An Educational Approach to Machine Learning with Mobile Applications

by

Kevin Zhu

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2019

Author . . . .
Department of Electrical Engineering and Computer Science
May 24, 2019

Certified by. . . . .
Harold Abelson
Class of 1922 Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by . . . .
Katrina LaCurts
Chairman, Department Committee on Graduate Theses

An Educational Approach to Machine Learning with Mobile Applications

by

Kevin Zhu

Submitted to the Department of Electrical Engineering and Computer Science
on May 24, 2019, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

Machine learning has increasingly become a major topic in computer science for students to learn. However, it can be quite technical and thus difficult for students to grasp, especially those in high school and under. To make machine learning and its applications more accessible to younger students, we developed a series of machine learning extensions for MIT App Inventor. MIT App Inventor is a web application for users with minimal programming experience to easily and quickly build mobile applications, and these extensions allow users to build applications that incorporate powerful machine learning functionality. These extensions were tested over a 6-week class with about 10 students and can be used as an educational tool.

Thesis Supervisor: Harold Abelson
Title: Class of 1922 Professor of Computer Science and Engineering

Acknowledgments

I would like to thank Hal Abelson for all the advice he has given to me during the development of this thesis, Evan Patton for answering many questions I had about developing for App Inventor, and Kelsey Chan for helping me develop several of the extensions and tutorials.

Contents

1 Introduction 11

2 Related Work 13

2.1 Scratch 13

2.2 Cognimates 13

2.3 Machine Learning for Kids 14

2.4 Amazon and Google services 15

3 Background 17

3.1 MIT App Inventor 17

3.2 TensorFlow.js 20

4 Implementation 23

4.1 Components vs. Extensions 23

4.2 Extensions 24

4.2.1 Object recognition 24

4.2.2 Teachable Machine 26

4.2.3 OCR 28

4.2.4 Object detection 30

4.2.5 Sentiment analysis 31

4.2.6 Text filtering 32

4.2.7 Speech processing 32

5 Applications **35**

5.1 What is it? 35

5.2 Rock–paper–scissors 35

5.3 Emoji scavenger hunt 36

5.4 Google Lens 36

5.5 NLP tutorials 37

5.6 Robot control 37

6 Evaluation **39**

6.1 Class 1 40

6.2 Class 2 40

6.3 Class 3 41

6.4 Class 4 42

6.5 Class 5 43

6.6 Class 6 43

6.7 Results 44

7 Conclusion **47**

7.1 Future Work 47

7.1.1 Other computer vision extensions 47

7.1.2 Other NLP extensions 48

A Tutorials **49**

List of Figures

2-1 The Scratch interface. 14

3-1 The App Inventor designer interface. 18

3-2 The App Inventor blocks interface. 19

4-1 A demonstration application using the Teachable Machine extension. 27

4-2 A demonstration application using the OCR extension. 29

5-1 The code for a “What is it?” application. 36

5-2 The code for a basic OCR application. 37

Chapter 1

Introduction

Over the past decade, machine learning has emerged as one of the most popular subjects for students, and for good reason: its applications appear everywhere, from speech recognition and movie recommendations to chess and Go playing and self-driving cars [1]. As a result, younger students may also be eager to study machine learning, but they may find the topic difficult because machine learning is fairly technical and relies on a solid mathematical foundation. This may discourage students from advancing further, which is unfortunate since the applications of machine learning are fascinating.

In this thesis, I developed several extensions for MIT App Inventor, a block-based web application for designing mobile applications, that will introduce students to what machine learning can do and allow them to build powerful applications. Currently, users are able to build a wide variety of applications by using standard code blocks (control flow, logic, and procedures) and other components that support more complicated features. My extensions integrate a common use of machine learning into an easy-to-use package that operates on the device itself. Examples of such components are ones related to computer vision, like object recognition or optical character recognition (OCR); natural language processing (NLP), such as sentiment analysis; and others, like speech processing. To use an extension, students simply have to upload it to App Inventor and use the extension's API calls to integrate its functionality into their application.

To test these extensions, I first created several tutorials about how to use each extension in a sample application. For instance, a computer vision application could be built into an application that recognized an object currently captured by a device's camera. I then created a short curriculum around these extensions and ran a 6-week class to determine its efficacy. Each lesson in the class had two halves: the first half consisted of lecture, where I discussed a certain machine learning topic, and the second half consisted of hand-on activities using an extension to develop applications that involved topics mentioned in the first half. I conducted a brief survey with the students and found that they had a stronger understanding of machine learning at the end of the course and that the extensions aided them significantly in learning the material.

The following chapters go over these steps in more detail. Chapter 2 discusses related work—other projects done by extending Scratch to include machine learning and other prepackaged machine learning services. Chapter 3 gives some background about App Inventor and TensorFlow.js, which was used to create the extensions. Chapter 4 contains implementation details of the extensions, and Chapter 5 includes examples of applications to be built using the extensions. Finally, Chapter 6 goes over how the extensions were evaluated in the context of a class.

Chapter 2

Related Work

Several educational coding platforms, some of which teach machine learning, already exist, as well as other products that make machine learning more convenient or accessible. The ones described below served as a guide for the App Inventor extensions developed for this thesis.

2.1 Scratch

Scratch [2] is a block-based programming language developed by the Lifelong Kindergarten Group at the MIT Media Lab. Originally released in 2003, Scratch is now used by many to create animations and games. Using Scratch is simple: on its website, simply drag blocks from a panel on the left into the middle screen, and the output of the blocks is shown on the right. Figure 2-1 shows a basic application made in Scratch; the left panel consists of a list of blocks categorized by function (e.g. motion, events, control, variables).

2.2 Cognimates

Cognimates [3], created by Stefania Druga with the MIT Media Lab in 2018, is a platform for teaching children about artificial intelligence (AI) and machine learning. It is based around several AI extensions for Scratch, allowing users to build Scratch

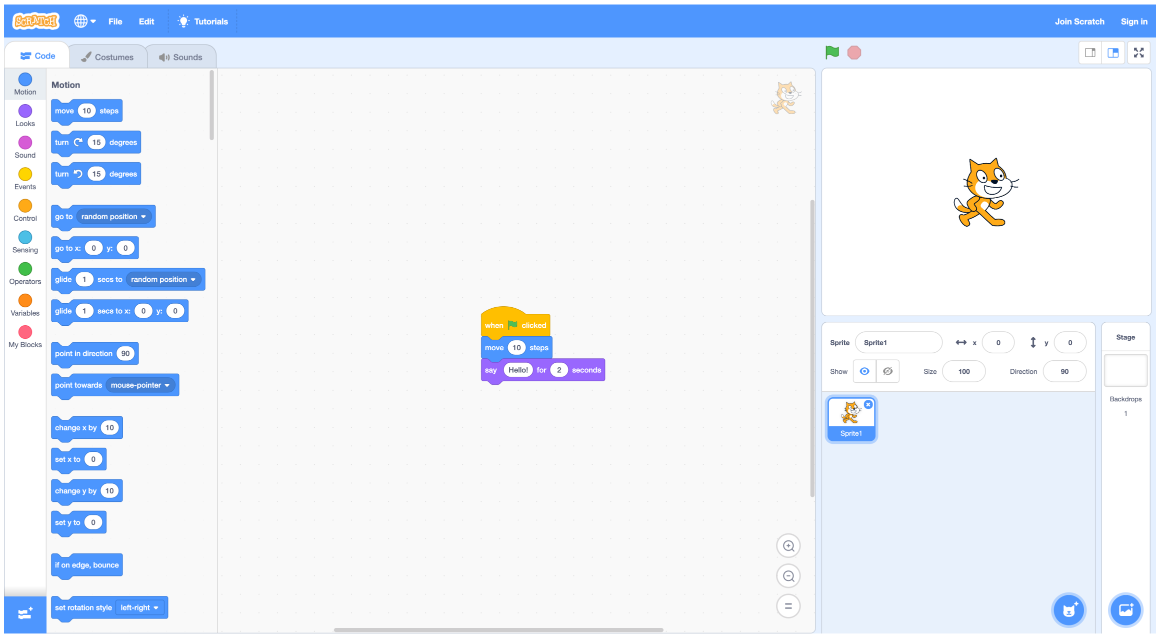


Figure 2-1: The Scratch interface.

projects that incorporate AI functionality, such as playing rock-paper-scissors with a webcam, classifying images, or performing speech-to-text. Cognimates is a desktop application and relies on API calls to IBM Watson [4] as its back end.

2.3 Machine Learning for Kids

Machine Learning for Kids [5], created by Dale Lane in 2017, is another platform for teaching machine learning using Scratch. Like Cognimates, it is a web application that uses IBM Watson for its functionality. Students can use a series of worksheets to guide them through developing their own Scratch applications that use machine learning, like image recognition, chatbots, and game playing.

In summary, Cognimates and Machine Learning for Kids extend the functionality of Scratch with machine learning blocks; likewise, my goal for this thesis was to develop extensions for MIT App Inventor so that students could make applications using machine learning. Unlike the works described above, applications with these extensions will work on mobile devices, not the desktop, and they can be integrated into a more complex application. My extensions should also work offline, so they must perform the back end machine learning computations on-device, rather than querying an external service like IBM Watson. In my extensions, these computations are handled with TensorFlow.js, which will be discussed in a later section.

2.4 Amazon and Google services

Companies like Amazon and Google provide services for developers to build machine learning applications. Amazon provides Alexa, NLP tools, Amazon Rekognition, AWS DeepLens, and others [8], all of which are capable resources but require some fee to use. In particular, DeepLens is a $249 video camera sold by Amazon for developers to experiment with computer vision, including object and face recognition, image classification, and style transfer. Likewise, Google's Cloud AI products [9] offer APIs for a variety of services but at a cost.

These services make machine learning more convenient for users in the sense that common uses are prepackaged, but they come at some cost, which may not be ideal for educational purposes. My App Inventor extensions don't rely on querying an external service, so they don't internal any costs on the user, yet they provide enough functionality to be an effective learning tool.

