DeepScratch: Scratch Programming Language Extension for Deep Learning Education

Nora Alturayeif[1], Nouf Alturaief[2], Zainab Alhathloul[3]
Department of Computer Science, King Fahd University of Petroleum & Minerals,
Dhahran, Saudi Arabia

***Abstract*—Visual programming languages make programming more accessible for novices, which open more opportunities to innovate and develop problem-solving skills. Besides, deep learning is one of the trending computer science fields that has a profound impact on our daily life, and it is important that young people are aware of how our world works. In this study, we partially attribute the difficulties novices face in building deep learning models to the used programming language. This paper presents DeepScratch, a new programming language extension to Scratch that provides powerful language elements to facilitate building and learning about deep learning models. We present the implementation process of DeepScratch, and explain the syntactical definition and the lexical definition of the extended vocabulary. DeepScratch provides two options to implement deep learning models: training a neural network based on built-in datasets and using pre-trained deep learning models. The two options are provided to serve different age groups and educational levels. The preliminary evaluation shows the usability and the effectiveness of this extension as a tool for kids to learn about deep learning.**

***Keywords*—Deep learning; visual programming languages; programming education; formal language definitions; neural networks**

I. Introduction

Programming nowadays is considered an essential skill and has been introduced in a novice level for different ages. Moreover, visual programming languages make programming more accessible for young people, which open more opportunities to innovate and explore. Scratch [1]; a visual programming environment developed by MIT, is one of the most popular block-based visual programming languages that allows users to create interactive and media-rich projects. On the other hand, deep learning is one of the trending computer science fields during the last years, and it acquired interest and focus in many different fields. Deep learning has a profound impact on our daily life and it is important that young people are aware of how our world works. However, it is not an easy task to understand the concepts of a deep learning as it requires deep understanding of mathematics and calculus. Understanding and applying deep learning requires spending hundreds of hours learning and debugging code, which is mostly frustrating for juniors.

The aim of this research is to extend the vocabulary of Scratch programming language to help young people designing and implementing deep learning applications. Deep learning has a profound impact on society. It has many applications in finance, healthcare, customer experience, weather prediction, etc. Nowadays, it is important that kids are aware of how the world works and understand the capabilities of deep learning. This paper presents DeepScratch, a new programming language extension to Scratch that provides powerful language elements to facilitate deep learning concepts to allow kids and high schoolers to understand and develop deep learning applications. This research is an extension of the paper: "Extending Scratch: New Pathways into Programming" [2].

DeepScratch provides two options to implement deep learning models: training a neural network based on built-in datasets, or using pre-trained deep learning models. The two options are provided to serve different age groups and educational stages.

This paper introduces two main contributions:

- Extend the vocabulary of Scratch visual programming language to enable developing deep learning applications using Scratch, which opens an opportunity for researchers to continue and expand our work.
- A tool for educators to teach kids basic deep learning concepts (different neural networks architectures, hyper-parameters tuning, and classification metrics).

Being able to build deep learning application with visual programming language should be very useful for kids. In addition, high schoolers who are interested in deep learning can implement various applications in an environment that does not require understanding of programming, mathematics, and calculus concepts. We believe that this work will help in closing the knowledge gap between educators and students, thus enabling them to explain machine learning concepts in an environment that are more suitable for novice programmers. Our preliminary evaluation showed significant effects of using DeepScratch on students' understanding of deep learning.

The rest of this paper is organized as follows: Section II introduces the background. Section III presents previous studies that proposed related applications. Section IV introduces the methodology followed in this study. Section V describes the syntactical definition (grammar) used for DeepScratch extension. Section VI presents the lexical definition of DeepScratch by explaining the functionalities of the extended vocabulary.

Section VII demonstrates some examples of simple programs developed using DeepScratch. Section VIII discusses the significance DeepScratch and describes the conducted evaluation process to ensure the functionality and the usability of the developed extension. Finally, the conclusion is presented in Section IX.

II. Background

Visual programming languages allow users to develop programs by manipulating elements graphically instead of writing a program as a text. These languages can potentially allow young people to acquire the computational concepts more easily by reducing unnecessary syntax and facilitating the use of dragging and snapping the command blocks. With such features, these frameworks can help reduce the cognitive load on novices by allowing them to focus on the logic and structures of a program rather than worrying about the syntax and the mechanism of coding [3].

There is a rich history of different visual programming tools designed for novices comprehensively surveyed in [4]. AgentSheets by Repenning and Sumner [5] is a tool that introduced the blocks programming in 1995 to create games and simulations. Their work marked a substantial step in the field of visual programming language [6]. Several block-based programming language were designed after AgentSheet, such as Squeak eToys, Alice, and Scratch [6].

