INTRODUCTION TO MACHINE LEARNING

Image Classification

Table of Contents

1. Synopsis

Students will learn about the basics of machine learning and create their own apps that implement these concepts through image classification. The students will take photos with their smartphone or tablet cameras and the apps will identify objects in those photos. Each classification comes with a confidence level, a value of how confident the app is with its classification. Students will use MIT App Inventor's machine learning extension called the LookExtension when creating this app.

2. Learning Objectives

After completing this unit, students will be able to:

1. Use the LookExtension to create an image classification app using App Inventor.
2. Demonstrate a basic understanding of machine learning.

3. Mapping with the CSTA Standards

This table shows the alignment of this unit with the intended learning outcomes to the CSTA CS Standards.

2-DA-08	Collect data using computational tools and transform the data to make it more useful and reliable. [C] DA: Collection; Visualization & Transformation [P] Testing (6.3)	App takes images and converts objects in them into classifications.
2-AP-13	Decompose problems and subproblems into parts to facilitate the design, implementation, and review of programs.	App creation is split into parts: classification function, camera toggling, and error handling.

	[C] AP: Modularity [P] Computational Problems (3.2)	
2-AP-17	Incorporate existing code, media, and libraries into original programs, and give attribution. [C] AP: Program Development [P] Abstraction (4.2), Creating (5.2), Communicating (7.3)	Students use the MIT App Inventor LookExtension library.
2-IC-23	Describe tradeoffs between allowing information to be public and keeping information private and secure. [C] IC: Safety, Law, & Ethics [P] Communicating (7.2)	Students will discuss the extent to which private data can be used and the consent involved and needed to implement any form of machine learning on said data.

4. Learning Prerequisites

Students should have had experience with a block-based language like Scratch or MIT App Inventor, and understand computing concepts like sequences and event handling.

Students should also know the fundamental blocks for basic MIT App Inventor components such as labels, buttons, texts, lists, etc.

5. Hardware Prerequisites and Setup

Important Note: Though you won't be using the Whatisit app until Lesson 2, it is important to test whether the app will work at all beforehand. Follow the instructions below under Lesson 2 to test whether the Whatisit App will work on your classroom mobile devices.

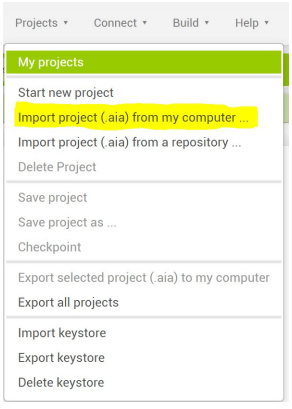
Lesson 1

For the first lesson, each student will need individual access to a computer or laptop that is connected to the internet. They will need to be able to run Teachable Machine on it. Note that this first lesson needs neither App Inventor nor mobile devices.

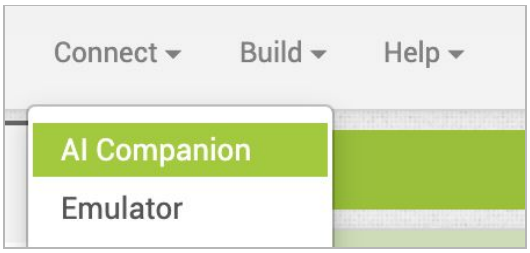
Lesson 2

For the second lesson, each student will need individual access to a computer or laptop that is connected to the internet. They will also need Android mobile devices (smartphones or tablets) that have cameras and can run App Inventor's Look Extension (noted in previous section). Make sure that the mobile devices have the latest updates and have the MIT AI2 Companion installed so that students can run the app on the device (Google Play MIT AI2 Companion link).

In order to check beforehand if the Whatisit app will work on your device, we've provided a completed version of the project that the students will be building. To test it, start by downloading the completed project (link) onto your computer. Then, follow this link to App Inventor and import the completed app you just downloaded by clicking on "Projects" and then "Import project (.aia) from my computer" as shown below.



To connect your app to a mobile device, keep the imported project open and click on “Connect” and then “AI Companion” as shown below.



Open the MIT AI2 Companion on your mobile device and connect either with a code or by scanning. The app should show up on your mobile device. Wait for the button on the screen to switch from “Waiting…” to “Ready” and the app should be set up. If *Ready* does not appear when you start the app, or you don’t see the camera image, try exiting the app and restarting it. If this does not work after a few tries, the app simply might not work on the device, due to hardware limitations. If that is the case, it won’t be worth doing the unit with students. If the app does work, you can try classifying various objects around you by pressing the buttons to get an idea of what the app can do.

6. Lesson Plan (2 x 45 minutes)

This unit consists of two 45-minute lessons. Detailed teacher guides are located in the appendix.

Lesson 1

Time	Activity
10 min	**Introduction to Unit** Discuss what machine learning is and how it is used.
25 min	**Play with Teachable Machine** Students go onto the web and use Google's Teachable Machine demo to get a basic understanding of how machine learning works.
10 min	**Wrap-up Discussion** Discuss how data is collected and the extent to which information can be used and thoughts on machine learning.

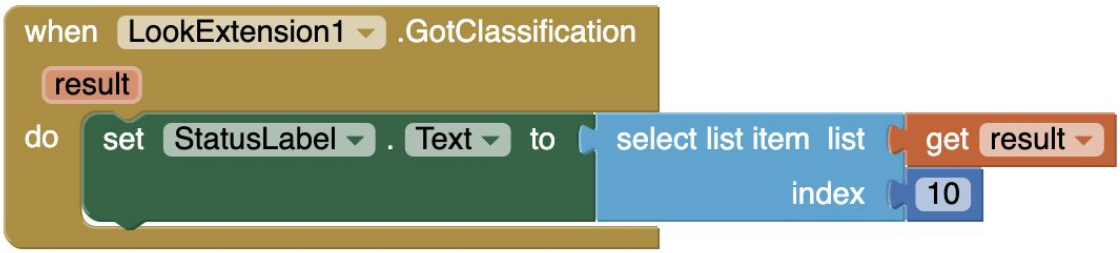
Lesson 2

Time	Activity
5 min	**Introduction to Activity** Introduce the WhatisitApp and compare it to Teachable Machine.
20 min	**Coding of Whatisit App** 1. Download and import the Whatisit Template in App Inventor. 2. Students work to finish creating an image classifier app. 3. They may use either the built in sidebar tutorial or a pdf tutorial.
10 min	**Testing the Whatisit App** 1. Students run their completed app on their tablets. 2. Students experiment with the app's benefits and limitations.
10 min	**Wrap-up Discussion** Discuss how their app worked based on the tests, in terms of advantages, limitations, and suggest ways to improve it.

