the indexing phase only occurs once. We want to instead test how the device performs under normal use.

Figure 6 shows these results as reported from Xcode. We have found that our system under normal use consumes very little CPU and is marked with zero average energy impact. As shown in the last chart in figure 6, the CPU usage varies depending on when the user issues a photo prediction command. These results agree with the user feedback as users reported no noticeable responsiveness issues.

In terms of memory usage, our system is almost pushing the limits [3]. Despite the look of the chart, the CPU usage limit

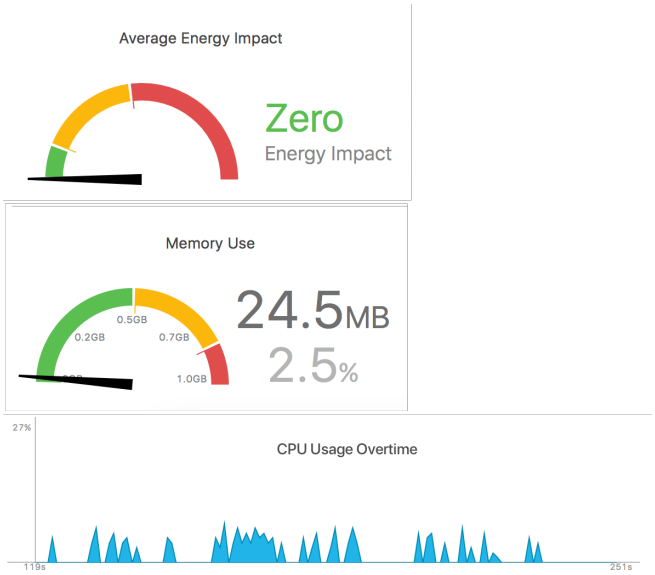

Figure 6. CPU usage, memory usage and battery impact of DeepBoard.

for keyboard extensions is around 30 MB. DeepBoard is consuming roughly 80% of this limit. Note that this is not necessarily a hard cap and may vary on newer devices. Regardless, DeepBoard is under this limit.

FUTURE WORK

There is certainly more room to improve DeepBoard. For starters, the user feedback suggests that users would like an interface for recent photos and the ability to view all photos for cases when the prediction fails. Users also wanted a way to correct incorrect predictions as well.

There is also a lot more room to improve the models as well. For example, we could use additional context such as location, contacts or current activity to help improve which photos to predict. We could train the model on a custom dataset that is more appropriate for user photo data and integrate additional models for increased prediction coverage.

Lastly, more work is needed in a formal user study. While the early evaluation is certainly useful, a proper formal user study could help reveal much more about the system and its use cases. We already have received useful feedback from just a few users, so a more developed study could certainly help improve DeepBoard.

CONCLUSION

We have seen how DeepBoard attempts to solve the problems with adding photos to messages or documents. Currently, it can take multiple steps for a user to search through their photos for the intended photo. However, DeepBoard uses the concept of keyboard photo prediction in order to present the user with photos based on the semantics of the text in the message or document. We have seen based on the evaluation and feedback that this offers an improvement over the default system keyboard in the popular messaging app, iMessage.

We have also seen the various optimizations and how DeepBoard retains its responsiveness despite the challenges of running deep learning models on mobile devices. We have also profiled DeepBoard to confirm this to be true.

DeepBoard is the only keyboard with photo prediction (to the best of our knowledge), so there is still much more room for others to work on this technology. We believe the concept of user photo prediction for keyboards is a novel concept that users would find very useful in their daily tasks.

REFERENCES

1. Core ML. **https://developer.apple.com/documentation/coreml**.
2. Google Gboard. **https://itunes.apple.com/us/app/gboard/id1091700242**.
3. iOS Keyboard Extension. **https://developer.apple.com/library/content/documentation/General/Conceptual/ExtensibilityPG/CustomKeyboard.html**.
4. Fellbaum, C. *WordNet: An Electronic Lexical Database*. Bradford Books, 1998.
5. Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).
6. Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (2012), 1097–1105.