AI Programming by Children

Ken Kahn, *toontalk@gmail.com*

Niall Winters, *niall.winters@education.ox.ac.uk*

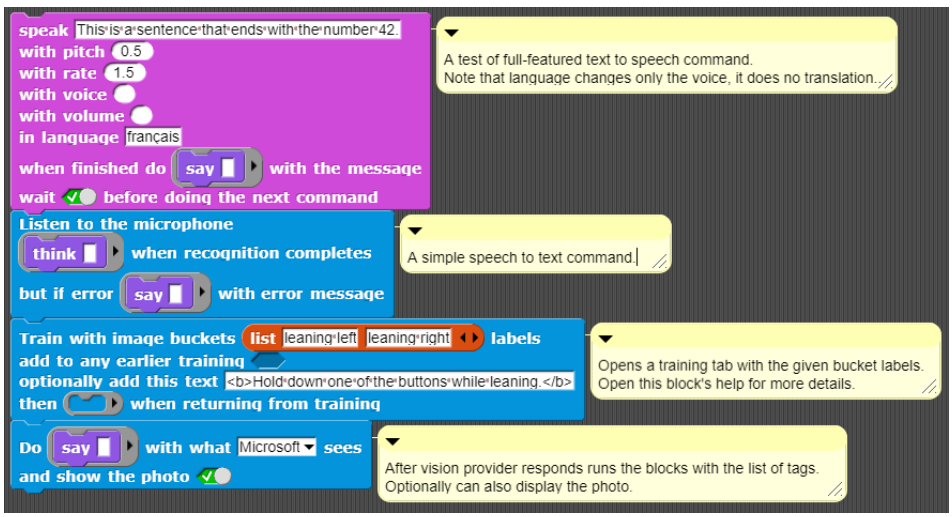
Department of Education, University of Oxford, 15 Norham Gardens, Oxford, OX2 6PY

Abstract (Research Paper)

The idea of children constructing artificial intelligence programs goes back to the early days of Logo (Papert & Solomon 1971; Kahn 1975; Kahn 1977). After decades of little activity recent efforts to support students in making AI programs has come from Stephen Wolfram (Wolfram 2017b), Google (Google 2018a, 2018b), the Machine Learning for Kids website (Lane 2018), and the eCraft2Learn project (eCraft2Learn 2018b). Two technological developments underlie the feasibility of these efforts: (1) AI cloud services and (2) mainstream laptops and desktop computers that now can run sophisticated machine learning algorithms. And all of these developments can be made accessible in a web browser, thereby running on many platforms without the need to install software.

Given appropriate programming tools children can make apps and intelligent robots that rely upon speech and image recognition services. They can add custom capabilities to their programs by using machine learning training and prediction. In doing so they may learn about perception, language, psychology, and the latest empowering technologies.

We describe the addition of new programming blocks to the Snap! visual programming language (Harvey & Mönig 2010) that provide easy-to-use interfaces to both AI cloud services and deep learning functionality. Interactive learning materials have been developed and preliminarily trialled with students. We anticipate in future trials to observe children creatively using these new blocks to build very impressive programs. Children are likely to be even more motivated to program when the results are such capable programs.



A small sample of new Snap! blocks for AI

Keywords

Visual programming; machine learning; block languages; Snap!, AI services; Cloud services; speech synthesis; speech recognition; image recognition;

Abstract

The idea of children constructing artificial intelligence programs goes back to the early days of Logo (Papert & Solomon 1971; Kahn 1975; Kahn 1977). After decades of little activity recent efforts to support students in making AI programs has come from Stephen Wolfram (Wolfram 2017b), Google (Google 2018a, 2018b), the Machine Learning for Kids website (Lane 2018), and the eCraft2Learn project (eCraft2Learn 2018b). Two technological developments underlie the feasibility of these efforts: (1) AI cloud services and (2) mainstream laptops and desktop computers that now can run sophisticated machine learning algorithms. And all of these developments can be made accessible in a web browser, thereby running on many platforms without the need to install software.

Given appropriate programming tools children can make apps and intelligent robots that rely upon speech and image recognition services. They can add custom capabilities to their programs by using machine learning training and prediction. In doing so they may learn about perception, language, psychology, and the latest empowering technologies.

We describe the addition of new programming blocks to the Snap! visual programming language (Harvey & Mönig 2010) that provide easy-to-use interfaces to both AI cloud services and deep learning functionality. Interactive learning materials have been developed and preliminarily trialled with students. We anticipate in future trials to observe children creatively using these new blocks to build very impressive programs. Children are likely to be even more motivated to program when the results are such capable programs.