7. Assessment

Multiple-choice questions to test understanding after the unit

1. A student changes the index value in the code below from 1 to 10; what is displayed on the screen in the StatusLabel?



A. It displays the tenth best classification.

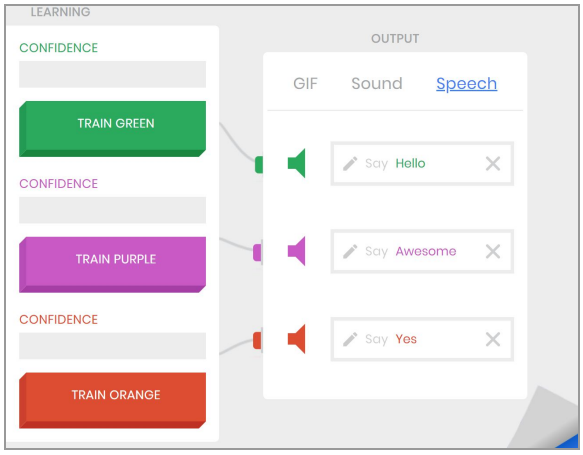
B. It loops around and displays the best classification.

C. It displays an error message.

D. Nothing happens because index value doesn't matter.

(Answer: **A**; remember that result will be an ordered list of the ten best classification. By changing the index value to ten, StatusLabel will display the tenth item in the list.)

2. Think back to when you used Teachable Machine. Suppose you trained green with images of a clock, purple with images of a phone, and orange with images of a mug. You then test the machine with an image of a phone. What would you expect to happen? Hint: Use the image below.



A. It plays the word “Hello”.

B. It plays the word “Awesome”.

C. It plays the word “Yes”.

D. Nothing happens because the image couldn’t be identified.

(Answer: **B**; Purple is trained on phones and you showed the machine a phone to test. You would expect it to recognize the phone and highlight purple, which then plays the word “Awesome”.)

3. Is it possible for your Whatisit app to correctly classify any object? Why or why not?

A. Yes, it collects data from other objects and compiles it to create a new classification of any new object.
B. Yes, it uses a complex function to classify what any object is.
C. No, it cannot as it is limited by the database it has.
D. Not, it cannot as it can only identify a few objects due to hardware limitations of phones.

(Answer: **C**; the extension LookExtension, used in the application can only classify the 999 different objects stored in its database. If one would like to classify more objects, more data/images are needed.)

Survey of learning attitudes

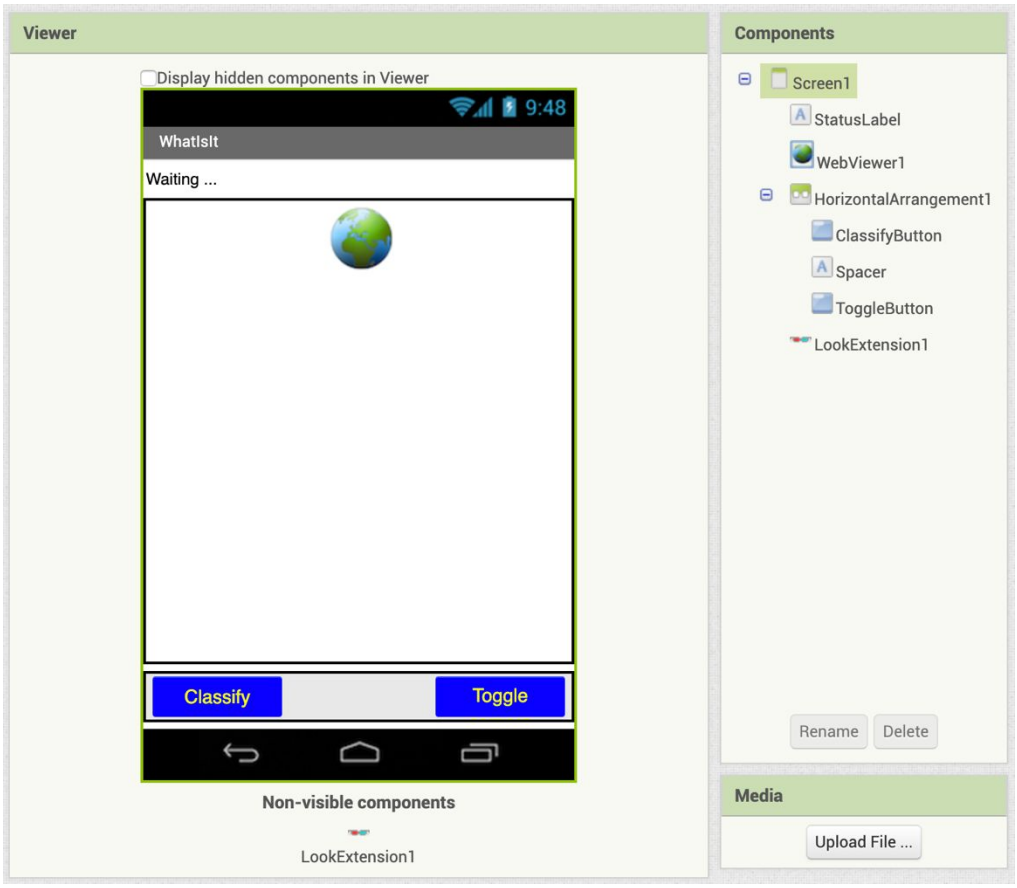
In order to evaluate students' attitude, perception, and understanding towards coding, students are required to finish a 5-point scale survey below by putting a "✓" in the appropriate box.

After completion of this unit, I think…	**Disagree**	**Somewhat disagree**	**Neutral**	**Somewhat agree**	**Agree**
Learning how to make apps makes me want to learn more about coding.					
I feel more connected to the technology around me when I make apps.					
I am excited to share this app with friends and family.					