Chapter 3

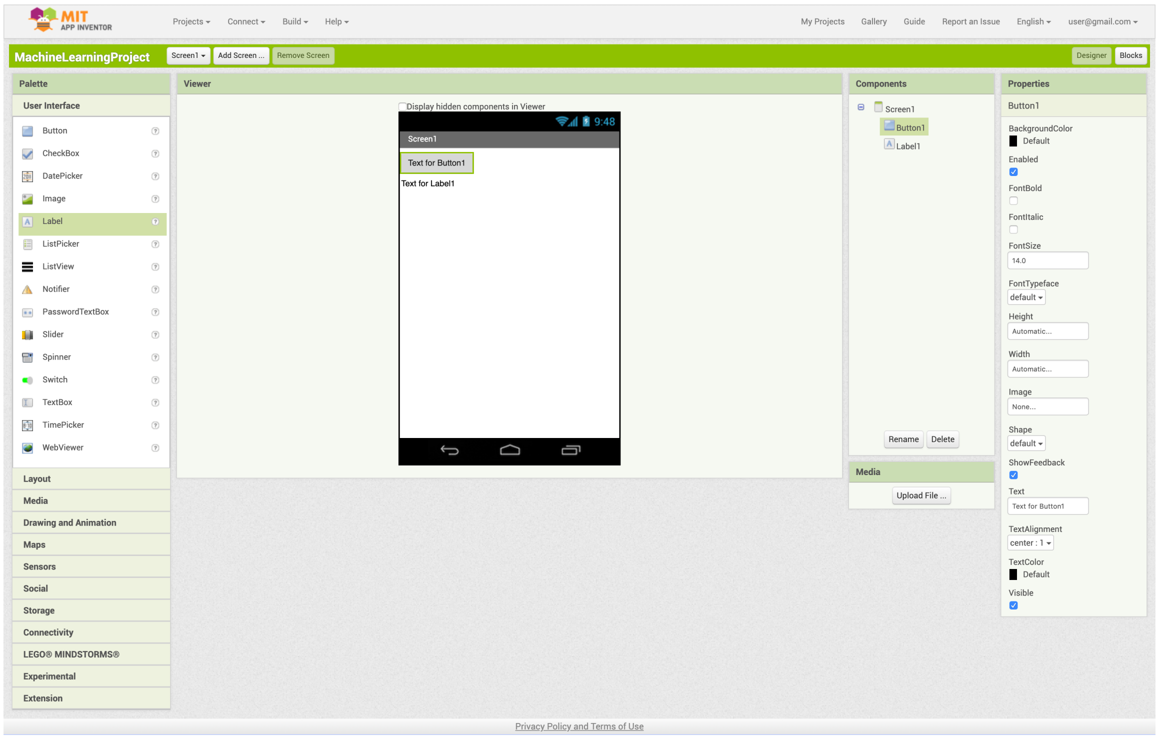
Background

This chapter discusses MIT App Inventor and TensorFlow.js. MIT App Inventor is the main application that the extensions are based on, and TensorFlow.js was used to build the extensions themselves.

3.1 MIT App Inventor

MIT App Inventor [6] is a web application for users with minimal programming experience to easily and quickly build mobile applications. It was first released in 2010 and was originally developed by Google; it has been maintained by MIT since 2011. App Inventor has over 400,000 unique monthly active users from 195 countries who have created almost 22 million applications [7]. Like Scratch, App Inventor features a simple drag-and-drop user interface for layout design along with a block-based programming language to enable interactivity.

The App Inventor development website consists of two interfaces: the designer interface, shown in Figure 3-1, and the blocks interface, shown in Figure 3-2. The designer interface contains various visible components, like buttons, images, labels, and tables, that define the layout of the application, as well as non-visible components, such as cameras, databases, and clocks, that are used by the application. These components are located in the palette on the left side of the interface. They are added to the application by dragging them into the middle, which shows a preview



Button1
BackgroundColor ■ Default
Enabled ☑
FontBold ☐
FontItalic ☐
FontSize 14.0
FontTypeface default
Height Automatic...
Width Automatic...
Image None...
Shape default
ShowFeedback ☑
Text Text for Button1
TextAlignment center : 1
TextColor ■ Default
Visible ☑

Figure 3-1: The App Inventor designer interface.

of what the application looks like. Each component has properties that can adjusted on the right side. These properties vary by the type of component added:

Visible components are added onto the application preview window, where they can be re-positioned to the user's liking. Non-visible components are displayed at the bottom of the preview window to indicate that they are being used. It is possible for the application to consist of multiple screens (different pages of the application), so the second column from the right shows all components organized by screen. Media can also be uploaded (like images), which then can be used in the application.

The blocks interface can be accessed by clicking the "Blocks" button on the top right of the designer interface. These blocks define the logic of the application and are can be categorized into two main groups: built-in blocks and component blocks. The built-in blocks are grouped as follows:

- Control blocks: these blocks control the flow of the program and include if/else statements, for loops, and while loops.

- Logic blocks: these blocks include true/false blocks as well as not/and/or oper-

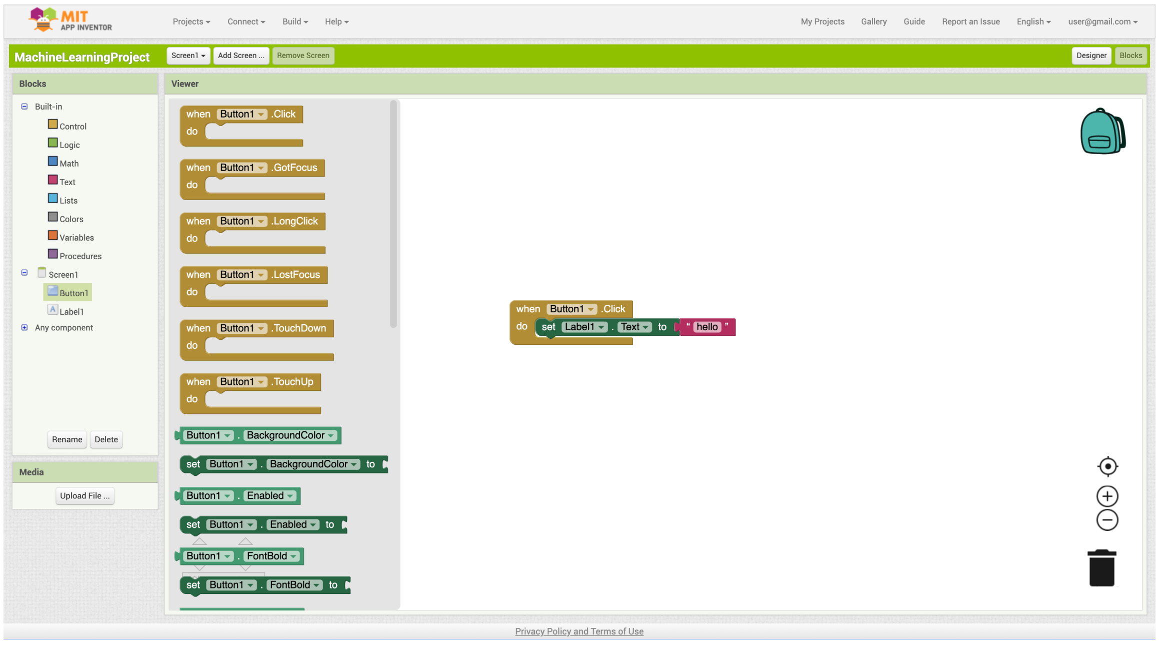


Figure 3-2: The App Inventor blocks interface.

ators.

- Math blocks: numbers and various math functions like arithmetic operators are found here.
- Text blocks: these blocks relate to strings and string operations.
- List blocks: these blocks manipulate lists (creation, length, additions/removals, etc.).
- Color blocks: these blocks are used to specify colors.
- Variable blocks: set and get variables with these blocks.
- Procedure blocks: these blocks are for using procedures.

Furthermore, each component (visible or non-visible) currently being used in the designer will also have its own series of blocks. These are usually methods to change the properties of the components (e.g. the text of a label) or events that fire when triggered through the application (e.g. when a button is clicked, execute a series of blocks).

These blocks can be dragged out from the left side into the viewer and combined to form a full-fledged program. Figure 3-1 and Figure 3-2 show a simple project consisting of a button and a label—when the button is clicked, the label's text is set to “hello”. From the designer interface, one can see that two components have been added: a button and a label. The blocks interface makes use of two component blocks—one from each component—and one built-in block (the string “hello”). Clicking the button causes the dark yellow event block is fired, and the functions inside the event are called. The event block contains just one function, which sets the label text to “hello”.

Together with the designer, these blocks make up an application that can be interacted with on a mobile device. This is done by first downloading the App Inventor Companion application on the mobile device. A QR code can then be displayed by the development interface by clicking the “AI Companion” option under the “Connect” drop-down menu. Scanning the QR code using the Companion application brings up the custom-built application on the device, where it can be interacted with by the user.

3.2 TensorFlow.js

Most of the machine learning extensions are built using TensorFlow.js [10], a JavaScript library that can be used to develop machine learning applications on the web. As its name suggests, TensorFlow.js is based on TensorFlow [11], an open source machine learning framework developed by Google; TensorFlow.js is relatively new, having been announced in the first half of 2018. Normally, running computationally intensive calculations, like those required by machine learning, would be infeasible using JavaScript, which is fairly slow compared to compiled languages like C++. TensorFlow.js, however, runs using WebGL, meaning that the machine learning code runs on the device's graphics processing unit and can take advantage of its parallel processing abilities to drastically speed up computations.

TensorFlow.js was chosen over more native solutions on iOS and Android to maintain cross-compatibility and ease the development process since the JavaScript code is platform agnostic. To run JavaScript inside an application, the code is executed inside a web view, as discussed in the next section.

Chapter 4

Implementation

Seven extensions were developed for App Inventor, spanning several machine learning subfields, including computer vision and natural language processing. I implemented the first three extensions below as components in the spring of 2018 with the help of a UROP student (Kelsey Chan).

4.1 Components vs. Extensions

There are two main ways to add new features to App Inventor: adding a component or creating an extension. The majority of App Inventor is written in Java, and both ways require writing a Java class that contains the component or extension code, along with adding asset files. The Java class contains methods that correspond to the blocks (functions and events) for that component or extension. Components are essentially part of App Inventor itself—they are included in the software by default. For example, the items in the palette mentioned in previous chapter, like buttons, labels, and clocks, are components. To add a new component, the entire App Inventor system 9both the web and mobile application) would have to be updated. In contrast, an extension can be thought of as a custom component and is a file that is separately uploaded to App Inventor to add new features. Uploading an extension is simple: users just need to open the "Extension" section at the bottom of the palette and select the appropriate extension file on their computer. Writing extensions instead of components makes them easier to update, as they are not tied to App Inventor updates.

Initially, the extensions were implemented as visible components, and they were integrated into a custom build of App Inventor. After the first three components were finished, it was decided to convert them to extensions for the reasons above, as well as all future components.

4.2 Extensions

Each extension is implemented as a non-visible component. To load one into an application, a WebViewer (a built-in visible component for App Inventor) first needs to be added into the application. The properties panel for the extension shows a list of WebViewers in the screen, and one of them must be chosen to load the content of the extension.

The extension itself is a webpage that runs JavaScript code. All the files for the JavaScript code and other files needed for the extension, like TensorFlow.js and the machine learning models, are stored with the extension, and all the JavaScript runs client-side, so the entire application can run offline. Communication between the extension and the rest of the App Inventor application is handled in two ways, depending on the direction of communication. The main extension code (in Java) is run when blocks are executed, and these call the JavaScript code in the WebViewer using the evaluateJavascript function. the evaluateJavascript function contains a snippet of JavaScript code (in text form) that runs in the WebViewer. For the JavaScript code to send results back to the main extension, a JavascriptInterface is used, which adds a class with the name of the extension to the JavaScript code. Methods can be called on this class in the JavaScript extension code, which triggers event blocks.

4.2.1 Object recognition

Given an image or a video stream, this extension tries to identify what is being shown. More precisely, the extension outputs a list of candidate labels and confidences, which can be parsed by App Inventor blocks. This extension is intentionally designed to be as simple as possible because it is meant to serve as an introduction to all the other components; an image classifier can be built in only a few minutes. To classify video, all that is needed is to repeatedly make a call to classify the video stream (such as every 500 milliseconds).