Scratch was created by MIT Media Lab's Lifelong Kindergarten Group in collaboration with Yasmin Kafai's group at UCLA [7]. The main idea for Scratch was inspired by LEGO bricks, as Scratch research team worked closely with LEGO company [8]. Scratch grammar was converted to a programming blocks which represent the bricks in the LEGO. To create a program, users need to simply tinker the blocks together [8]. Thereafter, in the third version of Scratch, they collaborated with Blockly, a project developed by Google. Blocks are end-user composable, editable, and can be arranged geometrically to represent tree structure and to define syntax [9]. The previous four keys form the properties of a highly accessible visual programming paradigm combined by the AgentSheets [6]. According to [6], Scratch and Blockly adopted these properties to be in their core, and became the popular blocks programming language. Fig. 1 demonstrates how a Python code block looks like in Scratch programming language.

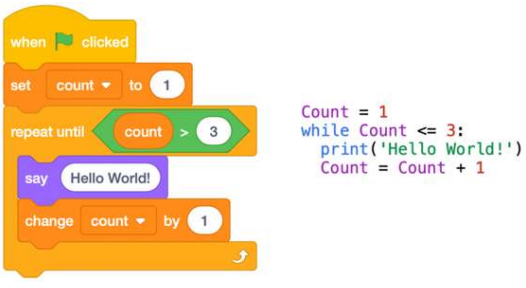

Fig. 1. Scratch blocks and its equivalent Python code

In 2015, Scratch team presented the scratch extension system to enable programmers to innovate on the language itself by extending it [2]. In addition, "enabling learners with a diverse set of interests to engage in programming with Scratch by opening up a number of previously unavailable pathways, through new domain-specific programming primitives" [2]. However, deep learning development is not supported yet by the original Scratch language or as an extension to this language.

Deep learning is a sub-field of machine learning dealing with algorithms that are similar to how nervous system structured, where each neuron is connected and passing information to each other [10]. Deep learning aims to learn complex relationships among data that make out meaningful results, and predict from multiple applications.

III. Related Work

Machine learning, deep learning, and data science are Key topics that are not yet addressed much in the existing tools dedicated to youths. These are advanced fields for novices, but becoming essential to learn for the world we live in today and the one we will experience in the future.

Few systems have been designed to provide a straightforward and simple understanding of data science and machine learning geared towards kids or high school students. For Data Science applications, *Scratch Community Blocks* [11] is a system that enables children to programmatically analyze and visualize data about their participation in Scratch, and learn how to reason about complex information visualizations. *DataSnap* [12] is an extension to the block-based language *Snap* which can fetch and analyze data from online sources.

Similarly, Machine Learning is available as an education tool using different programming languages. *Machine Learning for Kids* project [13] brings in the power of the IBM Watson engine by presenting Machine Learning Building Blocks such as image recognizers and text classifiers that can then be imported to a Scratch program. Ken Kahn [14] has created resources to allow beginners to create AI applications such as speech synthesis, speech recognition, and image recognition, in the block-based language Snap.

Teachable Machine [15] is a web-based tool developed by Google that allows users to train and test machine learning model without writing code. However, Teachable Machine is a very abstract tool and does not introduce basic machine learning concepts such as hyper-parameters, different algorithms, and evaluation metrics. Our research aims to introduce an extension of Scratch programming language that covers more advanced machine learning concepts, specifically, deep learning concepts.

Our literature review revealed the need to develop an environment that supports teaching kids different machine learning concepts including learning about different neural network architectures, optimizing a neural network, and performing

hyper-parameter tuning. Previous studies tackled this topic in a more abstract level, allowing the kids to build machine learning applications without learning the underlying processes and concepts of the machine learning models.

IV. Methodology

The Incremental Model is used to develop the DeepScratch extension. This model is based on developing an initial component, which usually includes the fundamental requirements. After evaluating the initial component, several other components are added until the software is fully developed. The Incremental Model can be Plan-Driven or Agile. In this research, the Agile approach is adopted as it does not require a pre-planned set of increments. Agile model makes the development process flexible and cost-efficient to adapt to the changes [16]. Since this is a research-based project, and research is expected to be exposed to changes; the Incremental Agile Model is selected.

A. Process Activities

- **Software specification:** It is the process of understanding and identifying the required services for the software [16]. In general, the specifications of this extension include implementing the following features:
  - Input methods: since deep learning applications require a lot of data that need to be labeled, built-in datasets are available in DeepScratch, which are the *Iris* and *MNIST* data sets.
  - Neural networks: Dense, RNN, and CNN.
  - Predicting new data.
  - Evaluation metrics: loss function (cross-entropy), training accuracy, and testing accuracy.