8. Screen Design and Code Blocks

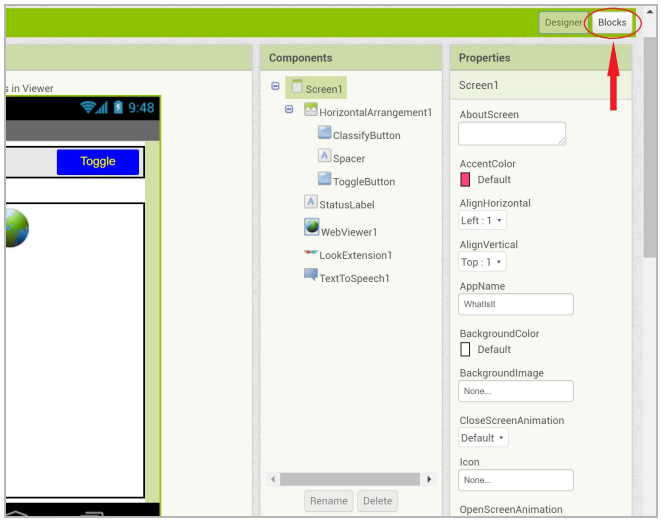
Below are images of the completed Whatisit App for reference if students are having trouble with Lesson 2.

Designer:

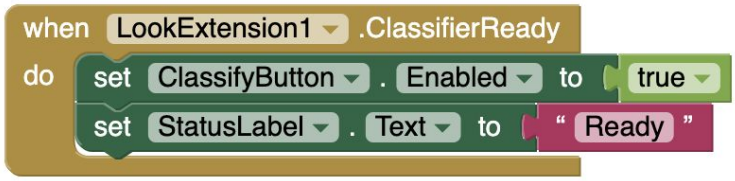


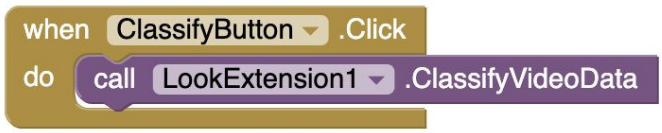
Blocks:

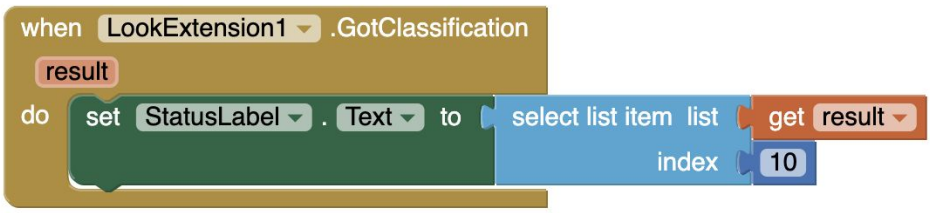
To get to the Blocks Editor, click on the upper right hand button as shown below:

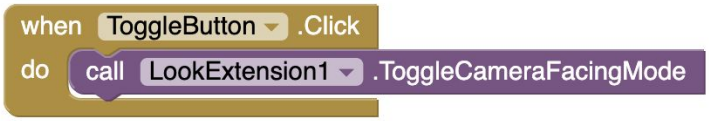


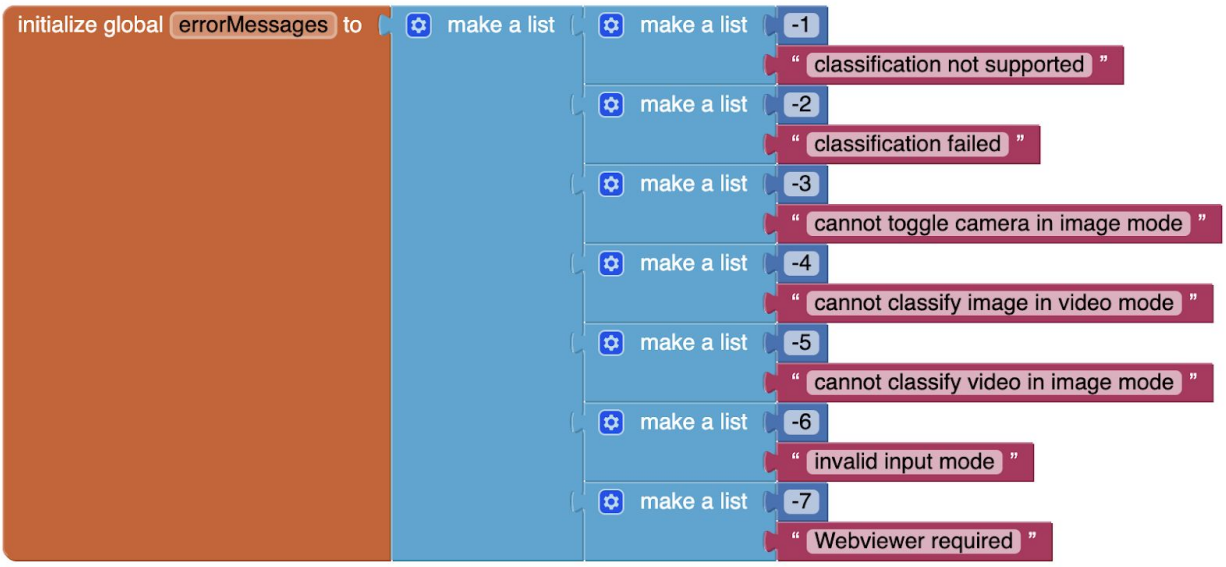
All completed blocks for the Whatisit App:

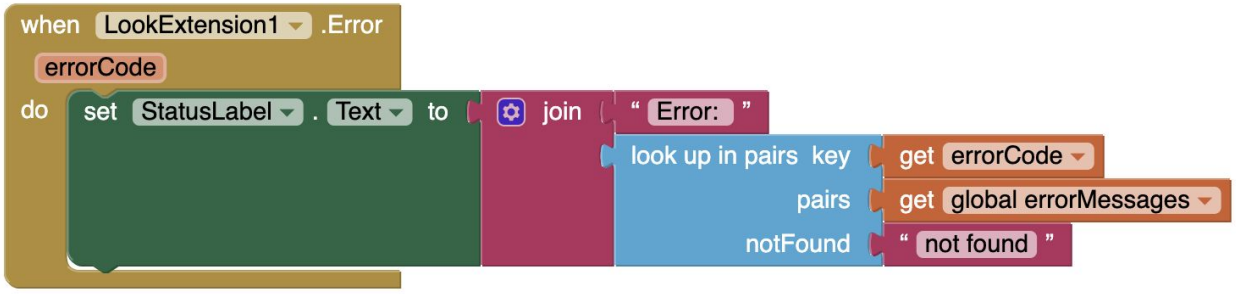












MIT APP INVENTOR

Appendix 1

Teacher's Guide

Lesson 1

Learning Objectives

At the end of the lesson, students should be able to:

1. Gain a basic understanding of what machine learning is.
2. Gain practical knowledge of image recognition using a demo.