Internally, this extension uses a MobileNet [12], a small, low-latency, low-power convolutional neural network designed by Google that is specifically meant to run on mobile devices. MobileNet is trained on a list of 1000 labels, so this extension can theoretically differentiate between these 1000 objects. TensorFlow.js loads the structure and weights of this neural network, which are also small enough to be stored with the application, and then performs inference using the given input data. This MobileNet is used in several other components as well.

The object recognition extension uses the following blocks:

- ClassifierReady event: this event is called when the extension is ready to classify images or video.
- GotClassification event: this event is called when a classification has finished successfully. It returns the top 10 results, ordered by highest confidence, of the current image or video.
- Error event: this event is called when an error occurs. It returns an error code.
- ClassifyImageData function: this function performs classification on a given image path. It triggers the GotClassification event when classification is finished successfully.
- ClassifyVideoData function: this function performs classification on the current video frame. It also triggers the GotClassification event when classification is finished successfully.
- ToggleCameraFacingMode function: this function toggles the current camera between the user-facing and environment-facing camera.

- InputMode property: this property determines if the extension is configured to use image or video data.

4.2.2 Teachable Machine

The Teachable Machine extension is based off Google's Teachable Machine [13], a browser application meant to teach users about how training and using a machine learning model works. Users are able to provide samples of webcam data to train the model, and then the model predicts which sample of data the current webcam image is closest to (along with its confidence). Google's Teachable Machine uses a k-nearest neighbors (KNN) classifier to classify images.

The App Inventor counterpart is implemented similarly but with slightly more flexibility: users are able to train a custom number of classes using their device's camera and provide custom labels. The extension continuously updates the predict classes and confidences, and users are able to save and load sets of classes as well.

It is also possible to implement this extension with transfer learning on top of MobileNet, but results are not as immediate, and the slightly higher accuracy obtained is not worth the tradeoff.

This extension uses the following blocks:

- ClassifierReady event: this event is called when the extension is ready to classify video.

- GotSampleCounts event: this event is called when the sample counts of any class have been updated.

- GotConfidences event: this event is called when the confidences of any class have been updated.

- GotClassification event: this event is called when a classification has finished successfully. It returns the class label with the highest confidence.

- DoneSavingModel event: this event is called when saving a model has finished.

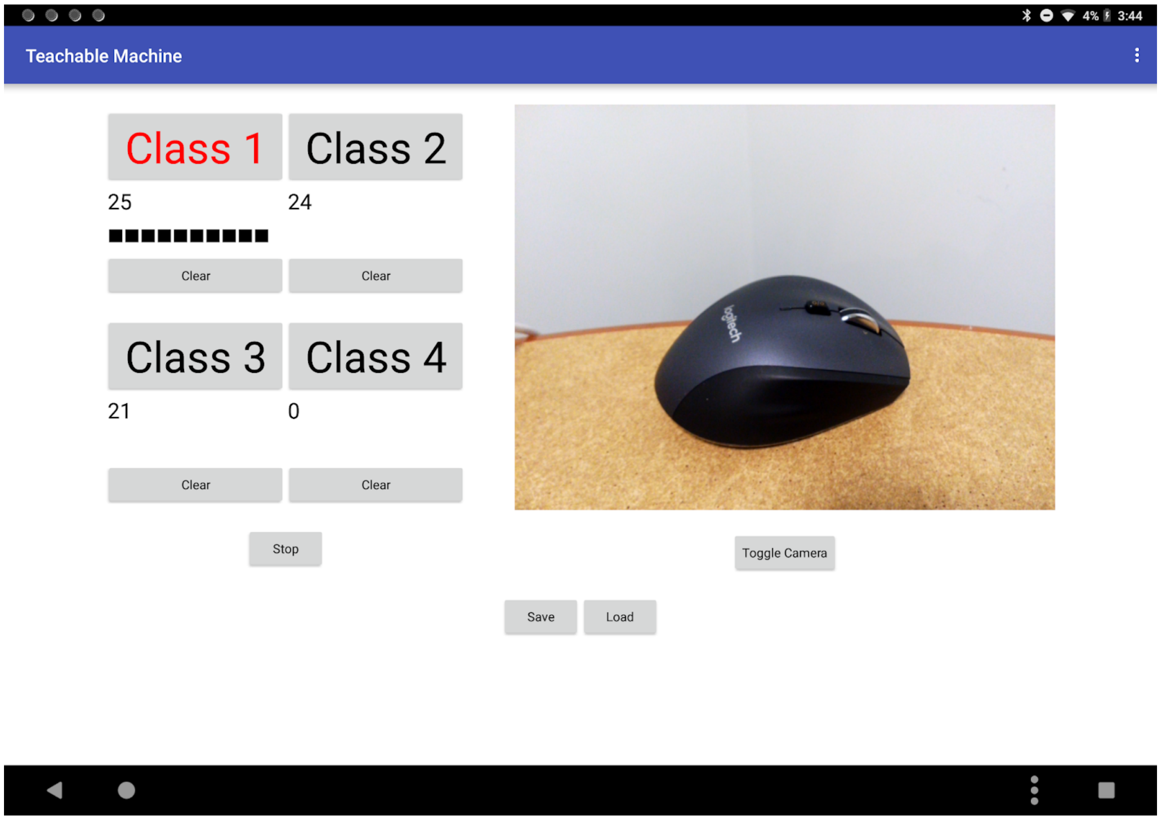


Figure 4-1: A demonstration application using the Teachable Machine extension.

- DoneLoadingModel event: this event is called when loading a model has finished.
- Error event: this event is called when an error occurs. It returns an error code.
- StartTraining function: this function starts training the machine to associate images from the camera with the provided label.
- StopTraining function: this function stops collecting images from the camera to train the machine.
- Clear function: this function clears the training data associated with the provided label.
- SaveModel function: this function saves the current model (the set of samples and labels) with the provided name.

- LoadModel function: this function loads the model with the provided name.

- ToggleCameraFacingMode function: this function toggles the current camera between the user-facing and environment-facing camera.

Figure 4-1 shows an application built using the Teachable Machine extension. The screenshot shows that class 1 has 25 samples, class 2 has 24 samples, class 3 has 21 samples, and class 4 has 0 samples. The application is currently predicting that the input video (which is showing a mouse) matches class 1 with the highest confidence, so class 1 is in red. The black boxes underneath class 1 are a visual indicator of the confidence: in this case, one box equals a confidence of 10%, so the model is 100% (to the nearest 10%) confident of its prediction.

4.2.3 OCR

The OCR extension is unique in that it does not rely on TensorFlow.js; instead, it uses Tesseract.js [14], a JavaScript library specializing in multilingual OCR. Users are able to submit an input image, and the extension outputs the text in the image, along with the bounding boxes of each word and each word's confidence. The extension also allows users to select the language used, blacklist or whitelist certain characters, and filter the text with a minimum confidence level.

This extension uses the following blocks:

- ClassifierReady event: this event is called when the extension is ready to classify images or video.

- GotProgress event: this event is called when the progress of the extension is updated. This is entirely optional and can be used as a progress marker.

- GotText event: this event is called when the OCR has finished successfully. It returns the text in the image or video frame.

- GotFilteredText event: this event is also called when the OCR has finished successfully. Every word that the OCR extension extracts has an associated

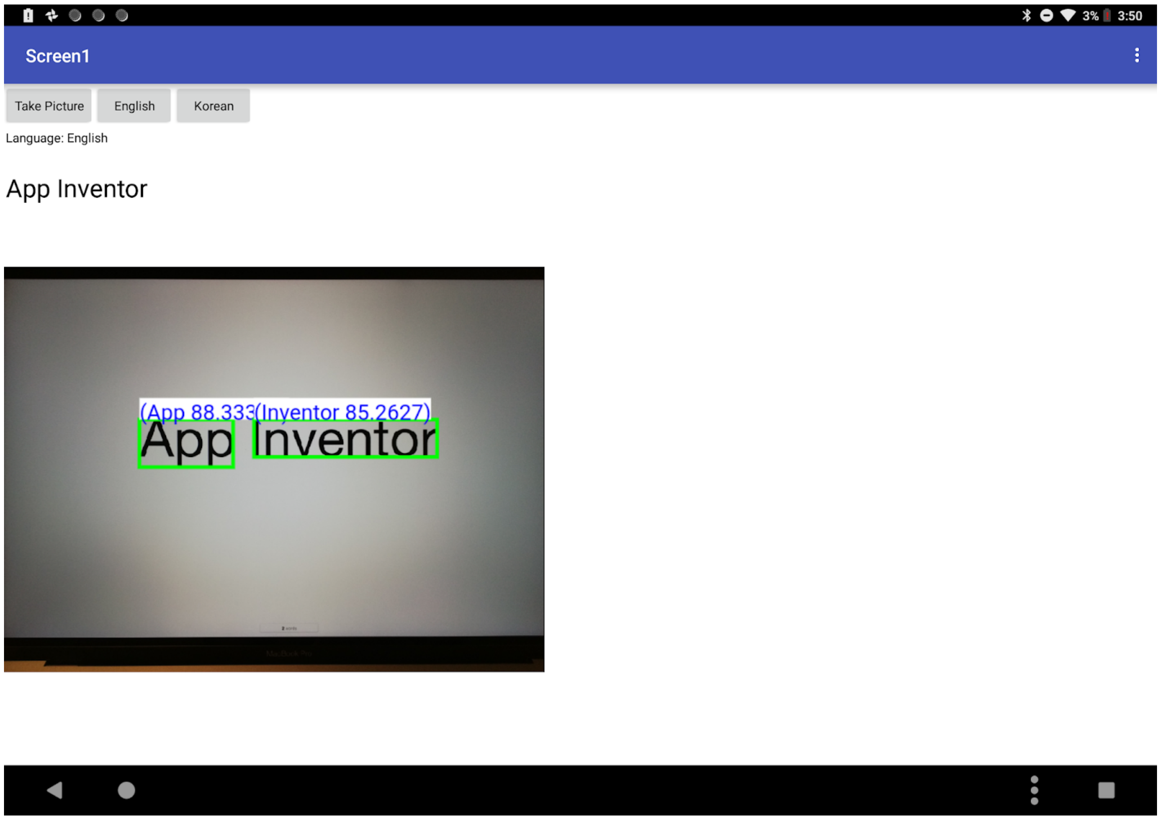


Figure 4-2: A demonstration application using the OCR extension.

confidence, and this event returns a filtered version of the text in the image or video frame: only words with a confidence higher than a set threshold are included in the output.

- GotWords event: this event is also called when the OCR has finished successfully. It returns a list of all the words in the image or video frame with their associated confidences.

- Error event: this event is called when an error occurs. It returns an error code.

- RecognizeImageData function: this function performs OCR on a given image path. It triggers the GotText, GotFilteredText, and GotWords events when the OCR is finished successfully.

- RecognizeVideoData function: this function performs OCR on the current video frame. It also triggers the GotText, GotFilteredText, and GotWords events when the OCR is finished successfully.