Introduction to AI programming by children

From the early days of Logo research (Papert & Solomon 1971; Kahn 1975; Kahn 1977) there was interest in supporting children in creating artificial intelligence programs. The decision making, perception, learning, and natural language understanding in these programs was very simple due to the inabilities of the computers of those days. In the process of programming AI systems one naturally reflects on one's own thinking processes. This provided a very good fit for the constructionist ideas of learning through construction and reflection.

Among the many reasons for supporting AI programming by children are

1. Students may become better motivated and empowered to produce very capable artefacts
2. Students may learn about perception, reasoning, psychology, and animal behaviour in the process of building perceptive robots and apps
3. Students may learn about cloud services, machine learning, artificial intelligence, and other advanced technologies
4. Students may reflect more deeply upon their own abilities to hear, see, learn, and respond appropriately.

AI cloud services

Today's AI cloud services provide a new opportunity to support a new class of student AI projects – those that rely upon state-of-the-art AI "subroutines" (Kahn & Winters 2017). These services include speech synthesis and recognition as well as image recognition. Children can design and build impressive intelligent artefacts by composing and customising components provided by world-class AI teams.

Several companies are offering AI cloud services via a web connection. These include Google's machine learning services, IBM Watson cloud services, Microsoft cognitive services, and Amazon AI services. Many of these services include recognition services for obtaining descriptions of what is being spoken or seen. Other services analyse text for content, tone, and sentiment. They all include machine learning services that find patterns in data.

These are commercial services that cost a few dollars for a thousand queries. Fortunately for schools with limited budgets, free quotas are provided which allow a hundred recognition queries per day or a few thousand a month.

The service providers support accessing these services from many programming languages. Unfortunately, these are complex interfaces designed for use by professional programmers. In this paper, we show how to provide easy-to-use interfaces to these services, opening up their potential to children who are learning to program.

Students today are often doing physical computing projects involving micro-controllers such as Raspberry Pi, Arduinos, or Micro:bits. They are also often programming pre-built robots. In many cases these projects could benefit significantly from the ability to recognize what is being spoken or what is in front of a camera. For example, a student could build a robot that when it hears “push the red ball” will move to the ball and push it. This could be accomplished by sending the output from a microphone to an AI cloud service, picking out the keywords in the response, then repeatedly turning and sending images from a camera to a service until the response is that a red ball is in the image and then heading in the direction the camera is facing.

There is a long tradition of children programming language-oriented programs. In the early days of Logo children programmed poetry generators, silly sentence makers, chatbots, and more (Papert & Solomon 1971; Kahn 1975). The appeal of these kinds of projects increases when speech input and output replaces reading and typing, an area of research that has been neglected but can now be revisited to using the power of cloud-based AI services. In addition, student projects can use other AI services including sentiment analysis of what is spoken to respond in appropriate or amusing ways. This opens up the potential of AI to children in a simple and interactive manner, an emerging area of research we are exploring.

Machine learning

AI cloud services are created by large teams of experts using complex algorithms, massive data, and very large collections of servers. Systems are trained to recognise speech, images, video, sentiment, and more. Clients of these services have no control over the training. The services are black boxes to their users. In contrast, when children use machine learning software (Wolfram 2017b; Lane 2018) they need to provide the training examples before the system can do any classification or recognition. Programs can be created that respond to hand gestures, individual faces, or odd sounds that AI cloud services are not trained to recognise.

While there are cloud services for machine learning, we are not aware of any that have free quotas or are inexpensive enough to be used by school children. The Machine Learning for Kids website (Lane 2018) has a special arrangement with IBM for limited use of their machine learning cloud services. Consequently, the site requires registration and limits the number of projects allowed.

An alternative to cloud services appeared recently (tensorflow.js 2018). Tensorflow.js (formerly named deeplearn.js) is an open source JavaScript library that is able to support both training and prediction using deep neural nets. It runs in a browser and is accelerated hundreds of times by its use of graphical processing units (GPUs). These are now common on modern laptops, desktop computers, tablets, and smartphones. For example, a high-end laptop can process up to ten images per second during training and make up to twenty predictions per second. Programs using tensorflow.js on devices that run at one tenth of that speed can still support a wide range of applications.

Machine Learning for Kids

The Machine Learning for Kids website (machinelearningforkids.co.uk) provides an interface where users can define a number of labelled buckets and fill them with images, text, or numbers.