Lesson 1 Outline

Introduction to Machine Learning (10 minutes)

Using the presentation, give a general overview of what machine learning is. Additional explanations and resource links are in Appendix 3.

Play with Teachable Machine (25 minutes)

Have your students load https://teachablemachine.withgoogle.com/ in a browser on their computers. Then, have them go through the guided audio tutorial and play around with it. The website in itself gives a simple understanding of how machine learning works.

Wrap-up Discussion (10 minutes):

Once students have fully tested out Teachable Machine, lead the class in a discussion of the lesson. Below are some possible questions to begin:

1. We introduced machine learning at the beginning of class. Now that you've tested a machine learning model, how has your understanding of what machine learning is changed?
2. Can you think of any examples of machine learning you have seen in the world?
3. What did you learn from trying out Teachable Machine on your own?
4. Were there any issues with trying to get Teachable Machine to recognize images? Why was that? Did anything you tried improve it?
5. How could image recognition be useful in your life?

Appendix 2

Teacher's Guide

Lesson 2

Learning Objectives

At the end of the lesson, students should be able to:

1. Gain an understanding of the potential and limits of machine learning image recognition.
2. Build their own machine learning app and test its capabilities.

Lesson 2 Outline

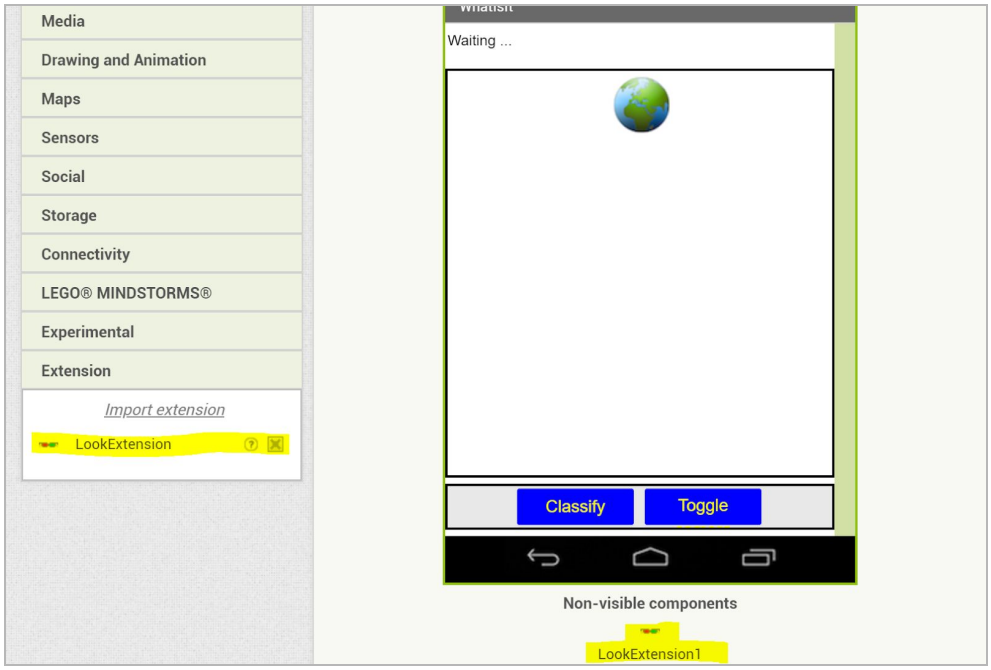
Introduction to the Whatisit App (5 minutes)

1. Introduce to students that they will now be exploring and then making their own image recognition project (presentation link, note the speaker notes for some of the slides). Compare the Whatisit app to the exploration they did with Teachable Machine in the previous lesson.
2. Note that the main difference between Teachable Machine and Whatisit is that in Teachable Machine, students do training for three classes. For Whatisit, the neural network has already been trained for 999 different classes.

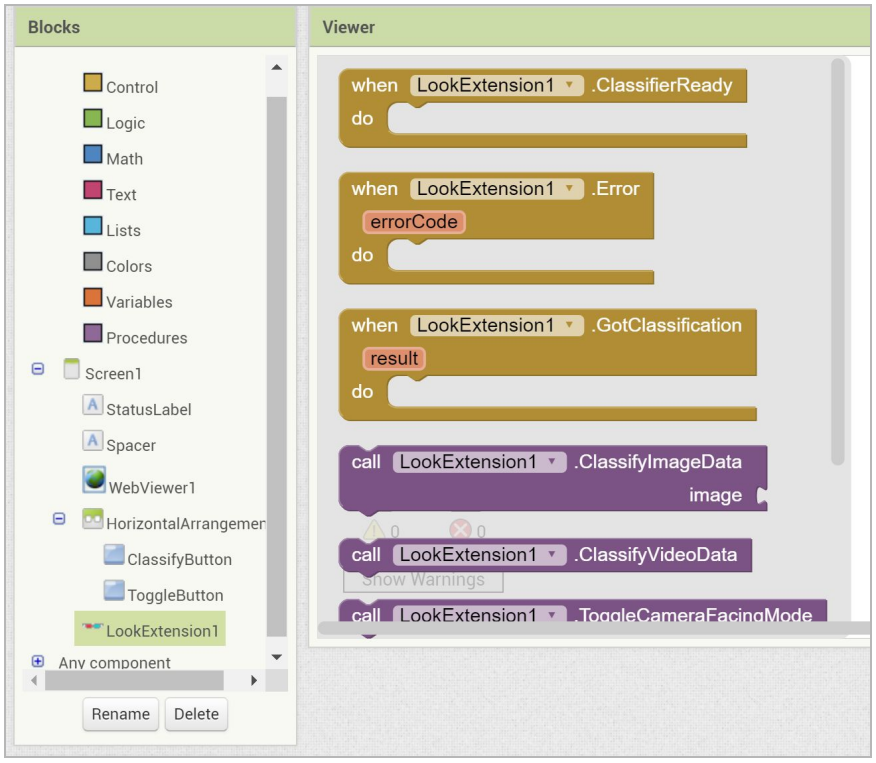
Setup and Code Whatisit App (20 minutes):

LookExtension:

1.) LookExtension is an extension to MIT App Inventor that helps run image classification on the app. An extension is a set of rules and blocks that are not part of core MIT App Inventor, but can be imported from an outside source. The Look Extension will have already been imported to the Whatisit app template your students will complete.