- **Software design and implementation:** It is the process of converting the software specifications into executable software. However, this activity might include some refinements on the software specification [16]. The design of our extension follows the design principles established by Scratch. These design principles aim to help the developers to design simple and easy to use language that encourages users to quickly explore and experiment with the language functionalities [9]. Scratch allows extending the programming language using JavaScript [2]. Therefore, *TensorFlow.js* [1]; a library for deep learning in JavaScript will be used for the implementation.

- **Software validation:** Verification and validation is applied to each increment based on its requirements. Once the software is complete, a full system functionality testing will be applied as the final stage of the testing process. Functionality testing is used to ensure that the software meets its specifications and is ready to be published [16]. Moreover, usability testing will be conducted by a focus group of kids to validate the extension's ease of use.

- **Software evolution:** As this is a research-based project, this step is considered as the future work.

[1] https://www.tensorflow.org/js/

V. Syntactical Definition

Any visual programming language (VPL) is characterized by two main elements: a grammar (syntactical definition) and a vocabulary (lexical definition) [17]. Together they define the set of concepts that can be expressed with the programming language. The grammar of VPL is described by the visual metaphor, such as blocks and wires. Whereas the vocabulary is the collection of blocks, icons, or other components that allow a programmer to express concepts.

This section explains the grammar of Scratch, which will be the base of the proposed extension. In practice, to run a text-based program, a program takes the program as an input and extracts lexemes (sequence of characters in the source program) and tokens (categories of lexical units). Then, following the context-free grammar, a program takes the tokens and creates a parse tree. Since Scratch is not a text-based language, the interpreter does not need to tokenize and parse the program.

The grammar of a text-based language is usually defined by metasyntax notation such as EBNF (Extended Backus–Naur Form). However, the grammar of Scratch is described by defining blocks of different shapes representing expressions, statements, and control structures. These shapes are fitting together in only syntactically-correct ways. This eliminates syntax errors by setting geometric relations rules (containment, horizontal/vertical concatenations, etc.) to connect the blocks together [18].

There are six shapes of Scratch blocks: *Hat* (trigger blocks), *Stack* (command blocks), *Boolean*, *Reporter* (function blocks), *C* (control structure), and *Cap* [1]. Each shape has its own function and properties [1], [18]. In our work, we will use the Stack and Reporter blocks to build Scratch-based deep learning models. The Stacks (command blocks) are like the statements of a text-based language, they are shaped with a notch at the top and a bump on the bottom, so that blocks can be placed above and below them to create a sequence of commands [1]. Whereas Reporters (function blocks) hold values (Number or String) as constants or in a variable. They can be used as arguments to commands to build expressions [1].

According to [2], the affordances of the Scratch extension system allow for extending the vocabulary rather than the grammar, and thus, DeepScratch extends the language by augmenting the vocabulary. The following section will describe the extended vocabulary in detail.

VI. Extension's Vocabulary

There are 119 blocks working as the standard Scratch vocabulary (not including extensions) [18]. The syntax of some of these blocks is illustrated in [18]. This section demonstrates the vocabulary of DeepScratch, which are custom programming blocks written in JavaScript. While adhering to Scratch grammar, each extension block is mapped to a JavaScript script that gets invoked through a "bridge" layer implemented within Scratch. In addition, *Tensorflow.js* library is utilized for building and executing deep learning models that run in a web browser and in the Node.js.

Our Scratch extension will provide the users with two options to implement deep learning models. The first option enables the user to train the model, by giving the ability to choose the dataset, the architecture of the neural network, and a variety of hyper-parameters. Dense, RNN, and CNN are the architectures of neural networks available for training in DeepScratch. The other option allows the user to run pre-trained models that can be used to predict new data. DeepScratch will support different pre-trained models offered by TensorFlow from Google, such as object detection model. Sections VI-A and VI-B describe these two options in detail.

The complete vocabulary of DeepScratch is represented in Tables I and II. The DeepScratch blocks can be combined with other Scratch blocks to build an application, such as adding "say" block by Scratch to present the prediction result. Lastly, to save our work and for further development, we made our code available through GitHub[2].

A. Training Deep Learning Model Blocks

Training a deep learning model needs expertise in certain programming languages. This process is simplified through DeepScratch blocks that bind together as in Fig. 1.