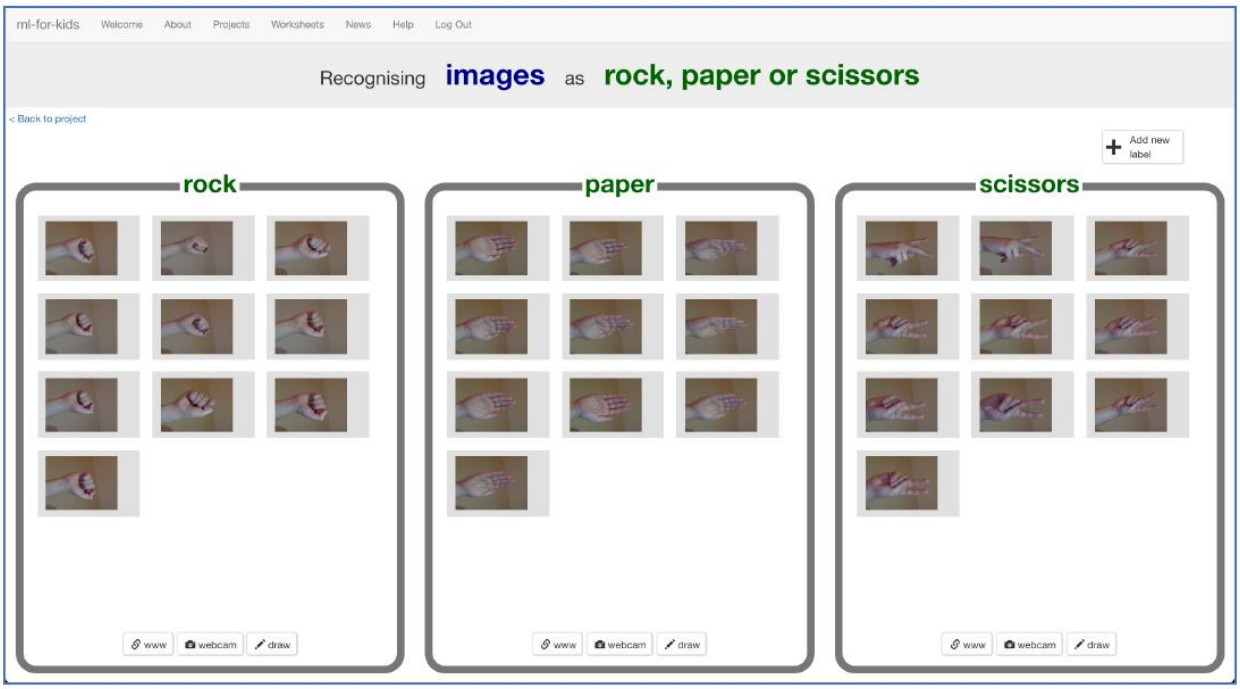


Figure 1 - Training to classify a hand as rock, paper, or scissors at Machine Learning for Kids

After the buckets have been filled they are sent off to the IBM Watson servers to generate a model for classifying new images. It then adds new blocks to the Scratch programming language (Resnick et al 2009) that can be used to determine which category a new costume image belongs to. Another Scratch block is available to send new data for additional training.

The Machine Learning for Kids website has many suggested projects. For example, one implements sentiment analysis using textual data. Other projects involve learning to play Pac Man or Tic-Tac-Toe (Noughts and Crosses) using numerical training data. The Pac Man program learns to play better by adding additional training data whenever a human playing the game makes a move. The Machine Learning for Kids website and project ideas inspired the machine learning work described in this article.

Machine Learning in Wolfram Language

Stephen Wolfram wrote a blog post entitled *Machine Learning for Middle Schoolers* (Wolfram 2017b) about the new extensive section of his Wolfram Language textbook (Wolfram 2017a). (The Wolfram language is the language of the Mathematica system.) In this blog post he describes a variety of machine learning programming exercises including classification, clustering, text recognition, image recognition, feature extraction, feature spaces, and training. The textbook explains how neural nets can be defined and visualised.

Teachable Machine and Model Builder

Google recently released Teachable Machine (Google 2018c) which demonstrates image training and classification in a web page. It directly inspired the machine learning work reported below.

Google also recently released the Model Builder (Google 2018b). This is a website where one can design and train a deep neural net and run experiments on it. While one can learn a great deal about machine learning from this site, it doesn't provide a programming interface for creating apps that use the trained models.

AIY Projects

Google has begun to release a series of do-it-yourself AI kits they call AIY kits (Google 2018a). The kits are based upon Raspberry Pi computers. The first kit enabled one to build a box with a

button that can do speech recognition and synthesis. It can answer questions the same way “OK Google” does. The second kit is for building a box that can do image recognition. Sample programs written in Python are provided.

Creating Child-friendly Programming Interfaces

Our efforts as part of the eCraft2Learn project are not to create a new programming environment for children, but instead to enhance existing ones. We have added speech input and output to ToonTalk Reborn (Kahn 2014) and to Snap! (Harvey & Mönig 2010). We have also added image recognition and machine learning to Snap!. This paper reports on our Snap! efforts.