Above: The Look Extension imported into the Whatisit project.



Above: New blocks added by LookExtension to the blocks editor.

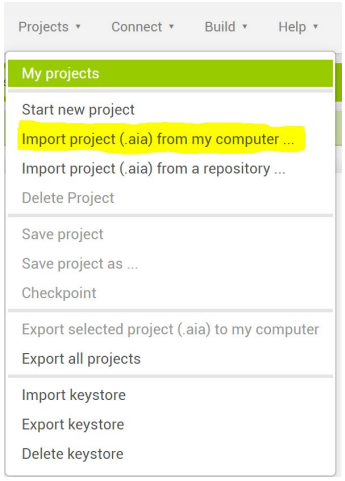
2.) **Make sure to review the student guide and/or the sidebar tutorial beforehand to help out students as best as possible!**

Setup:

Students have two methods of completing the tutorial. Option 1 is a sidebar tutorial where the tutorial appears in a sidebar directly in the App Inventor development environment in their browser. Option 2 is a pdf text document students can access online or teachers can print out for students. If choosing Option 1 for students, they simply have to access this URL (http://bit.ly/2OfPMOr) and App Inventor will open the template project with attached tutorial.

Alternatively, if you choose Option 2, students will have to import the template to their computers. Instructions for how to do this follow:

1. Have students download the Whatisit template on their computers. Make sure they know where it was downloaded to because it will need to be uploaded in the next step.
2. Have students open up MIT AppInventor (http://ai2.appinventor.mit.edu) and log in.
3. Students can now import the Whatisit template by clicking "Projects" at the top of the main screen and then "Import project (.aia) from my computer" as shown below. Open the project by clicking on it.



Coding of App:

1. Students code the app using one of the two options above.
2. Refer to the blocks pictures above if students have any questions about specific steps.

Testing the Whatisit App (10 minutes)

1. Have the students follow the instructions in their tutorials and connect their app to the smartphone or tablet. Reference the official guide

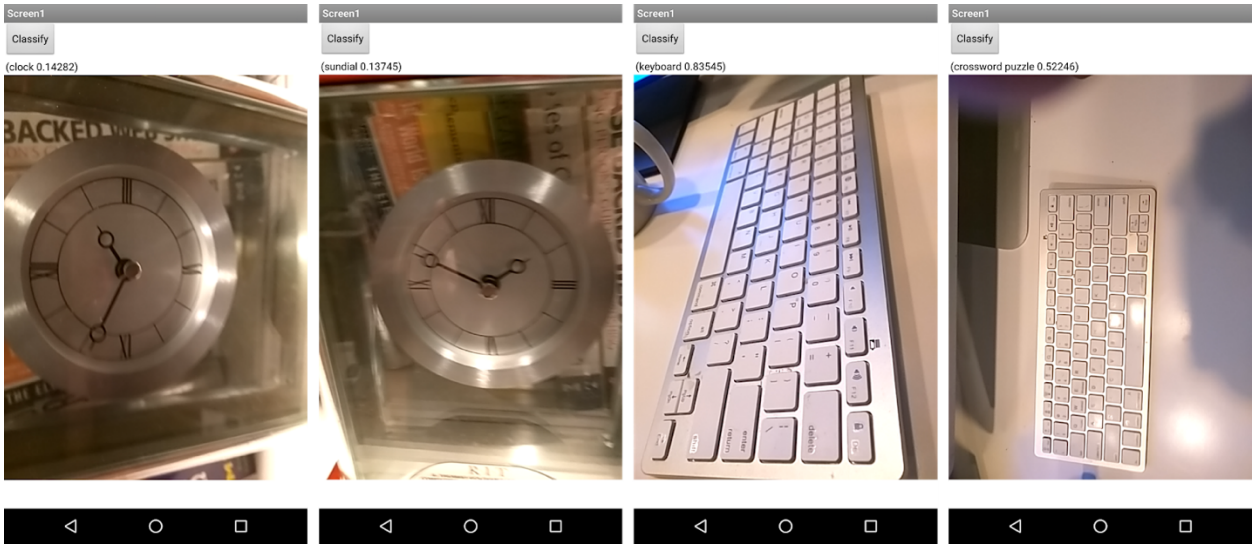
(https://appinventor.mit.edu/explore/ai2/setup.html) for more information on how to set up their computers and mobile devices.

2. Your students can now experiment with the app. On the main screen, you will see a button marked *Classify* and under it a message that initially reads "*Waiting* ...." After a second or two, the message will change to "*Ready*" and the screen area below the message will show the scene in the phone's camera.

How to test:

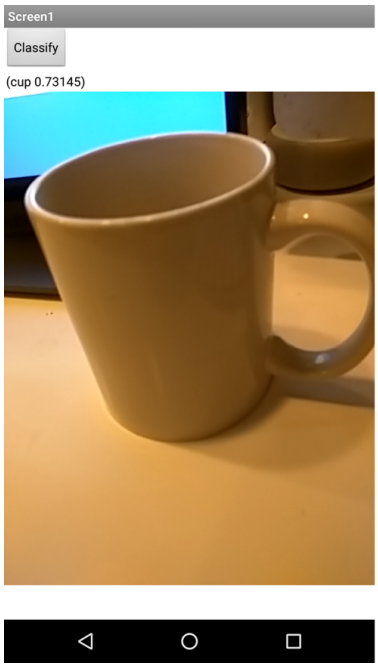
Point the camera at something and press *Classify*. The app should show its identification of the object. Don't be surprised if identification is wrong: it usually will be wrong. Together with the class will be a *confidence level* between 0 and 1 that indicates how confident the app is with the result.