- SetLanguage function: this function sets the language in which to perform OCR.
- Clear function: this function clears the image or video of bounding boxes around words.
- DrawBoundingBox function: this function draws a bounding box around a given word with a customizable label.
- ToggleCameraFacingMode function: this function toggles the current camera between the user-facing and environment-facing camera.
- InputMode property: this property determines if the extension is configured to use image or video data.

Figure 4-1 shows an application built using the OCR extension. The screenshot shows bounding boxes around each word in the image, with the label of each box set to the detected word and its confidence. The language used for OCR was English.

4.2.4 Object detection

The object detection extension works like the object recognition extension, but it detects all objects in the input with their confidences rather than one and also identifies where each object is with a bounding box. One downside is that this extension can only recognize 90 different classes as opposed to the 1000 classes for object recognition. The object detection extension is based off research from Google relating to real-time object detection [15] and uses a Single Shot MultiBox Detector (SSD) [16] to draw bounding boxes.

This extension uses the following blocks:

- ClassifierReady event: this event is called when the extension is ready to classify images or video.

- GotDetection event: this event is called when the detection has finished successfully. It returns the objects in the video frame with their confidences.

- Error event: this event is called when an error occurs. It returns an error code.

- DetectImageData function: this function performs OCR on a given image path. It triggers the GotText, GotFilteredText, and GotWords events when the OCR is finished successfully.

- DetectVideoData function: this function performs OCR on the current video frame. It also triggers the GotText, GotFilteredText, and GotWords events when the OCR is finished successfully.

- Clear function: this function clears the image or video of bounding boxes.

- ToggleCameraFacingMode function: this function toggles the current camera between the user-facing and environment-facing camera.

- InputMode property: this property determines if the extension is configured to use image or video data.

4.2.5 Sentiment analysis

A popular NLP application is sentiment analysis, which returns the sentiment (i.e. positivity or negativity) of a given piece of text, typically a sentence to a paragraph. This extension is based around a long short-term memory (LSTM) network and can be trained on a set of labeled movie reviews.

This extension uses the following blocks:

- ClassifierReady event: this event is called when the extension is ready to classify text.

- GotClassification event: this event is called when the text filtering has finished successfully. It returns the sentiment of the text on a scale of 0 to 1, with 0 being negative and 1 being positive.

- Error event: this event is called when an error occurs. It returns an error code.
- ClassifyTextData function: this function classifies the given text.

4.2.6 Text filtering

Another common use of NLP is to filter text based on its content. This extension categorizes input text based on one of seven different types of toxicity.

This extension uses the following blocks:

- ClassifierReady event: this event is called when the extension is ready to classify text.
- GotClassification event: this event is called when the text filtering has finished successfully. It returns the confidences of each of the toxicity types.
- Error event: this event is called when an error occurs. It returns an error code.
- ClassifyTextData function: this function classifies the given text.

4.2.7 Speech processing

The speech processing extension continuously samples input audio and recognizes when one word out of a limited set of 18 words has been said (as well as general categories for an unknown word or just background noise). After every such sampling, the confidences of each category are outputted. This extension works by training a convolutional neural network on the spectrograms of hundreds of thousands of input audio clips [18].

This extension uses the following blocks:

- ClassifierReady event: this event is called when the extension is ready to classify audio.
- GotClassification event: this event is called when the speech processing has finished successfully. It returns the confidences of each of the 18 words from a live audio clip.

- Error event: this event is called when an error occurs. It returns an error code.
- StartListening function: this function starts listening for audio to classify.
- StopListening function: this function stops listening.

Chapter 5

Applications

To evaluate these extensions, I developed a simple curriculum using these extensions to teach students about machine learning with a series of tutorials. The curriculum involves utilizing these extensions to build "cool" yet educational mobile applications.

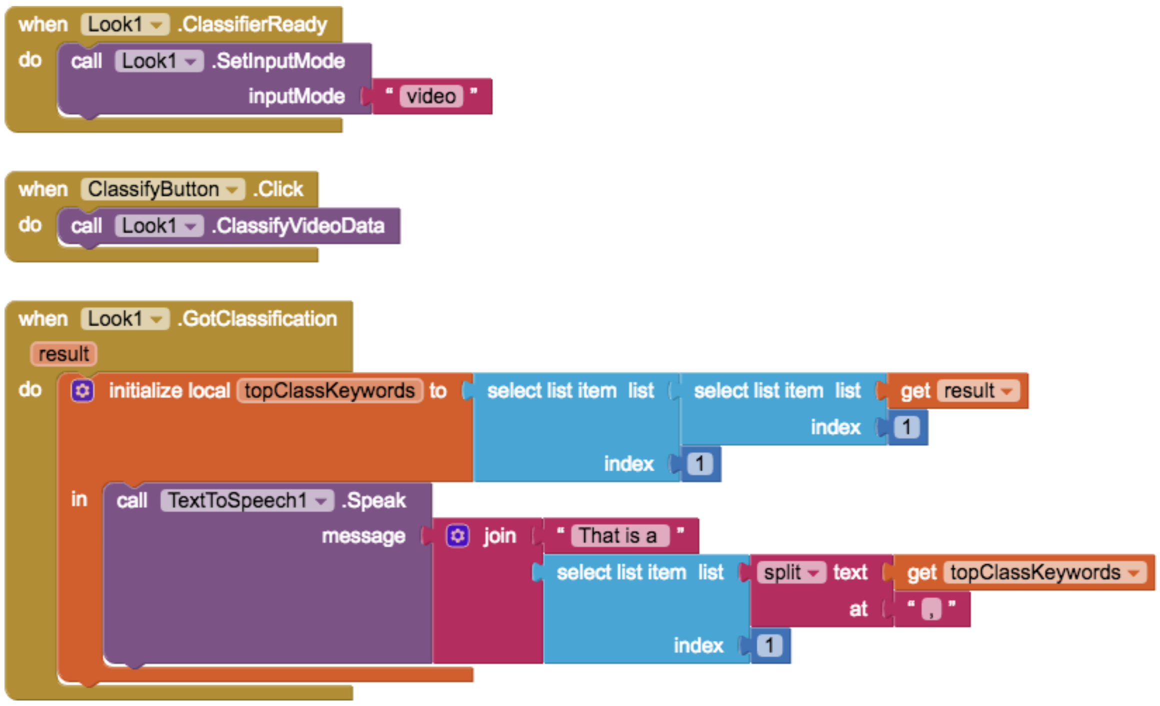
5.1 What is it?

The first tutorial uses the object recognition extension to create a simple "What is it?" application, which will identify the object currently shown in the input image (either via a photo or a video stream). This could be paired with a translation block to build a simple tool to learn the names of objects in another language. As mentioned previously, this tutorial is straightforward and serves as an introduction to the more complicated tutorials later.

This tutorial could also be extended with the object detection extension in order to list all the objects appearing in a scene, instead of just one.

5.2 Rock–paper–scissors

This tutorial uses the Teachable Machine extension to build an application to play rock–paper–scissors with a computer. First, each class of hand signs is trained, and then the game is started. After a countdown, the application detects the hand sign



```
when Look1 .ClassifierReady
do  call Look1 .SetInputMode
               inputMode  " video "

when ClassifyButton .Click
do  call Look1 .ClassifyVideoData

when Look1 .GotClassification
  result
do  initialize local topClassKeywords to  select list item  list  select list item  list  get result
                                                                                      index  1
                                                             index  1
    in  call TextToSpeech1 .Speak
                  message  join  " That is a "
                                 select list item  list  split text  get topClassKeywords
                                                               at  " , "
                                                   index  1
```

Figure 5-1: The code for a "What is it?" application.

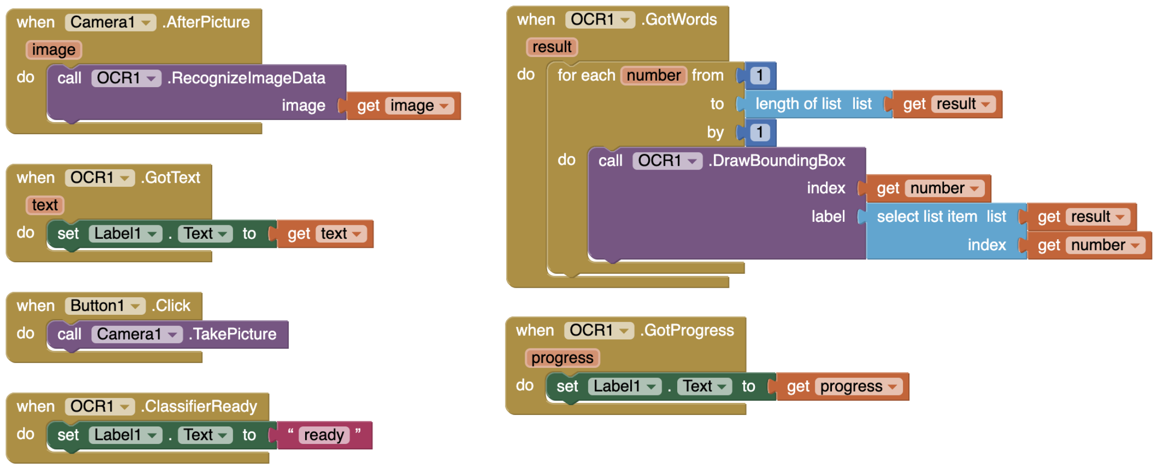
shown, chooses a random hand sign, and compares the results to determine who won. This can repeated multiple times, and other variations can be incorporated (e.g. adding a fourth object).

5.3 Emoji scavenger hunt

This tutorial is based on Google's Emoji Scavenger Hunt [19]. An emoji will be shown at random, and users will have a limited amount of time (typically 30 seconds) to find an object represented by the emoji in the room and point the device's camera to that object. If the object recognition extension confirms the match, the next emoji is shown; otherwise, it's game over.

5.4 Google Lens

Using the OCR extension, it's possible to build a rudimentary version of Google Lens (where translations of words appear over ones in another language in the camera preview). The OCR extension allows custom labels to be affixed to the bounding boxes of words, so using a translate block, these words can have their translations appear at the top.



```
when Camera1 .AfterPicture
  image
do  call OCR1 .RecognizeImageData
                          image  get image

when OCR1 .GotText
  text
do  set Label1 . Text to  get text

when Button1 .Click
do  call Camera1 .TakePicture

when OCR1 .ClassifierReady
do  set Label1 . Text to  " ready "

when OCR1 .GotWords
  result
do  for each number from  1
                     to  length of list  list  get result
                     by  1
    do  call OCR1 .DrawBoundingBox
                          index  get number
                          label  select list item  list  get result
                                                  index  get number

when OCR1 .GotProgress
  progress
do  set Label1 . Text to  get progress
```

Figure 5-2: The code for a basic OCR application.

5.5 NLP tutorials

The NLP extensions should stand alone; their functionality itself should be the tutorial. There should be a discussion about how the extension operates and experiments conducted on how well the extension performs (e.g. how accurate is the sentiment analysis or text filtering?).