Why was Snap! Chosen?

Snap! is a superset of the very popular children’s blocks-based programming language Scratch (Resnick 2009). It is well-suited to our efforts because

1. It is a powerful language that supports first-class data structures and functions
2. It is easy to define new blocks using JavaScript without touching the source code
3. It runs in every modern browser
4. There are versions that connect to Arduinos and Raspberry Pis

Speech synthesis

All the popular browsers except for Internet Explorer support the Speech Synthesis API. (There is a problem with Chromium that no voices are installed.) The API utters the provided text with control for the pitch, rate, volume, language, and voice. An issue arises because the API instructs the browser only to begin speaking. The new Snap! block that invokes this API returns at once and yet some projects need a way to respond to the speech finishing. This was accomplished by passing to the Snap! block an optional function that is called when the browser signals the event that the speech has ended.

Figure 2 - Simple speak command takes a text argument and an optional continuation function

More advanced users can use a full-featured ‘speak’ command that exposes most of the Speech Synthesis API functionality. While it is more difficult to use, all but the first parameter are optional and can be ignored. During testing students were clearly amused by entering different values for the pitch, rate, and voice. The underlying API expects the language parameter to be given as a BCP 47 language tag. This is a string that specifies the language and dialect. E.g., fr-FR is French as spoken in France while fr-CA is the Canadian dialect. This is not very user friendly. We addressed this by providing a Snap! reporter that supports search terms for installed voice names (e.g., “English”, “UK”, and “Female” in Figure 3). We also defined a way to change the default language for speech synthesis and recognition by giving the name of the language in the language or in English.

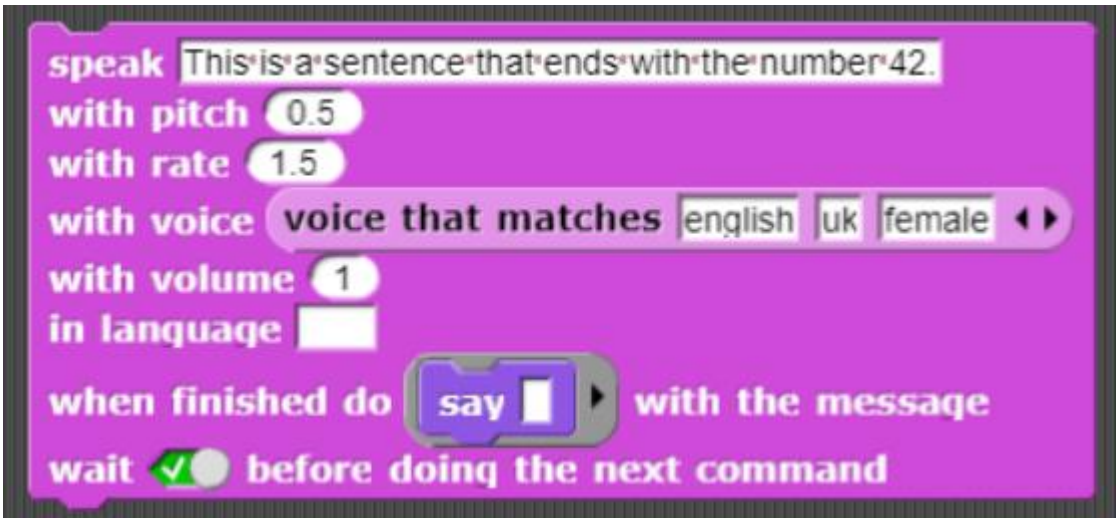


Figure 3 - Advanced speak command that exposes nearly all the functionality of the Speech Synthesis API

Speech recognition

While speech recognition is part of the Web Speech API, as of this writing only Chrome and Opera support it. However, AI cloud services for speech recognition are available from Google, Microsoft, IBM, and others.

Figure 4 - Simple speech recognition block for receiving text recognised and errors

(Empty fields here are shorthand for the argument passed to the command. The first one is what was heard and the second is the error message.)

The asynchronous nature of recognition services forces a reliance upon continuations (also called “callbacks”). Continuations are ideal from a technical point of view; however, we are concerned that student programmers may find them difficult. However, when students learn that the two fields of the listen command are simply places to put commands that run when and if the speech is recognised or when an error occurs. The combination of handling failures, errors, multiple outputs, and the asynchrony make the use of continuations more appropriate than Snap! reporters.