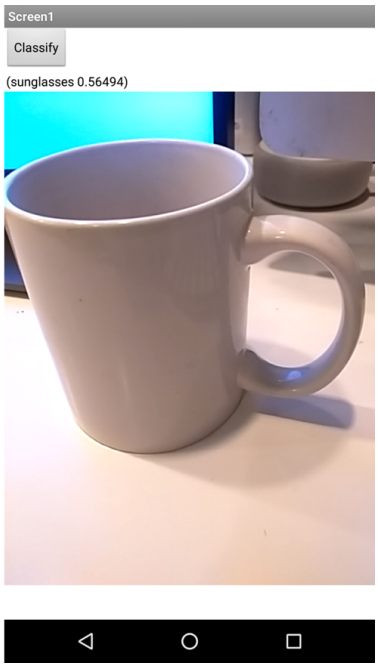
Students can learn a lot about computer vision by trying this app for a few minutes, pointing the camera at various scenes and examining the *Classify*. Here are some examples of the kinds of things you might notice as you explore:



clock 0.14282	sundial 0.13745	keyboard 0.83545	crossword puzzle 0.522446

The app classifies the clock on the left as a clock, but if you change the angle just a bit, the result is a sundial. Similarly, the keyboard on the left is seen as a keyboard, but rotate the image and it's seen a crossword puzzle. As you can see, the image identification is sensitive to details of the image, but these mistakes seem understandable, given that the two images in each case look similar to us.





cup 0.73145	sunglasses 0.56494

Other results are mysterious, such as where the cup on the left, with a slight change of angle and background, is classified incorrectly as sunglasses. That's a mistake no person would make and it seems hard to understand why the machine fails in this way. In fact, explaining why *machine learning* programs give the results they do is challenging even for experts, and it's an active area of current research in machine learning.

Ask students to try the following:

1. Hold up an object, like a notebook, in front of the camera and have the app classify it.
2. Take the same object and change the angle. Reclassify. Does it classify it as the same thing?
3. Rotate it or move the camera farther away and reclassify. Does it classify it as the same thing?

Spend some time experimenting with the WhatIsIt app. Try to find examples where it does an OK job and examples where it does poorly. For instance, the app ought to be able to identify cats as cats. But it will never identify people as people. That's because the app has been constructed using an AI system (*neural network)* called *Mobilenet*, which was built to recognize only 999 different classes of objects, and does not include any images of people.

Go around and discuss the findings of your students. While the confidences are continuous, generally a confidence above 0.65 is "good".

Wrap-up Discussion (10 minutes):

Once students have completed the app and tested it, lead the class in a discussion of the lesson. Below are some possible questions to begin:

a. Now that you've made the app, in what ways did Whatisit feel similar to Teachable Machine? In what ways did it feel different?
b. Were there any images that the app was very good or very bad at classifying? Why do you think that was?
c. How did it feel to code the app yourself? Was it easier or harder than you expected?

d. Now that you've seen two examples of image recognition , what other ways could you use it in your life? Do you have any ideas for apps you could build that use image recognition ?

e. mage recognition can be very powerful. Can you list some privacy concerns that might come up if devices got really good at recognizing images?

Appendix 3

Machine Learning Resources

What is Machine Learning?

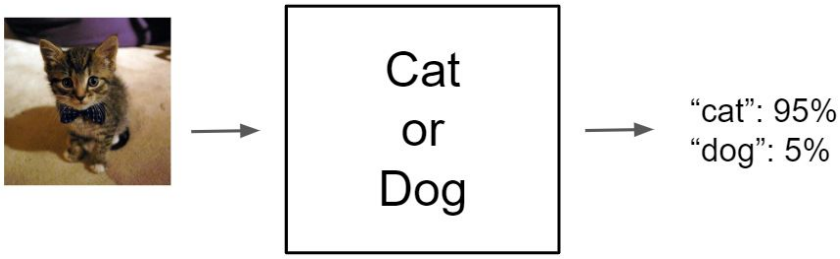
Artificial Intelligence (AI) and **Machine Learning** (ML) are fields of computation used to describe machines or systems that simulate human methods of cognition, such as learning or problem-solving. Examples of machine learning include Google's search engine, social media content suggestions, and the current development of self-driving cars.

How does machine learning and image classification work?

Image classification is a very important component of machine learning. For example, a self-driving car would have to instantly classify any images it sees, and the difference between a pedestrian and a green traffic light is critical.

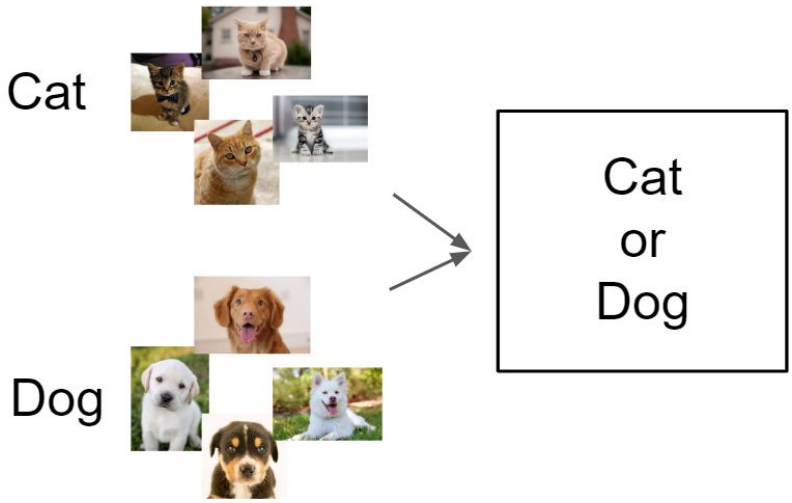
LookExtension is an example of a **machine learning** tool that implements image classification. One popular way to create machine learning programs is by ***training*** them to classify images into some number of classes.

The overall goal for the program is to receive an image as an **input** and to give some words (a classification of the image) as an **output**. Let's say you want to classify images as either cats or dogs:



A lot of computation takes place in order to get from an image to the output text, and we will only describe the general principle here.

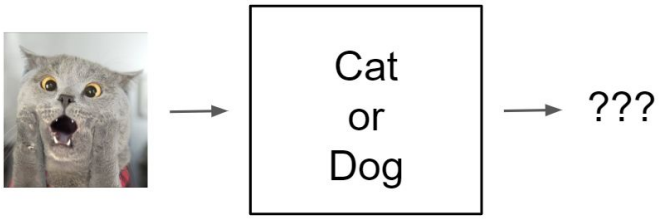
In order for the program to get started on this, you have to **train** it to classify certain images. For example, if you wanted to teach the program to tell whether an image is a cat or a dog, you would give it many images of both cats and dogs, labelled as such. It is important that you label each input image as a cat or a dog so that the program can know what kinds of images look like cats and what kinds of images look like dogs.