5.6 Robot control

For the speech processing extension, the tutorial is based on controlling an onscreen robot with voice controls. It is possible to command the robot to move in four directions, say various numbers, and tell the robot to go or stop.

Chapter 6

Evaluation

I tested these machine learning extensions through a class I ran during Spring HSSP 2019. HSSP is a program at MIT where MIT students can volunteer to run a class dedicated to a topic of their choosing for high school and middle school students located the Boston and Cambridge area. (My class was limited to high school students because I felt that the material and pace of the class wasn't too suited for middle school students.) The 6 classes were on Saturdays from February 23, 2019, to April 6, 2019 (there was no class on March 16 due to Splash, another educational program at MiT) from 1:00 to 2:30 PM. The class took place in a computer lab, so each student could use App Inventor, and I supplied them with tablets to test their applications. Attendance varied from 6 to 14 students over the 6 classes.

Each class was split up into two parts: lecture and application development. The lecture, which occupied approximately the first third to first half of each class, introduced various machine learning concepts to the students. The rest of the class was dedicated to building applications using the extensions, which related to the topics discussed beforehand.

I conducted a short survey at the beginning and end of the class, which found that the class, and in particular the extensions, helped the students gain an understanding of machine learning.

6.1 Class 1

Class 1 took place on February 23 and featured an introduction to machine learning as well as to App Inventor. Students built a “What is it?” application. For all of the classes, many of the figures in the slides were taken from [1].

A brief outline of class 1 is as follows:

- Introduction to machine learning
  - Types of machine learning
  - Supervised learning
    - Classification
    - Regression
      - Linear regression
      - Gradient descent
  - Unsupervised learning
    - Clustering
  - Semisupervised learning
  - Reinforcement learning
- Introduction to App Inventor
  - Using extensions
- Object recognition extension
  - Building a “What is it?” application

6.2 Class 2

Class 2 took place on March 2 and discussed regression and support vector machines (SVMs). Students worked on making “Teachable Machine” application.

A brief outline of class 2 is as follows:

- Regression
  - Polynomial regression
  - Regularization
  - Ridge regression
  - Lasso regression
  - Elastic Net
  - Logistic regression
- Support vector machines
  - Margins
  - Decision function
- Teachable Machine extension
  - Building a "Teachable Machine" application

6.3 Class 3

Class 3 took place on March 9 and included an introduction to neural networks. Students continued building on their "Teachable Machine" applications to make an interactive "rock–paper–scissors" game.

A brief outline of class 3 is as follows:

- Neural networks
  - Biological neurons
  - Perceptron
  - Backpropagation
  - Activation functions
  - Hyperparameters

  - Transfer learning
  - Optimizers
  - Learning rate
  - Dropout

- Introduction to convolutional neural networks
- Teachable Machine extension
  - "Rock–paper–scissors" application

6.4 Class 4

Class 4 took place on March 23 and contained an overview of convolutional neural networks (CNNs). Students worked with the object detection extension and an "Emoji scavenger hunt" game.

A brief outline of class 4 is as follows:

- Convolutional neural networks
  - Layers
  - Padding
  - Stride
  - Feature maps
  - Pooling
  - CNN architectures
- Object detection extension
- "Emoji scavenger hunt" game

6.5 Class 5

Class 5 took place on March 30 and went over recurrent neural networks (RNNs). Students tested out the OCR and sentiment analysis extensions (the text filtering extension was skipped).

A brief outline of class 5 is as follows:

- Recurrent neural networks
  - Neurons
  - Memory cells
  - Sequence classifiers
  - Time series
  - LSTM cells
  - Word embeddings
  - Encoders and decoders
- OCR extension
  - "Google Lens" application
- Sentiment analysis extension

6.6 Class 6

Class 6 took place on April 6 and briefly explored other topics in machine learning, like decision trees and ensemble learning. Most of the time was spent working on OCR and speech processing extensions.

A brief outline of class 6 is as follows:

- Decision trees
  - CART algorithm

- Ensemble learning
- Random forests
- Adaboost
- OCR extension
- Speech processing extension

6.7 Results

At the beginning of class 1, I asked the students to rate on a scale of 1 to 5 how eager they were learn about machine learning and how much they already knew about the topic. Students gave a rating of 4.07 out of 5 (with 14 responses) for the first question, showing that they wanted to learn about machine learning. The distribution was 3 3's, 7 4's, and 4 5's. This is perhaps not too surprising, considering that students had self-selected the courses that they wanted to take out of a catalog of tens of classes.

The second question received a rating of 1.43 out of 5 (with 14 responses), meaning that the students didn't have a lot of previous background on machine learning. The distribution was 8 1's and 6 2's. Since these students were only in high school, where machine learning normally isn't taught, this is to be expected.

At the end of class 6, I asked the students to rate if they felt they had a stronger understanding of machine learning. This questions got a rating of 4.5 out of 5 (with 6 responses), demonstrating that the class helped them to learn the subject. The distribution was 3 4's and 3 5's. Furthermore, I asked the students if they felt that working with the extensions aided them significantly in learning the material, and I received a response of 4.7 out of 5 (with 6 responses). The distribution was 2 4's and 4 5's. Even with the limited sample size, it is clear that the extensions proved beneficial to grasping the concepts introduced in the first part of the class.

In terms of more qualitative measures, I observed over the course of the class that while some students advanced more quickly than others and were able to complete their applications more quickly, most of the students were able to build completed applications with the extensions. I received feedback at the end of the class from students who appreciated the large amount of time given to them to work on developing their own mobile applications.

Chapter 7

Conclusion

Over the course of this thesis, I gave an overview of the process of developing machine learning extensions for MIT App Inventor, a web application designed to let students and other users build mobile applications without previous programming experience. A total of 7 extensions were made and span a large gamut of applied uses of machine learning. These extensions can be used to create interesting applications like a "rock–paper–scissors" game or a "Google Lens" text translator. Through a class, I demonstrated that students were able to pick up machine learning concepts using these extensions.

7.1 Future Work

7.1.1 Other computer vision extensions

There are several other possible computer vision extensions that were not implemented but could be added to App Inventor. Multi-person pose estimation is certainly feasible using PoseNet [17], and face or emotion recognition could be implemented as well (although face recognition is not a priority since this is doable with the Teachable Machine extension). Style transfer is another common technique in computer vision; however, it remains to be seen if style transfer can be done efficiently on a mobile device.

7.1.2 Other NLP extensions

Two NLP extensions that could be implemented in the future are translation and text generation. The translate extension would perform offline translation; while this extension would necessarily be inferior in quality to a typical Google Translate result, it would still be useful as a teaching tool around the amount of training data needed for translation tools. It would implemented as a sequence-to-sequence model trained on several hundred thousand pairs of sentences in two languages.

The text generation extension would use LSTM networks to generate text using a given seed text. For example, given the text of a Shakespeare novel, an LSTM network could, after sufficient rounds of training, produce Shakespeare-esque output; with written code as input, the network could generate pseudocode (that may not necessarily compile).

Appendix A

Tutorials

What Is It? Tutorial

With this app, you can capture live video using your phone camera, and with the click of a button, the app will tell you what object is in the live video.

Getting Started

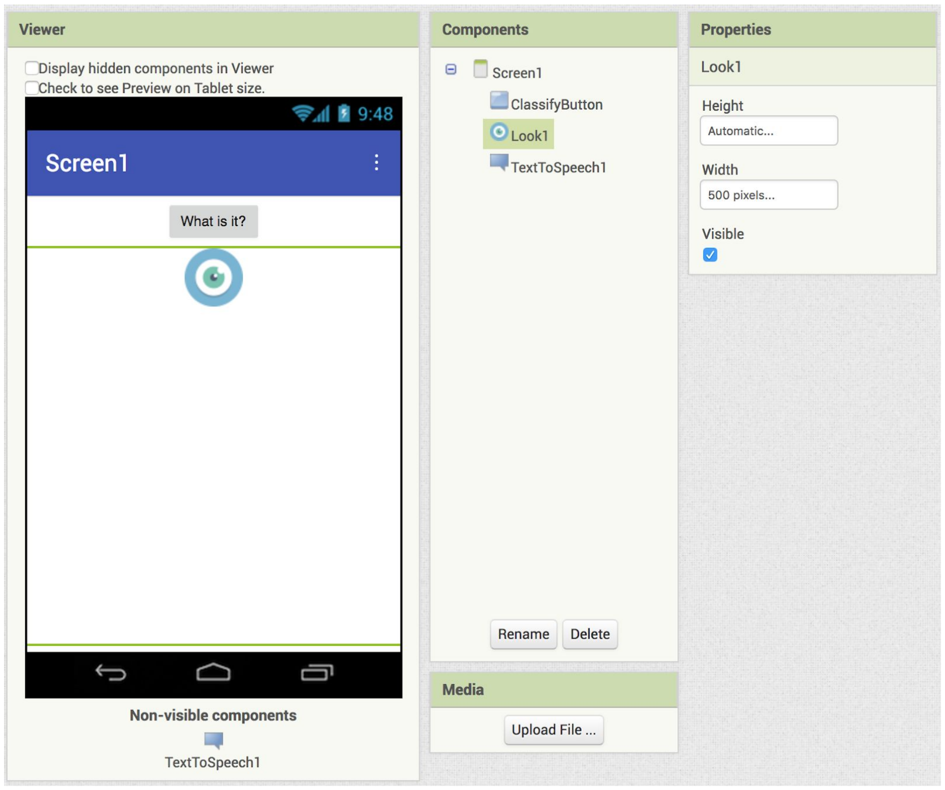
The app demonstrates how to use the **Look** component to recognize objects in live video.

It makes use of the following App Inventor components:

- Look
- TextToSpeech

User Interface

Here are the components for the What Is It? app, as shown in the Component Designer:

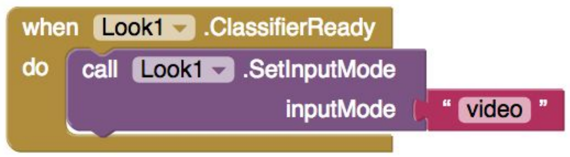


Height: Automatic...
Width: 500 pixels...
Visible: ✓

The user interface consists of one button and a Look component called Look1. There is also a non-visible TextToSpeech component. The width of Look1 is set to 500 pixels instead of "Automatic" to prevent it from filling the entire screen. When the camera starts, Look1 will show the live stream video seen by the camera.

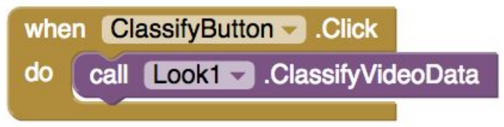
The App's Behavior

Here are the blocks for the What Is It? app:



```
when Look1 .ClassifierReady
do  call Look1 .SetInputMode
                   inputMode  " video "
```

Here we change the input mode of the Look component to video. By default, the Look component takes in image data, so to classify live video, the input mode needs to be changed. The “ClassifierReady” event is called when Look1 is ready to be used.



When the “What is it?” button is clicked, Look1 begins to classify the current frame from the live video. When this is finished, the result is returned in the “GotClassification” event.