As an easier alternative to the use of continuations, we also implemented a block using the ‘listen’ block that supports event broadcasting and uses global variables:

Figure 5 - A block that broadcasts speech recognition results

The underlying complexity is hidden (but available for ambitious students) so that one needs only to call the ‘broadcast speech recognition results’ block and then receive broadcasts when

something was heard and read a global variable containing the last thing spoken. For example, the following program repeats what was spoken, prefaced with “I think I heard”. Besides a questionable reliance upon global variables this technique makes it very difficult to respond to an utterance that wasn’t expected with a response such as “I don’t understand …”.

Figure 6 - Using broadcasting to respond to recognised speech

For more advanced uses of speech recognition, a user may want access to partial results as one is speaking, to receive alternative interpretations of what was said along with their confidence scores. And one may want to specify which language is expected. The following block provides this functionality. In Figure 7 while speaking partial results are displayed in a thought bubble (unless there is an error). French is expected. Up to five alternative interpretations of what was said will be displayed.

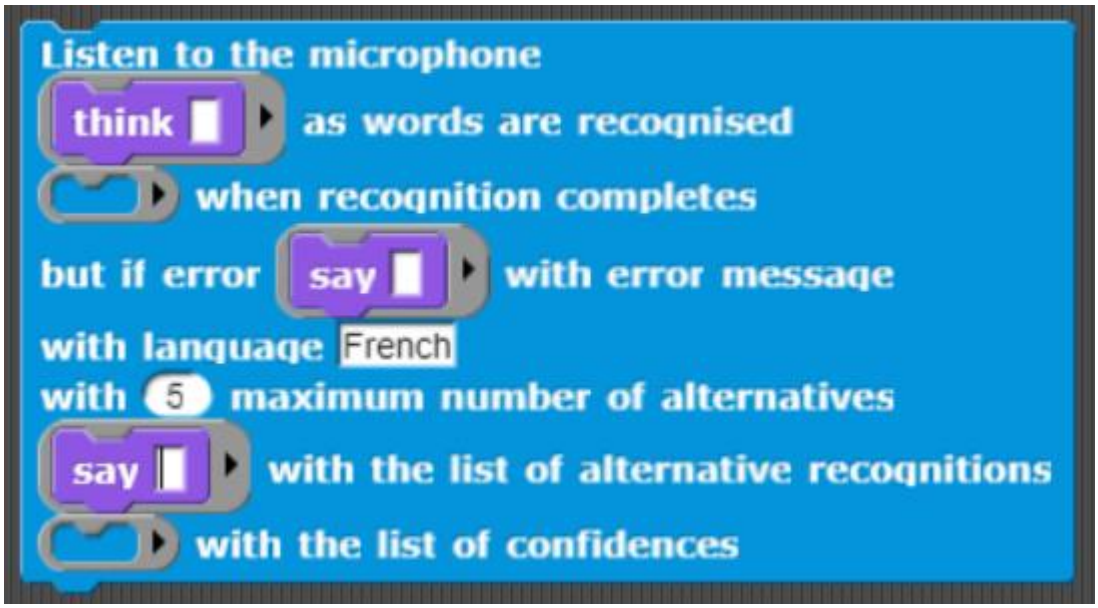


Figure 7 - A full-featured speech recognition block

Image recognition

There are no standard APIs for image recognition so the block we implemented supports the Google, IBM, and Microsoft APIs. The simple image recognition block has a parameter specifying which cloud provider, a continuation that will receive a description of the image, and a flag as to whether the image should be displayed.

Figure 8 - A simple block for obtaining a list of labels of what is in front of the camera

The responses from the vision recognition services are tables containing tables sometimes containing tables. And each provider has different structures. We provide blocks for accessing all the information a vision service provides. For example, The Google vision API supports more than labelling what is in the image. One can access text recognition, face detection, and image properties results as well.

API Keys

All the AI cloud vision services require that requests be accompanied by API keys. These keys are easy to obtain and entitle the user to a moderately generous free quota of requests. It is not wise to share widely a project that contains API keys since the free quota will be exhausted quickly. The first solution we implemented was that the keys are provided as URL parameters. This wasn't very child friendly and became too clumsy for projects that rely upon more than one AI service provider. We eventually settled on using Snap! reporters to hold the keys. Our Snap! blocks use keys if provided and if they are missing offers to take the user to a page documenting how to obtain keys.

Machine Learning Snap! Blocks

At the time of this writing the machine learning Snap! blocks are limited to images (and a crude learning algorithm for audio) The underlying technology can be enhanced to support audio, video, text, and data. The interface to machine learning requires a second browser tab in addition to Snap!'s. Technically this is necessary since the machine learning software uses the GPU and Snap! uses it for graphics via WebGL. It is also motivated from a user interface perspective since the second tab is well-suited for both training and prediction feedback.