Each machine learning program consists of three main stages: pre-processing data, training a model, and predicting new data. Preparing and pre-processing data can be complicated and frustrating for young people, and moreover, it's not in the scope of how machine learning models work. Hence, the proposed extension provides built-in datasets that are ready to be trained. These datasets are the popular Iris[3] and MNIST[4] datasets.

In DeepScratch, the first step in training a machine learning model is to choose which combination of a dataset and a neural network model to use. The available NN models are Dense, RNN, and CNN. Each model can be tweaked by changing some hyper-parameters specific to that model. For example, in the CNN model, the user can change the batch size and set the number of epochs as optional hyper-parameter. During training, the user will be able to monitor how the model optimizes by displaying accuracy and loss values. Once the training is done, the user will be able to check the accuracy on the testing data. This way, the user can learn how different hyper-parameters settings can affect the performance of the model. In Fig. 2, the user used the *train Dense* block to train the Iris dataset using a Dense model with two hidden layers and 20 epochs. If the user trains the model without setting the optional block (*number of epochs*), it will take the default value: 10 epochs. At last, the user used the basic *say* block to display the testing accuracy once the training is done.

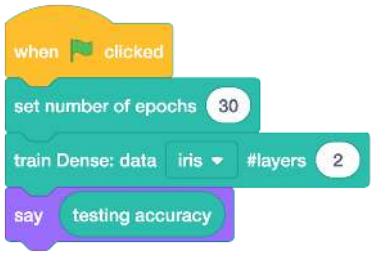

Fig. 2. Training Iris Data using Dense Model

After training the model, the user can use *predict* block to classify new data. There are different predict blocks to cover the variety of datasets that have been trained. The user needs to enter new data, and based on the input data, he/she will pick the suitable predict block. Fig. 3 illustrates an example of using *predict iris* block for the Iris data. The result of this prediction and the training performance is presented in Fig. 4. Fig. 5 illustrates all the logically possible combinations of the blocks.

The blocks used for training and prediction are implemented using the *Stack* blocks from Scratch. To present to the user how the model optimizes during training; *Reporter* blocks were used to display the training accuracy, testing accuracy, and loss values during training (Fig. 4).

Fig. 3. Predicting Iris Data

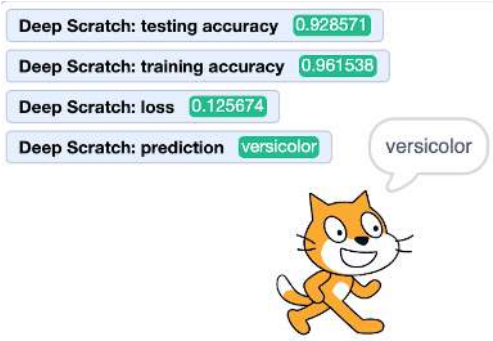

Fig. 4. Result of Training a Neural Network in DeepScratch

[2]https://github.com/Noufst/DeepScratch
[3]https://archive.ics.uci.edu/ml/datasets/Iris
[4]http://yann.lecun.com/exdb/mnist/

TABLE I. VOCABULARY OF DEEPSCRATCH (STACK BLOCKS)

ID	Block	Description	Parameters		Pre-condition
			Name	Data Type	
S1	set number of epochs 10	Change the number of epochs	N/A		None
S2	train Dense: data iris #layers 2	Train Dense neural network	Dataset #Layers	Drop down list Number	None
S3	train RNN: data iris	Train RNN neural network	Dataset	Drop down list	None
S4	train CNN: data MNIST batch size 320	Train CNN neural network	Dataset Batch size	Drop down list Number	None
S5	predict iris: sepal length 0 width 0 petal length 0 width 0	Predict new Iris data.	Sepal length Sepal width Petal length Petal width	Number Number Number Number	Should be stacked anywhere under Train Dense block (2) with Iris data selected
S6	predict hand-written digit from MNIST	Predict new hand written image.	N/A		Should be stacked under the Camera block (S8) with On selected.
S7	detect objects	Detect objects from the current video frame.	N/A		Should be stacked under the Camera block (S8) with On selected.
S8	turn video off	Turn on/off the camera	On/Off	Drop down list	

TABLE II. VOCABULARY OF DEEPSCRATCH (REPORTER BLOCKS)

ID	Block	Description
R1	training accuracy	Holds the value of the accuracy during training. The value gets updated after each epoch.
R2	loss	Holds the value of the loss during training. The value gets updated after each epoch.
R3	testing accuracy	Holds the value of the accuracy on the testing data. The value is set after the model training is complete.
R4	prediction	Holds the prediction value after running one of the prediction blocks (S5 or S6).
R5	object name 1	Holds the name of the detected object after loading the pre-trained model (S7) (the user can choose between three detected objects using the drop-down list if they exist).
R6	object correctness % 1	Holds the confidence percentage of the detected object (the user can choose between three detected objects using the drop-down list if they exist).
R7	object x position 1	Holds the x position for the detected object in the video frame (the user can choose between three detected objects using the drop-down list if they exist).
R8	object y position 1	Holds the y position for the detected object in the video frame (the user can choose between three detected objects using the drop-down list if they exist).