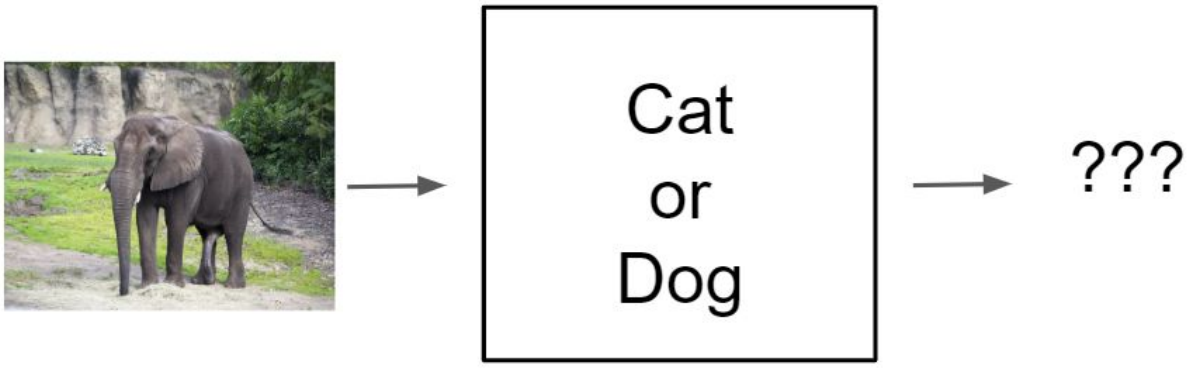
With enough examples, the program will be trained to “know” what types of images show dogs and what types of images show cats. It is important to note that this generally requires a huge

number of examples from each class. Otherwise, the program will not have enough information to reliably classify new images.

Once the program has been trained with enough images, it can be **tested** by giving it a new image that we didn't use to train it and see what the program outputs.



Based on the results, you can go back to training, and continue the training and testing process for as long as needed.



Be aware that the program will only have the chance to correctly classify the images it was trained on. Since we trained with images labelled as "cat" and "dog", any image that is tested will end up classified as a cat or a dog. Even putting in an entirely different image, such as one of

an elephant, will be classified as a cat or a dog because those are the only two **classes** of image it has been taught. A **class** is a possible classification or output. This means that the program will not be able to correctly classify an image if its class is not already in its database of trained classes.

Neural Networks

The LookExtension image classification uses a neural network to classify the input images. A neural network is effectively an interconnected series of nodes that all work together to create the output. A node is generally a series of random computations, though they can be connected or have specific computations.

When given an image, each node casts a "vote" of which of the classes the image is. This is then compared to what the image is actually labelled as. Correct nodes will be weighted more heavily in the future. For instance, given an image labelled as a "cat", the nodes that correctly labelled it as a cat will be weighted more strongly in the future, while nodes that incorrectly labelled it as a dog will be weighted less since they now have a record of classifying incorrectly.

The idea is that with enough training, the adjusted program with the adjusted weights will do a good job in classifying not only the training images but also a lot of other images that it has not seen. Whether this idea actually works depends on many details, such as how the errors are computed, how the weights are adjusted, and how well the training images represent the entire set of images to be classified. All of these issues are active areas of study in machine learning research.

Supervised versus Unsupervised Learning

The LookExtension uses **supervised learning**, which is a common way of classifying known categories. **Supervised learning** is when you have inputs (images) and you want to match them

to known outputs (the labels “cat” and “dog”). This way, during training, it is easy to tell when an image is classified correctly or incorrectly.

Unsupervised learning is when you have inputs but no known outputs and you want to use machine learning to find patterns that are not already known. This could be used in a large collection of medical data if one is trying to find patterns that humans may have missed. Generally, image recognition programs will use supervised learning since we already know what images we are looking for.

About LookExtension

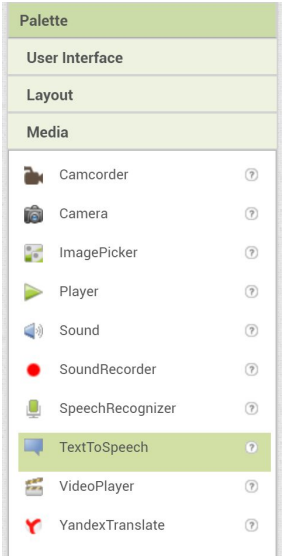
This extension uses a kind of neural network called a ***mobilenet*** that is specifically designed to work well for image classification on smartphones. The mobilenet in the Look Extension has been pre-trained to recognize 999 classes, based on training with millions of images. You can check all of the classes here, and note what labels are included. You can also use the App Inventor *LookExtension.knownClasses* block to obtain the list of classes for use in new apps. When you use the *WhatIsIt* app, you’re taking advantage of all that training. But you’re not doing any additional training yourself, so the image recognition never gets any better.

Blocks and Tips for Exploration

At the end of the Whatisit Tutorial, there is a section called “Expand your App” which offers additional exercises if students finish early. Below are hints you can provide for each proposed expansion if students need the help.

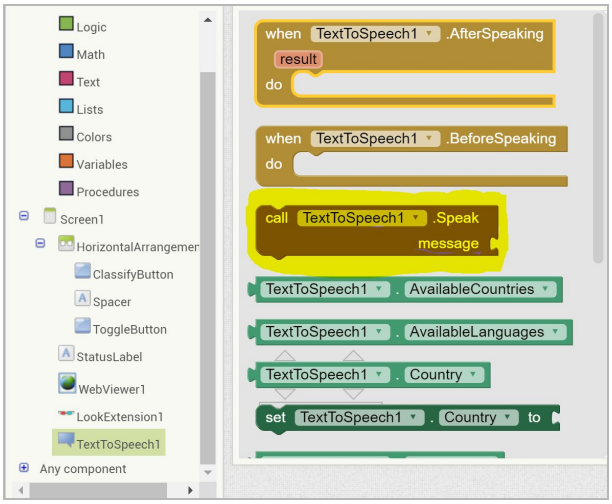
Speak the result:

In order for the result to be spoken, students will need to include a Text-to-Speech component. This component can be found in the Media section of the Palette. Just click and drag it into the Viewer to set it up. There is no need to change any of its properties.