The following command launches a tab for training whether you are leaning to the left or the right. It overrides the default text on the page.

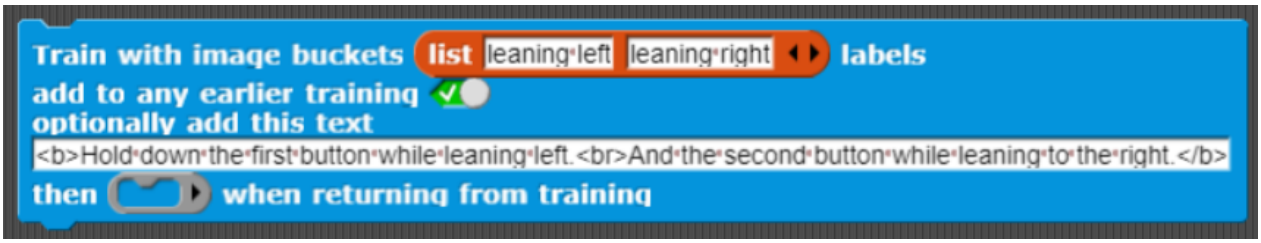


Figure 9 - Launching a training tab

The tab created includes buttons for adding new images to the training. It also provides feedback as to how confident it is that it can label the current image.

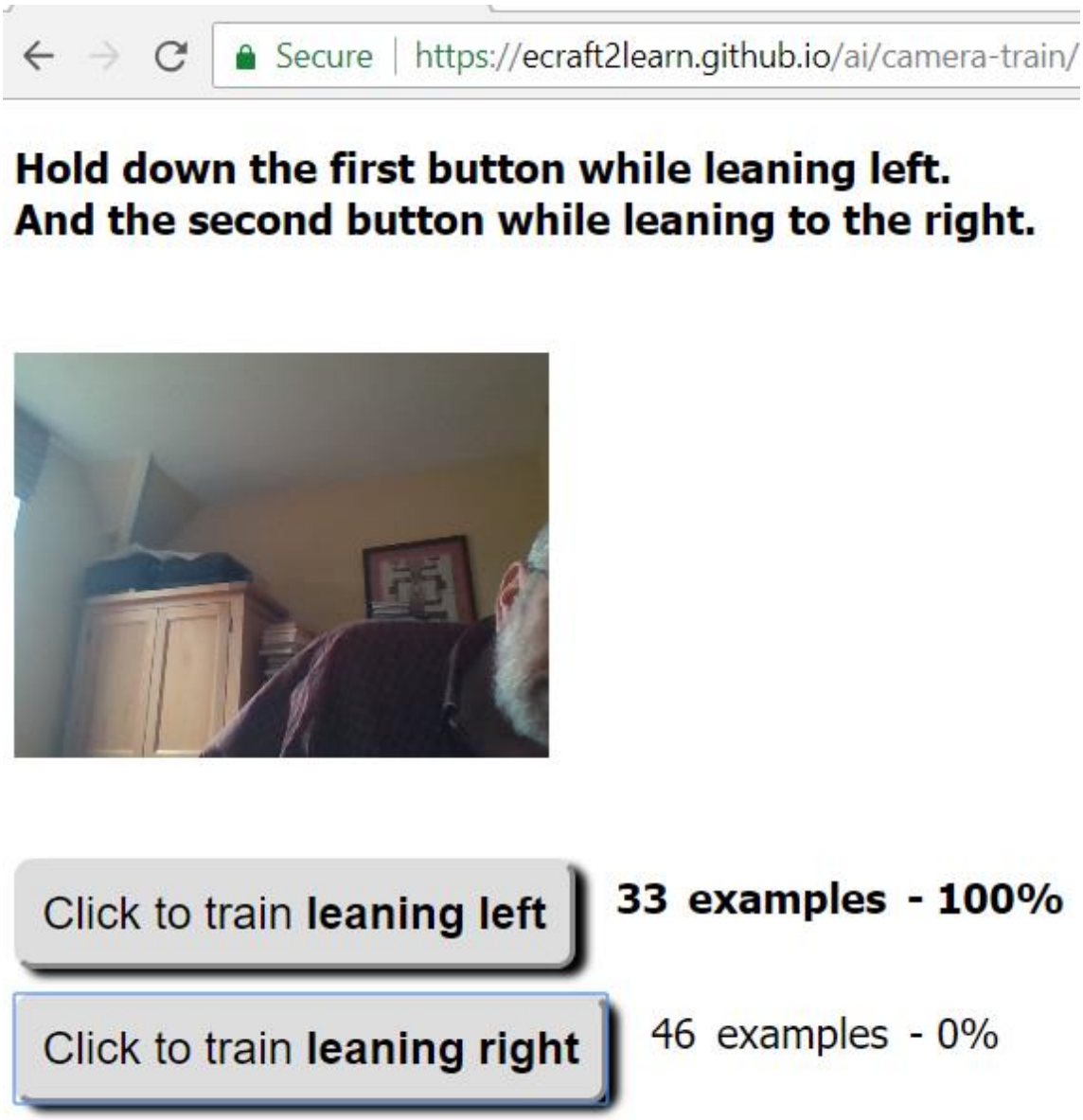


Figure 10 - A machine learning training window

Upon returning to the Snap! tab one can send an image from the camera to the training tab to obtain a list of the confidences that each label applies to the image.

Figure 11 - A block to obtain label confidences

An alternative to training using the camera is to use blocks that use the costumes of the Snap! sprites.

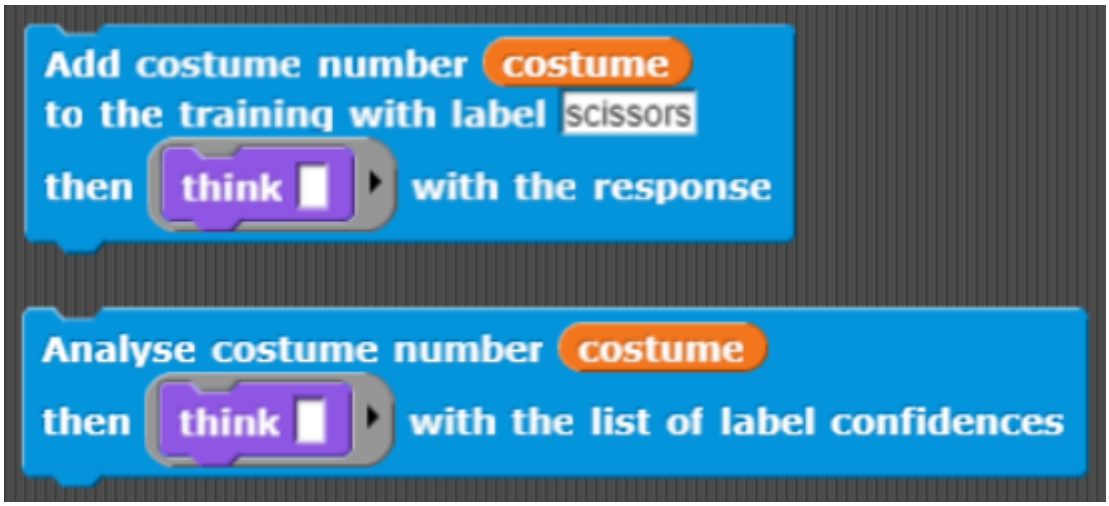


Figure 12 - Blocks to use costumes for training and prediction

A block has been defined that uses the 'Add costume …' block to send all the costumes of a sprite to the training tab. These costumes can be drawn, imported, or captured by a camera. Currently the underlying library providing the machine learning capabilities (tensorflow.js 2018) has no general way to save a model after training so applications need to run training before doing classification.

Student and teacher guide to AI programming

An extensive interactive guide to using all of these new Snap! blocks has been created. There is a student and a teacher version (though they are over 90% identical) (eCraft2Learn Project 2018b). The guides are web pages with dozens of Snap! projects embedded as iframes. Some are simple interactive introductions to a single block while some are more complex demonstration programs.

In addition to the technical content the guides include discussions of how the underlying technology works as well as societal impact. They also include exercises and project ideas.

The guides can be viewed via a local web server. This is useful when the Internet connection is slow or intermittent. If there is no Internet connection about half the guide's interactive elements are still functional. The machine learning chapter works fine locally without a network connection.

Internationalisation

Google Translate has been integrated with the AI programming guides, documentation, and the training tab interface. The speech synthesis and recognition blocks by default use English but there is a block for setting the default to any of the many languages that Chrome supports. Snap! itself supports several dozen languages. Unfortunately, the labels and help messages of the AI blocks themselves are not translated.

Sample programs using AI blocks

Currently ten sample programs are available that use these new blocks. They are designed to be readable and editable by students. Many of them implement only part of what a full app would contain leaving the missing parts for students to add.