B. Pre-Trained Deep Learning Model Blocks

A pre-trained model is a model that is already trained on a large dataset, and can be used directly to predict new examples. Providing pre-trained model blocks in DeepScratch should be very useful for kids, as they will be able to build deep learning applications with a visual programming language. TensorFlow from google offers many pre-trained models; our extension will provide the object detection model, that localize and identify multiple objects from an image or a video.

In order to use the object detection model within DeepScratch, the user should drag the *detect objects* block and give permission to enable the video camera. Once the user turn the video camera on, it will be used as an input for prediction based on the object detection model. *detect objects* block will capture three objects as maximum from the current frame in the video. The detected objects have four outputs: object class, confidence score, and their position coordinates (x,y). Each output is presented in a reporter block with a drop-down list (1, 2 or 3) that represent the number of the detected object. The *object class* block recognizes the detected object and return its class (e.g. person, cell phone, tie, or bottle).

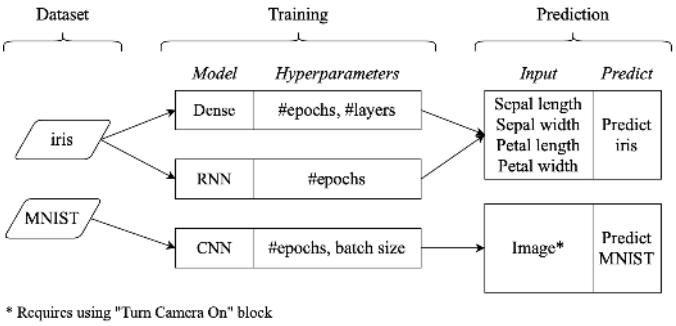

Fig. 5. Different Possible Combinations of Training and Prediction Blocks in DeepScratch

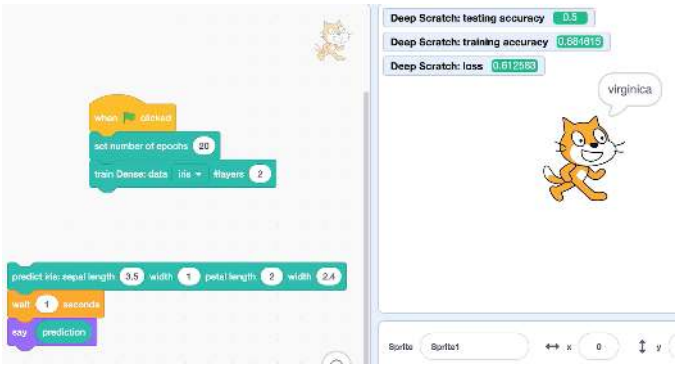

Fig. 6. Simple Program to Train and Predict Iris Data using Dense Neural Network

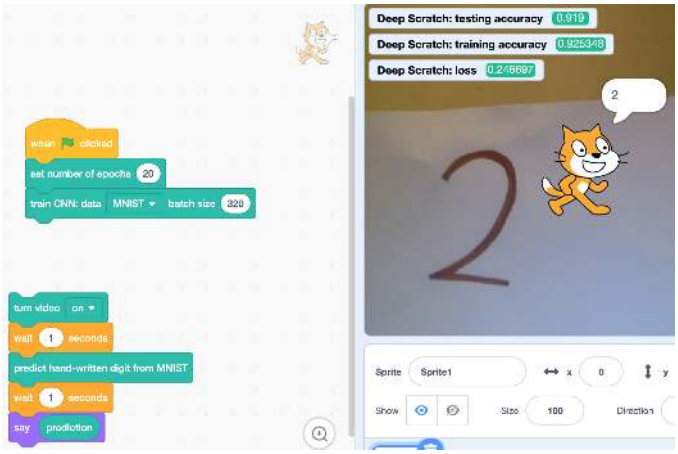

Fig. 7. Simple Program to Train and Predict Hand-Written Images using CNN Model

The confidence score indicates confidence that the object was genuinely recognised, and presented in the *object correctness* block. For the position coordinates, the video frame placed in Scratch stage deal with x and y as coordinates in Cartesian plane. The origin is the axes cross located in the center of the video frame, thus the coordinates will be in one of the four quadrants. However, the object detection model considers the upper right corner as the origin for x and y coordinates. Consequently, we convert the x and y that model returns to fit in the Cartesian plane to be returned by *X and Y positions* block, which will locate the position of the detected object in the video frame.