The Look component’s AI vision processing identifies an image as belonging to a class of images it knows about. The component also produces a number between 0 and 1 that reflects how much confidence the processing has in making that classification. The result of “GotClassification” is returned as a list in the form

```
[[class1, confidence1], [class2, confidence2], …, [class10, confidence10]]
```

Each item in the list is a pair consisting of **a class** together with **the confidence level for that class**. The pairs are ordered from most confident to least confident.

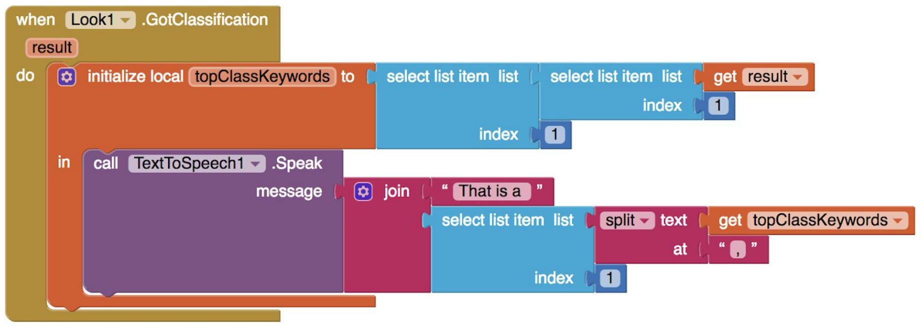
Each class itself is a sequence of keywords separated by commas. Some classes have only one keyword, but some classes have more than one keyword. For example, `class1` above could be

```
cellular telephone, cellular phone, cellphone, cell, mobile phone
```

GotClassification processes this result, first getting the class with the highest confidence by extracting the first [class, confidence] pair in `result` and saving the first element of the pair (the class with the highest confidence) as `topClassKeywords`.

It then splits the sequence of class keywords, which are separated by commas, and gets the first keyword in the sequence. This component passes the text “That is a [first keyword in class]” to the TextToSpeech component, which says the sentence out loud.

Here is the entire event handler:



```
when Look1 .GotClassification
  result
do  initialize local topClassKeywords to  select list item  list  select list item  list  get result
                                                                                     index  1
                                                                  index  1
    in  call TextToSpeech1 .Speak
                     message  join  " That is a "
                                    select list item  list  split  text  get topClassKeywords
                                                                   at  " , "
                                                      index  1
```

A complete list of possible classes can be found here.

Variations

- Add functionality to toggle between the front- and rear-facing cameras.
- Use the image input mode so that users can take a picture with the camera and classify the image they captured.
- Use the YandexTranslate component to translate the sentence into another language.

Review

Here are some of the ideas covered in this tutorial:

- The **Look** component can take live video as input and classify the object in the provided live video. Its GotClassification event is triggered after ClassifyVideoData has been called, and the classification has successfully completed.

Scan the Sample App to your Phone

Download Source Code

Tutorial Version:

- App Inventor 2

Tutorial Difficulty:

- Beginner

Tutorial Type:

- Computer Vision

What Is It? Tutorial

With this app, you can take a picture using your phone camera, and with the click of a button, the app will tell you what object is in the photo. By programming the app to speak in a different language, you can use this app to learn new vocabulary in a different language.

Getting Started

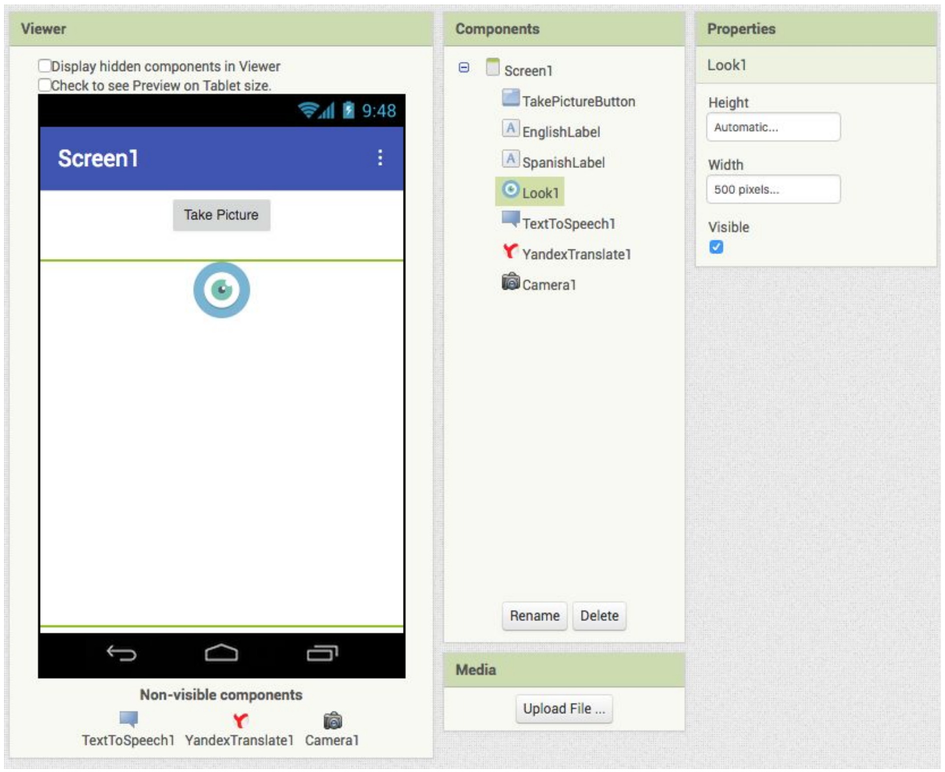
The app demonstrates how to use the **Look** component to recognize objects in live video.

It makes use of the following App Inventor components:

- Look
- TextToSpeech
- YandexTranslate
- Camera

User Interface

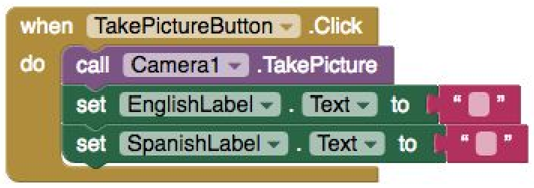
Here are the components for the What Is It? app, as shown in the Component Designer:



The user interface consists of one button, two labels, and the Look component. There are also three non-visible components: TextToSpeech, YandexTranslate, and Camera. The width of the Look component is set to 500 pixels instead of "Automatic" to prevent it from filling up the entire screen.

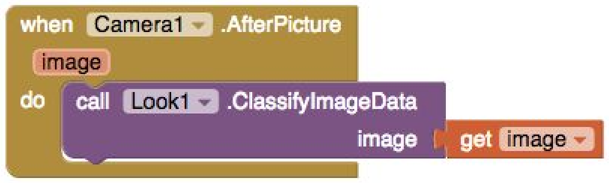
The App's Behavior

Here are the blocks for the What Is It? app:



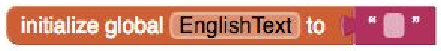
```
when TakePictureButton .Click
do  call Camera1 .TakePicture
    set EnglishLabel . Text to " "
    set SpanishLabel . Text to " "
```

When the “Take Picture” button is clicked, the Camera component lets the user take a picture. It resets the EnglishLabel and the SpanishLabel.



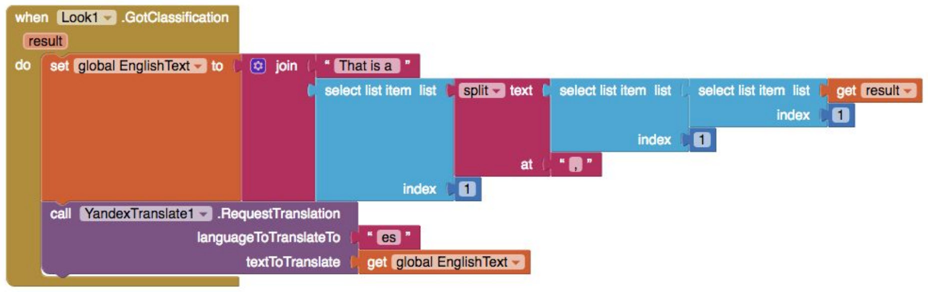
```
when Camera1 .AfterPicture
  image
do  call Look1 .ClassifyImageData
                            image  get image
```

After the camera has taken a picture, “AfterPicture” is called with the picture as a parameter. This picture is passed into the “ClassifyImageData” method of Look.



```
initialize global EnglishText to " "
```

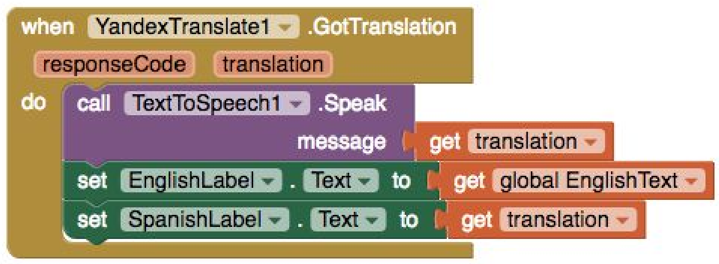
The global “EnglishText” variable will store the string “That is a [object]” and is set in “GotClassification” below.



```
when Look1 .GotClassification
  result
do  set global EnglishText to  join  " That is a "
                                     select list item list  split text  select list item list  select list item list  get result
                                                                                                                index  1
                                                                                          index  1
                                                            at  " , "
                                                      index  1
    call YandexTranslate1 .RequestTranslation
                          languageToTranslateTo  " es "
                                textToTranslate  get global EnglishText
```

The result of “GotClassification” is returned as a list in the form [[class1, confidence1], [class2, confidence2], …, [class10, confidence10]]. class1, etc. are of the form “cellular telephone, cellular phone, cellphone, cell, mobile phone” (a list of one or more items separated by commas). A complete list of possible classes can be found here[link]. This component sets the text “That is a [first item in class1]” to “EnglishText”, passes it to the YandexTranslate component, and requests for it to be translated into Spanish.

We store the entire English sentence in the EnglishText variable so that we can update the EnglishLabel and SpanishLabel simultaneously in the event handler below.


```
when YandexTranslate1 .GotTranslation
  responseCode  translation
do  call TextToSpeech1 .Speak
                  message  get translation
    set EnglishLabel . Text to  get global EnglishText
    set SpanishLabel . Text to  get translation
```

“GotTranslation” of YandexTranslate returns the translation of the text. The TextToSpeech component says the translation out loud. We also update the EnglishLabel and SpanishLabel to display the sentence in English and Spanish.

Variations

- Let the user toggle between classifying objects from camera images and from live video input.

Review

Here are some of the ideas covered in this tutorial:

- The **Look** component can take an image or live video as input and classify the object in the provided image or live video. Its GotClassification event is triggered after ClassifyVideoData or ClassifyImageData has been called, and the classification has successfully completed.
- The **Camera** component can take a picture using the device’s camera and pass the result into ClassifyImageData.
- The **YandexTranslate** component can translate user-provided text into different languages.

Scan the Sample App to your Phone

Download Source Code

Tutorial Version:

- App Inventor 2

Tutorial Difficulty:

- Advanced

Tutorial Type:

- Computer Vision

Teachable Machine Tutorial - Part 1

With this app, you can build a machine that can be trained to recognize classes of images that you submit through your device's camera. This application is based on Google's Teachable Machine.