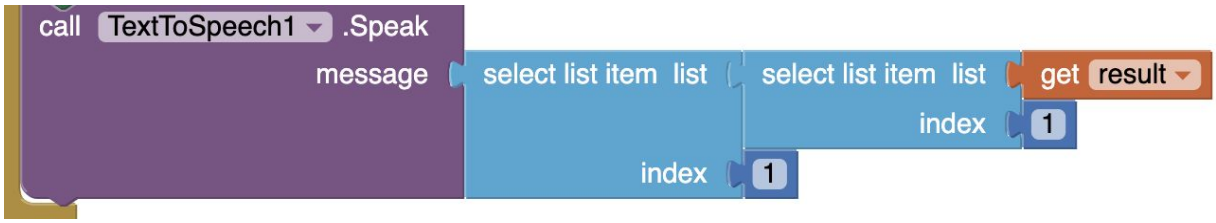


MIT APP INVENTOR

Next, in the blocks interface, the one new block that will be needed will just be the one that tells the Text-to-Speech function to speak whatever message it is given, highlighted below.

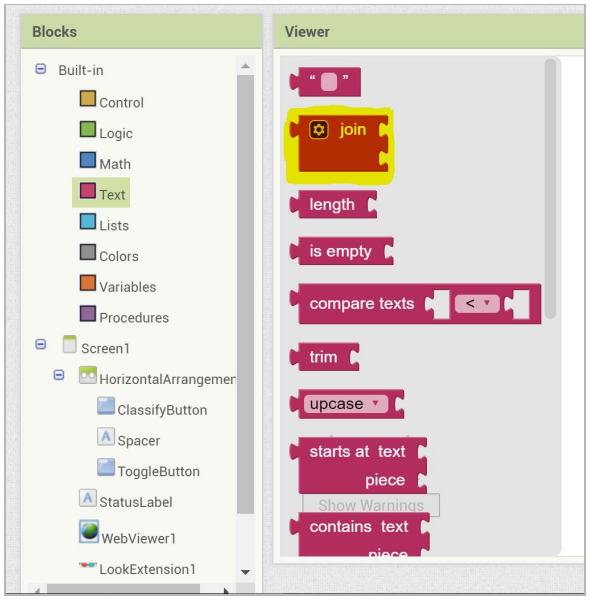


You should end up with something similar to this:

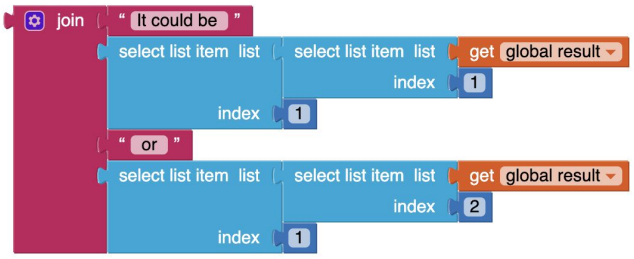


Using the confidences:

If you want the label to show multiple confidences, you may want for your students to use the join block, which joins together two different results. You can join as many results as you need to form an output to display.

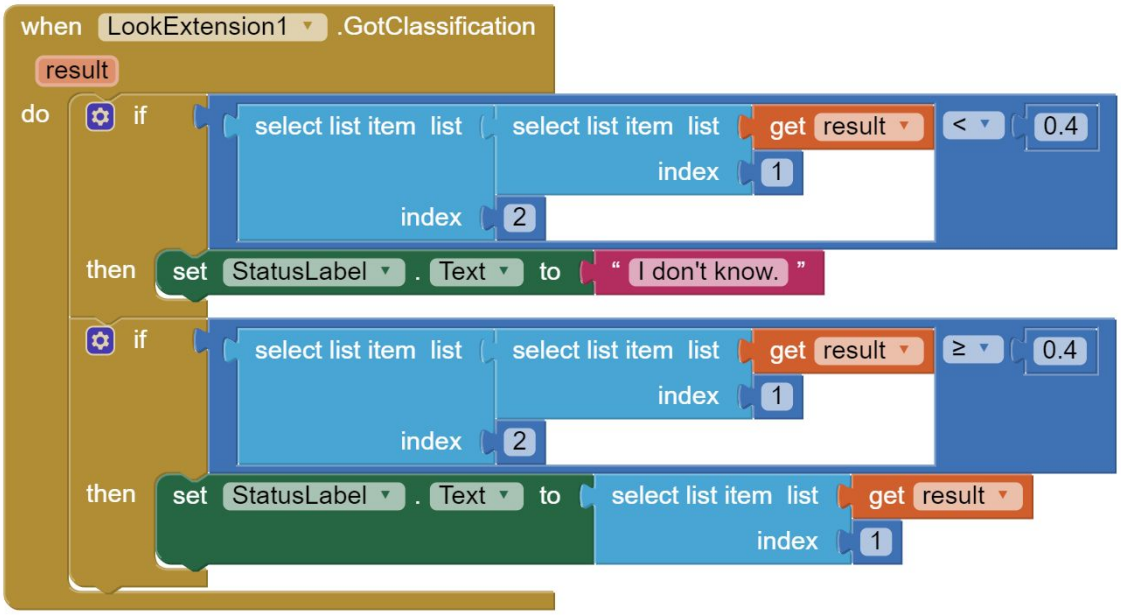


Join text strings:



If your students want to implement anything that compares results, you'll need an if/then statement. This means that IF a condition is fulfilled (ex, if the top confidence is too low) THEN it does something specific (ex, then print out "I don't know").

Here's an example. IF the first confidence in the list is less than (<) 0.4, THEN the label text will say "I don't know.". Next, IF the first confidence in the list is greater than or equal to (>=) 0.4, THEN the label text will say the object and its confidence as was described in the tutorial.



You could also have it display "I think it's (object 1), but it could be (object 2)" if the confidence values are close enough.

It was also mentioned in the unit that *LookExtension* returns the top 10 classifications, so you could also have LookExtension return the top 3 classifications (without the confidence values).

Still Image:

Another suggestion is to allow the app to process still images. To do this, you would add another Button to the app, and also the Camera component (found in the Media drawer). When the user clicks the Button, set the LookExtension's InputMode to "Image", and then call the Camera.TakePicture procedure to take a picture.

The Camera component triggers the Camera.AfterPicture event after the picture is taken. In that event, classify the image data.