VII. Illustrative Examples

This section presents some examples of how DeepScratch can be used and utilized in combination with the basic Scratch blocks to build a variety of applications.

Fig. 6 demonstrates a simple program to train and predict Iris data. The left-hand side represents the program code (the blocks), where the right-hand side shows the output of the program.
The first set of blocks are used to set the number of epochs to 20, and to train a Dense neural network with two hidden layers. As the model is training, the user can observe how the accuracy and the loss values are being optimized after each epoch. Once the training is done, the accuracy of the testing data will be available to the user. The second set of blocks are used to build the prediction phase. Based on the values the user typed inside the *predict* block, the prediction (name of the iris) is presented to the user by utilizing the basic *say* block.

In Fig. 7, the user built a program that detects hand-written numbers by training the MNIST data. For the training step, a CNN model is trained with 20 epochs and a batch size of 320. Similarly, training accuracy, loss, and testing accuracy values are available to the user. To predict a new image, the user must use the *turn video* block and show a picture of a hand-written number to the camera. After that, *predict* block captures the image, converts it to a tensor, and gives it to the model to generate the prediction to be displayed to the user.

The last example illustrates how a pre-trained model can be used to build a deep learning application. The program in Fig. 8 works as follows: First, the camera is turned on. Then, the model is loaded and a capture of the camera screen is passed to the model to detect objects. After that the sprite (the cat character) will *move* to the coordinates of the first detected object and *say* its name. At last, after 5 seconds, the sprite will do the same for the second detected object. Fig. 9 demonstrates the output of the first and second detected objects.

VIII. Discussion and Preliminary Evaluation

In this section, we discuss how DeepScratch will contribute to educating kids about machine learning principles compared to the similar work highlighted in Section III. Additionally, we present the evaluation process design and the preliminary results.

Scratch Community Blocks [11] and *DataSnap* [12] aim to develop the programming and the analytical skills in preference to teaching kids the principles and the concepts of machine learning. *Machine Learning for Kids* [13], Kahn's and Winters's project [14], and *Teachable Machine* [15] are developed to teach kids how machine learning models work in abstract level. They provide a high-level overview of the machine learning pipelines. In contrast, **DeepScrach** is

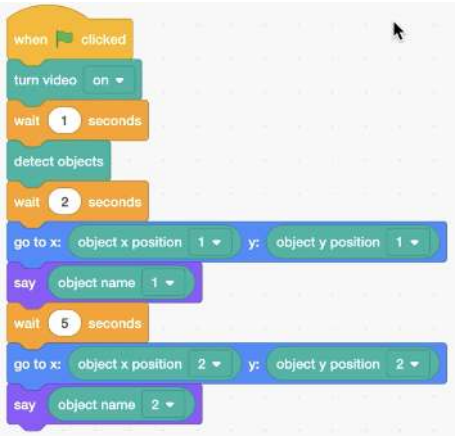

Fig. 8. Simple Program to Detect Object using the Pre-Trained Object Detection Model

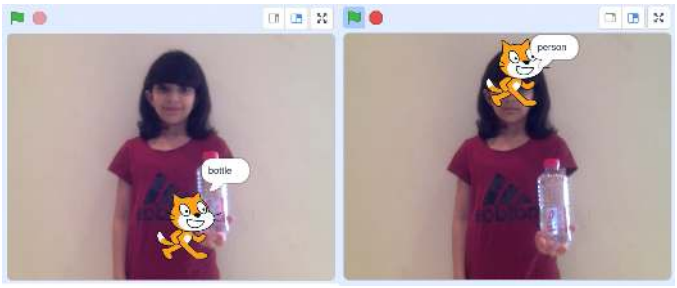

Fig. 9. Output of the Pre-Trained Object Detection Model

developed to promote teaching kids more technical skills that aim to pave the way for them to engage in more sophisticated concepts and models. The skills supported by DeepScratch are: learning about different deep learning architectures, how a model's performance is evaluated, and how to tune a model to produce a better performing model.

In order to evaluate DeepScratch extension, we conducted a full system functionality testing. Functionality testing is used to ensure that the software meets its specifications and is ready to be published. In addition, we conducted a usability study with a focus group of 5 students to validate the extension's ease of use. The plan was to conduct the user study in a school with more than 30 students; however, this could not be achieved as this research conducted during the COVID-19 curfew.