Getting Started

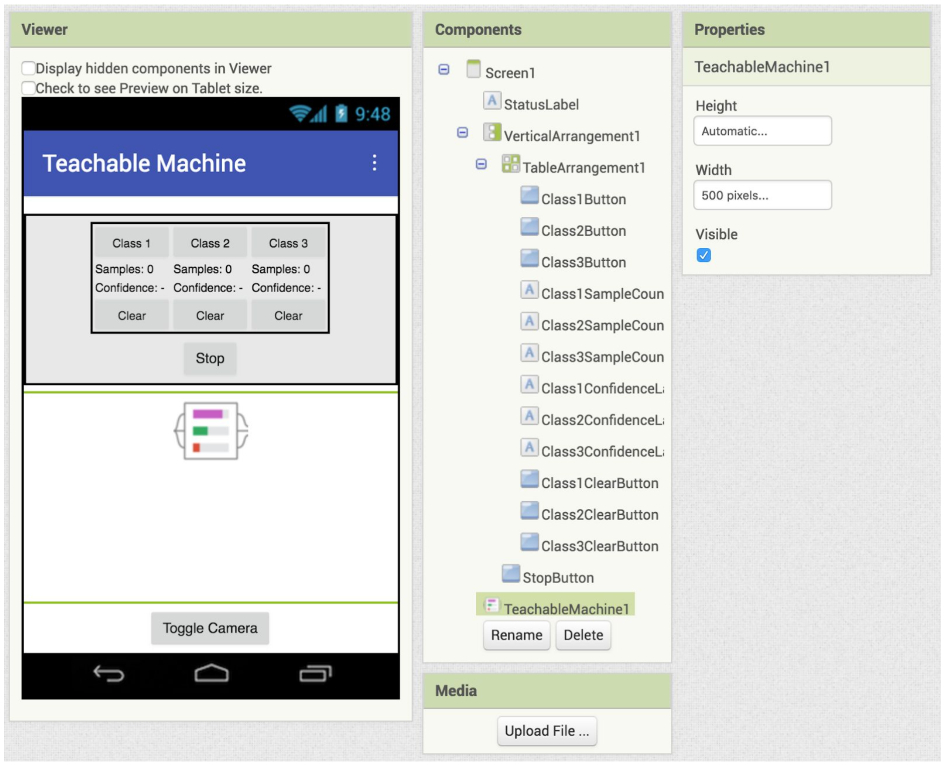
The app demonstrates how to use the **TeachableMachine** component to recognize objects in live video.

It makes use of the following App Inventor components:

- TeachableMachine

User Interface

Here are the components for the Teachable Machine app, as shown in the Component Designer:



```
Screen1
  StatusLabel
  VerticalArrangement1
    TableArrangement1
      Class1Button
      Class2Button
      Class3Button
      Class1SampleCoun
      Class2SampleCoun
      Class3SampleCoun
      Class1ConfidenceL
      Class2ConfidenceL
      Class3ConfidenceL
      Class1ClearButton
      Class2ClearButton
      Class3ClearButton
    StopButton
  TeachableMachine1
```

TeachableMachine1

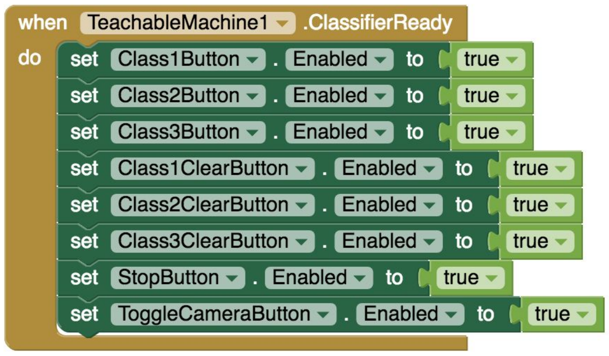
Height: Automatic...

Width: 500 pixels...

Visible: ✓

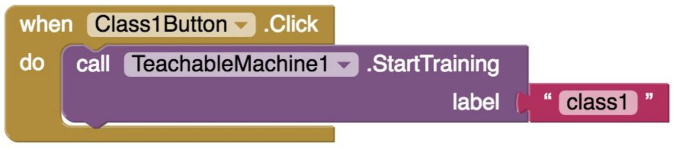
The App’s Behavior

Here is a walkthrough of the blocks that provide the app’s functionality:



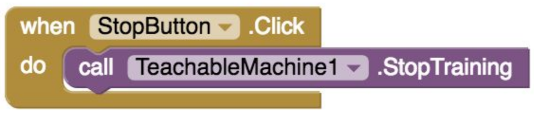
```
when TeachableMachine1 .ClassifierReady
do  set Class1Button . Enabled to true
    set Class2Button . Enabled to true
    set Class3Button . Enabled to true
    set Class1ClearButton . Enabled to true
    set Class2ClearButton . Enabled to true
    set Class3ClearButton . Enabled to true
    set StopButton . Enabled to true
    set ToggleCameraButton . Enabled to true
```

When the classifier is ready to begin training and classifying, the TeachableMachine1.ClassifierReady event handler is called. When this occurs, we enable all the buttons.



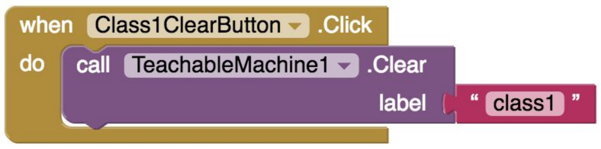
```
when Class1Button .Click
do  call TeachableMachine1 .StartTraining
                              label " class1 "
```

Each class has a separate button to begin training for that class. When the user clicks on the Class1Button, each frame from the live video will be associated with the provided label. In the blocks above, the video frames will be associated with “class1.” The video frames and the associated labels represent the **training data** for Teachable Machine.



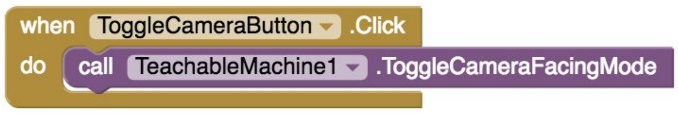
```
when StopButton .Click
do  call TeachableMachine1 .StopTraining
```

When the Stop button is clicked, we call StopTraining to stop collecting video data to train the Teachable Machine.



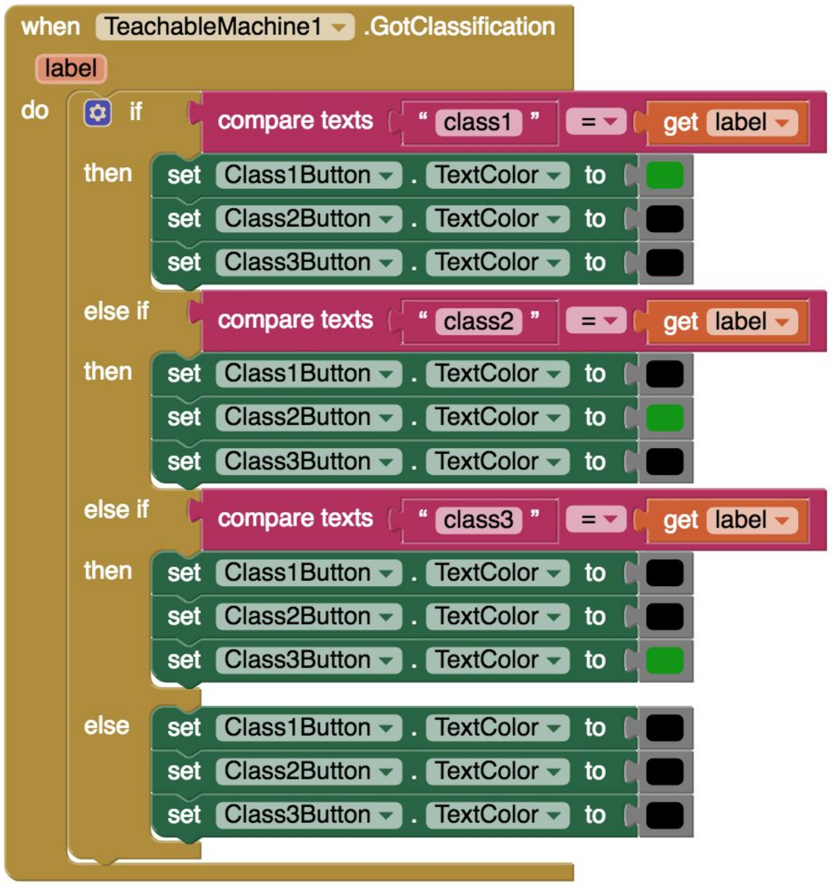
```
when Class1ClearButton .Click
do  call TeachableMachine1 .Clear
                         label " class1 "
```

Each class has a separate clear button to clear the video frames associated with a given label.



```
when ToggleCameraButton .Click
do  call TeachableMachine1 .ToggleCameraFacingMode
```

The toggle button toggles which of the device’s cameras should be used.

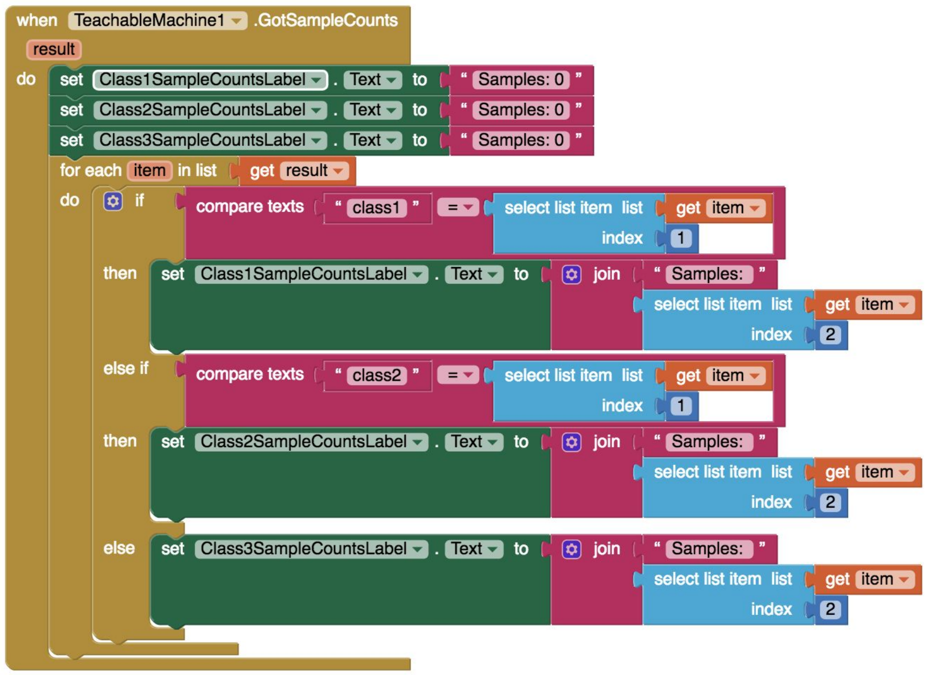


```
when TeachableMachine1 .GotClassification
  label
do  if      compare texts  " class1 "  =  get label
    then    set Class1Button . TextColor to 🟩
            set Class2Button . TextColor to ⬛
            set Class3Button . TextColor to ⬛
    else if compare texts  " class2 "  =  get label
    then    set Class1Button . TextColor to ⬛
            set Class2Button . TextColor to 🟩
            set Class3Button . TextColor to ⬛
    else if compare texts  " class3 "  =  get label
    then    set Class1Button . TextColor to ⬛
            set Class2Button . TextColor to ⬛
            set Class3Button . TextColor to 🟩
    else    set Class1Button . TextColor to ⬛
            set Class2Button . TextColor to ⬛
            set Class3Button . TextColor to ⬛
```

These blocks update which class Teachable Machine thinks the current video image is associated with by changing the text color on the button for that class.