1. Speak with random pitch, rate, voice, and language (Speech synthesis)
2. Speak commands to a sprite (Speech synthesis and recognition)
3. Generate funny templated sentences (Speech synthesis and recognition)
4. Generate template stories (Speech synthesis and recognition)

5. Talk to Wikipedia (Speech synthesis and recognition)
6. Report what is in front of the camera (Speech synthesis and image recognition)
7. Speak what is in front of the camera when a cloud service is asked to (Speech synthesis, speech recognition, and image recognition)
8. After training watch the turtle move depending on which way your finger is pointed or sounds you make (Machine learning)
9. After training watch the turtle move left or right depending on which way you lean (Machine learning)
10. Play a Rock Paper Scissors game by configuring your hand in front of the camera (Speech synthesis and machine learning)

All of these programs and the new blocks are available at (eCraft2Learn Project 2018b).

Using NetsBlox (NetsBlox 2018), a Snap! variant that supports multi-player projects, and together.js (together.js 2018), a library for creating multi-player JavaScript programs, we created a two-player version of Rock Paper Scissors. In the first phase both players connected via the Internet can train the same model to recognise whether a hand is showing rock, paper, or scissors. In the second phase the program says “1, 2, 3, go” and then the trained model and cameras on both computers are used to classify each players moves. Speech synthesis is used to inform the players of the outcome.

Field testing

The speech input and output blocks have been tested with 25 Singaporean undergraduate students, 18 children (aged 7 to 13, Sri Lankan and Singaporean) in afterschool programs and 40 Indonesian high school students. This preliminary testing has been very encouraging. The new Snap! blocks were easily understood and the users indicated that they enjoyed adding speech to their programming projects. 12 of the 18 children and all the high school students were also introduced to machine learning Snap! blocks and mastered several machine learning programming exercises.

A formal study of 40 Indonesian high school students using speech synthesis, speech recognition, and machine learning has been submitted for publication. Teachers in Finland and Greece have reported successfully introducing the Snap! AI blocks.

Conclusion and Discussion

We began by describing the history and current efforts to support AI programming by children. We then presented programming constructs we have developed that are suitable for use by beginners for speech synthesis, speech recognition, image recognition, and machine learning (of images). Artificial intelligence is much broader than this and includes common sense, planning, reasoning, understanding language, unsupervised learning, translation, and more. The research reported herein is just the start of an effort to give children the ability to construct AI applications.

The goal of this research is to demonstrate that AI programming need not be limited to those with advanced degrees. Even young children can be empowered to creatively use AI programming to make apps that speak, listen, see, and learn.

Acknowledgements

This research was supported by the eCraft2Learn project funded by the European Union’s Horizon 2020 Coordination & Research and Innovation Action under Grant Agreement No 731345.

References

Tensorflow.js (2018) https://js.tensorflow.org/.

eCraft2Learn Project (2018a) http://www.project.ecraft2learn.eu/.

eCraft2Learn Project (2018b) https://ecraft2learn.github.io/ai/.

Google (2018a) AIY Projects. https://aiyprojects.withgoogle.com/.

Google (2018b) Google Model Builder. https://deeplearnjs.org/demos/model-builder/.

Google (2018c) Teachable Machine. https://teachablemachine.withgoogle.com/.

Harvey, B., Mönig, J., (2010) Bringing "No Ceiling" to Scratch: Can One Language Serve Kids and Computer Scientists? In Proceedings: Constructionism, Paris, France.

Kahn, K. (1975) A Logo natural language system. Technical report, MIT AI Lab, LOGO Working Paper 46.

Kahn, K. (1977) Three Interactions between AI and Education. *Machine Intelligence 8.*

Kahn, K. (2014) ToonTalk Reborn: Re-implementing and re-conceptualising ToonTalk for the Web. In Proceedings: Constructionism, Vienna, Austria.

Kahn, K., Winters, N. (2017) Child-friendly programming interfaces to AI cloud services. In Proceedings: EC-TEL 2017: Data Driven Approaches in Digital Education, 10474, 566-570.

Lane, D. (2018) Explaining Artificial Intelligence. *Hello World 4*. Raspberry Pi Foundation. https://helloworld.raspberrypi.org/issues/4.

NetsBlox (2018) https://netsblox.org/.

Papert, S., Solomon, C. (1971) Twenty Things to Do with a Computer, MIT AI Lab, http://hdl.handle.net/1721.1/5836.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B. and Kafai, Y. (2009) Scratch: programming for all. *Communications of the ACM*, 52(11), 60-67 DOI=http://dx.doi.org/10.1145/1592761.1592779.

Together.js (2018) https://togetherjs.com/.

Wolfram, S. (2017a) *An Elementary Introduction to the Wolfram Language, Second Edition*. Wolfram Media.

Wolfram, S. (2017b) Machine Learning for Middle Schoolers. Stephen Wolfram blog. http://blog.stephenwolfram.com/2017/05/machine-learning-for-middle-schoolers/.