A. Functionality Testing

The functionality testing was conducted by constructing a test case for each vocabulary in DeepScratch. Each test case is described by input, pre-requisite, expected output, and actual output. If the actual output matches the expected output, the test case is considered as passed. In total, we constructed 16 test cases, all of them passed with the expected results.

B. Usability Testing

This section describes how the usability testing was designed, how the metrics were selected and measured, and the results of the usability evaluation process.

1) Usability Testing Design: A user study was conducted to inspect the usability of the proposed extension. As stated earlier, this research was conducted during the COVID-19 curfew which leads to a very limited number of participants. A preliminary experiment is conducted with a focus group of five students selected based on their availability. Participants were asked to perform two tasks to build \use deep learning networks.

Participants' sociodemographic characteristics (gender, age, spoken language) were recorded. Their ages ranged from 8 to 17 years (4 females, 1 male). In terms of their spoken language, all users were Arabic native speakers, 3 of them have intermediate English level and 2 are beginners. We interviewed the participants' to assess their level of familiarity with Scratch and deep learning on a 4-point scale (1 = not at all familiar, 4 = very familiar). The results show that all participants have basic experience of using Scratch and similar knowledge of deep learning concepts. Their description of "deep learning" concept was very similar, they described it as a way to imitates the way humans gain certain knowledge and to automate predictive analytic.

In order to measure the usability, a set of specific metrics should be used as a mean of the evaluation process. The International Organization for Standardization (ISO) defined usability as "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use" [19]. In other words, without measuring the *effectiveness*, *efficiency*, and *satisfaction*, usability evaluation cannot be achieved. The definitions of the usability metrics (according to ISO), and how each metric will be measured in our study are presented next.

- **Effectiveness:** Measures the completeness and accuracy of the goals achieved by the users. In this study, the effectiveness will be measured by the number of the completed tasks. Each task will be marked as: completed or not completed.

- **Efficiency**: Measures the expended resources to achieve the user's goals in relation to the level of effectiveness. Based on the goal of this extension, the temporal (time-based) efficiency will be measured, which is the effectiveness divided by the time used to complete the tasks.

- **Satisfaction**: Measures the extent to which the users are comfortable, and the positive attitudes of using the extension. As a mean to measure the satisfaction, questionnaires were provided to the users to answer subjective questions about their satisfaction with using the extension.

2) Usability Testing Results: On the day of the experiment, participants received a printed form that briefly describes DeepScratch, and explains the details of two tasks, followed

by a questionnaire to evaluate their experience.

We analyzed the data for all participants in accomplishing each task and answering the questionnaire. The three metrics were used as means of the evaluation process (effectiveness, efficiency, and satisfaction). For the effectiveness, the total number of completed tasks is divided by the total number of undertaken tasks. All the users completed the two tasks successfully in our experiment (i.e. the completion rate is 100%).

$$\text{Effectiveness} = \frac{\text{Number of completed tasks}}{\text{Total number of undertaken tasks}} \times 100\% \quad (1)$$

In order to examine the efficiency, we considered the *time-based efficiency*, which is the user effectiveness divided by the user time spent. Since our completion rate is 100%, all the user time spent will be considered in efficiency calculations, as shown in Table III. The average time that the user spends on both tasks is 6.18 minutes.
To calculate the time-based efficiency we applied the following equation:

$$\frac{\sum_{i=1}^{T}\sum_{j=1}^{U}\frac{u_{i_j}}{t_{i_j}}}{UT} \quad (2)$$

Where U equal to the number of users and T is the total number of tasks. Thus, Uij represents the result of task i by user j (Uij=1 when the user successfully completes the task, Uij=0 otherwise). Tij is the time spent by user j to complete task i. By applying the above equation, the overall efficiency is 0.37 tasks/minute. This infers that the DeepScratch is efficient and supports users in achieving their goals and tasks in minimal time. Furthermore, we can infer that implementing deep learning models using DeepScratch takes a considerable less time than implementing the same models with textual programming.

TABLE III. THE TIME SPENT BY USER IN EACH TASK IN MINUTES

Uesr	Task1	Task2
user 1	2.2	2.0
user 2	3.1	1.8
user 3	4.3	2.6
user 4	5.6	4.0
user 5	3.2	2.1

To measure the satisfaction, the questionnaire asked the participants to rate their experience with DeepScratch, in terms of usefulness and ease of use, along with their perceived knowledge of deep learning concepts. All participants reported that the applications were very easy to create, and supplemented their understanding of deep learning models. However, three students raised the need of translating the blocks into Arabic Language.

IX. CONCLUSION

In this paper, we presented the motivation and the design of DeepScratch; an extension to Scratch programming language. We explained how the system is designed to allow kids to engage with the concepts of deep learning -normally practiced only by expertise- in a simple and attractive way. We built this programming language extension in order to satisfy our goal of promoting new pathways to understand deep learning concepts and applications. This paper presented the implementation process of DeepScratch, and explained the functionality of the extended vocabulary. DeepScratch provided two options to implement deep learning models: training a neural network based on built-in datasets, and using pre-trained deep learning models. The two options are provided to serve different age groups and educational levels.

To inspect the functionality and the usability of the proposed extension, we conducted a full system functionality testing and a user study. Both methods revealed the effectiveness of the proposed extension, and the participants reacted positively to the experiment of DeepScratch. As this was a preliminary user study, the obtained results were encouraging. However, for more concrete results, it will be necessary to expand the experiment with a larger group of kids. Furthermore, future studies can extend DeepScratch to include more neural network architectures and different applications.

REFERENCES

[1] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond, "The scratch programming language and environment," *ACM Transactions on Computing Education*, vol. 10, no. 4, pp. 1–15, 2010.

[2] S. Dasgupta, S. M. Clements, A. Y. Idlbi, C. Willis-Ford, and M. Resnick, "Extending Scratch: New pathways into programming," *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC*, vol. 2015-Decem, pp. 165–169, 2015.

[3] C. Kelleher and R. Pausch, "Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers," 2005.

[4] S. Y. Lye and J. H. L. Koh, "Review on teaching and learning of computational thinking through programming: What is next for K-12?" *Computers in Human Behavior*, vol. 41, pp. 51–61, 2014. [Online]. Available: http://dx.doi.org/10.1016/j.chb.2014.09.012

[5] A. Repenning and T. Sumner, "Agentsheets: A Medium for Creating Domain-Oriented Visual Languages," *Computer*, vol. 28, no. 3, pp. 17–25, 1995.

[6] A. Repenning, "Moving Beyond Syntax: Lessons from 20 Years of Blocks Programing in AgentSheets," *Journal of Visual Languages and Sentient Systems*, vol. 3, no. 1, pp. 68–91, 2017.

[7] J. Maloney, N. Rusk, L. Burd, B. Silverman, Y. Kafai, and M. Resnick, "Scratch: A sneak preview," *Proceedings - Second International Conference on Creating, Connecting and Collaborating Through Computing*, pp. 104–109, 2004.

[8] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai, "Scratch: Programming for All," *Communications of the ACM*, vol. 52, no. 11, pp. 60–67, 2009. [Online]. Available: http://scratch.mit.edu

[9] Scratch, "Scratch for Developers," 2019. [Online]. Available: https://scratch.mit.edu/developers

[10] L. Deng and D. Yu, "Deep learning: Methods and applications," *Foundations and Trends in Signal Processing*, vol. 7, no. 3-4, pp. 197–387, 2013.

[11] S. Dasgupta and B. M. Hill, "Scratch community blocks: Supporting Children as data scientists," *Conference on Human Factors in Computing Systems - Proceedings*, vol. 2017-May, pp. 3620–3631, 2017.

[12] J. D. Hellmann, "DataSnap: Enabling Domain Experts and Introductory Programmers to Process Big Data in a Block-Based Programming Language," 2015. [Online]. Available: https://vtechworks.lib.vt.edu/handle/10919/54544

[13] D. Lane, "Machine Learning for Kids." [Online]. Available: https://machinelearningforkids.co.uk

[14] K. Kahn and N. Winters, "AI Programming by Children," *Proceedings of the Constructionism 2018 Conference*, pp. 315–324, 2018.

[15] Google, "Teachable Machine," 2019. [Online]. Available: https://teachablemachine.withgoogle.com

[16] I. Sommerville, *Software Engineering*, 2013.

[17] E. Pasternak, R. Fenichel, and A. N. Marshall, "Tips for creating a block language with blockly," *Proceedings - 2017 IEEE Blocks and Beyond Workshop, B and B 2017*, vol. 2017-Novem, pp. 21–24, 2017.

[18] Scratch Wiki, "Scratch blocks," Feb 2020. [Online]. Available: https://en.scratch-wiki.info/wiki/Blocks

[19] "Ergonomic requirements for office work with visual display terminals (VDTs) — Part 11: Guidance on usability," International Organization for Standardization, Geneva, Switzerland, Standard, Mar. 1998.