GotClassification is called every time the classification for the current video image is updated. We match up the label with our list of classes and then change the button's text color to green.

It is possible for the label to be an empty string if all classes are cleared, so to handle this case, we include the else clause that sets all the text colors to black.



```
when TeachableMachine1 .GotSampleCounts
  result
do  set Class1SampleCountsLabel . Text to " Samples: 0 "
    set Class2SampleCountsLabel . Text to " Samples: 0 "
    set Class3SampleCountsLabel . Text to " Samples: 0 "
    for each item in list  get result
    do  if   compare texts " class1 " = select list item  list  get item
                                                          index 1
        then set Class1SampleCountsLabel . Text to  join " Samples: "
                                                         select list item  list  get item
                                                                           index 2
        else if  compare texts " class2 " = select list item  list  get item
                                                              index 1
        then set Class2SampleCountsLabel . Text to  join " Samples: "
                                                         select list item  list  get item
                                                                           index 2
        else set Class3SampleCountsLabel . Text to  join " Samples: "
                                                         select list item  list  get item
                                                                           index 2
```

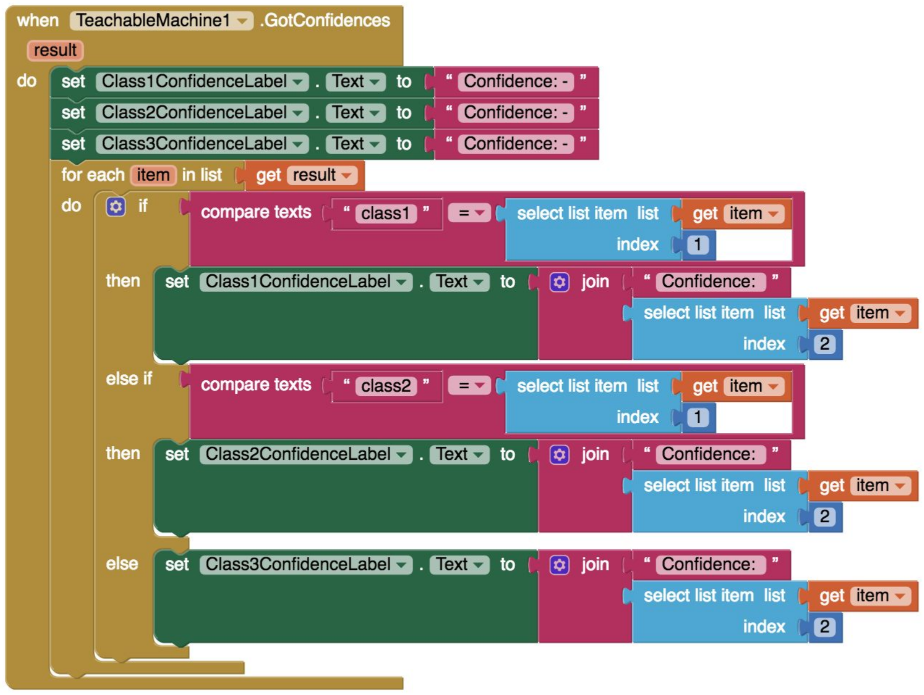
These blocks update the number of sample counts for each class.

GotSampleCounts is called every time the number of samples for the classes we've trained is updated. result is of the form

```
[[label1, sampleCount1], [label2, sampleCount2], ..., [labelN, sampleCountN]].
```

First, the sample counts for each of the labels is set to 0 because GotSampleCounts is called whenever a label is cleared, and only labels with non-zero sample counts are included in result. If we don't reset the samples to 0 at the beginning of the event handler, the sample counts won't reset to 0 when Clear is called.

Next, for each item (e.g. `[label1, sampleCount1]`) in result, we determine which sample counts label to update by comparing the label to our classes. If the label matches our class, we update the sample counts label for that class.



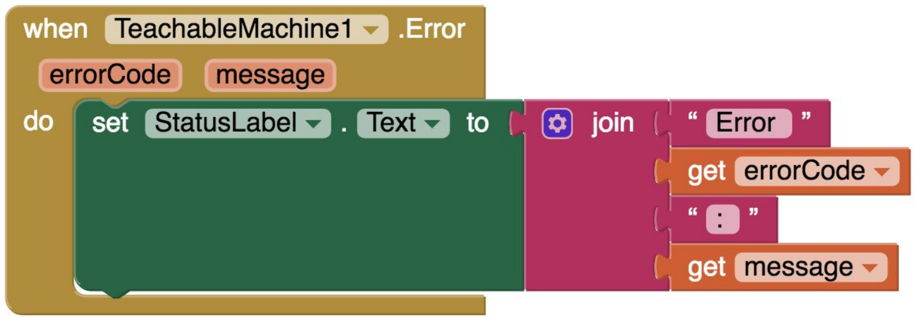
```
when TeachableMachine1 .GotConfidences
  result
do  set Class1ConfidenceLabel . Text to " Confidence: - "
    set Class2ConfidenceLabel . Text to " Confidence: - "
    set Class3ConfidenceLabel . Text to " Confidence: - "
    for each item in list  get result
    do  if    compare texts " class1 " = select list item list get item
                                                  index 1
        then  set Class1ConfidenceLabel . Text to join " Confidence: "
                                                       select list item list get item
                                                                        index 2
        else if compare texts " class2 " = select list item list get item
                                                    index 1
        then  set Class2ConfidenceLabel . Text to join " Confidence: "
                                                       select list item list get item
                                                                        index 2
        else  set Class3ConfidenceLabel . Text to join " Confidence: "
                                                       select list item list get item
                                                                        index 2
```

GotConfidences is called every time the classification for the current video image is updated. The **confidence** is a number between 0 and 1 that reflects how much confidence the processing has in making that classification. result is of the form

```
[[label1, confidence1], [label2, confidence2], …, [labelN, confidenceN]].
```

First, the confidence for each of the labels is reset because GotConfidences is called whenever a label is cleared, and only labels with non-zero sample counts are included in result. If we don't reset the confidences in this event handler, the confidences will not be reset when Clear is called.

Next, for each item (e.g. `[label1, confidence1]`) in result, we determine which confidence label to update by comparing the label to our classes. If the label matches our class, we update the confidence label for that class.



If an error occurs, the Error event handler will be called. These blocks update the status label with the error code and its message. [link list of possible error codes]

Variations

- Add a more visually appealing representation of the confidences. For example, you can use 10 squares (■■■■■■■■■■) to represent a confidence of 1, 5 squares (■■■■■) to represent a confidence of 0.5, etc.
- Add more comprehensive error messages using the error codes returned by the Error event handler. You can also make the error message disappear after 3 seconds.
- Add a fourth class to be trained and update all the blocks correctly.
- Do something with the classification, like gesture controls!

Review

Here are some of the ideas covered in this tutorial:

- The **TeachableMachine** component can collect training data from live video input and learn to associate different video images with different labels. The GotClassification, GotConfidences, and GotSampleCounts event handlers are triggered whenever the current video frame is updated.

Scan the Sample App to your Phone

Download Source Code

Tutorial Version:

- App Inventor 2

Tutorial Difficulty:

- Advanced

Tutorial Type:

- Computer Vision

Teachable Machine Tutorial - Part 2

This app builds on the Teachable Machine app that you created in Teachable Machine Tutorial - Part 1. The user of this app will train the Teachable Machine to recognize hand signs for rock, paper, and scissors so that the user can play a game of rock–paper–scissors against the computer.

Getting Started

The app demonstrates how to use the **TeachableMachine** component to play a game of rock–paper–scissors.

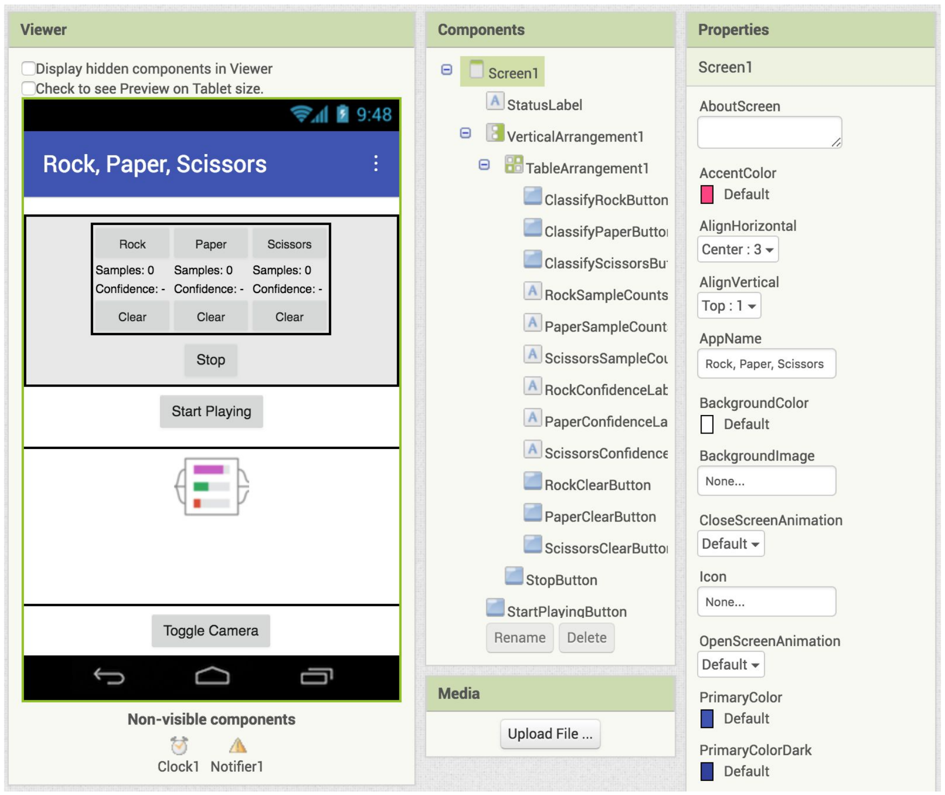
It makes use of the following App Inventor components:

- TeachableMachine
- Clock
- Notifier

This tutorial assumes that you have completed Part 1 of the Teachable Machine Tutorial. You can download a starter template here, which is the application you created in Part 1 with a few components renamed for rock–paper–scissors.

User Interface

Here are the components for the Teachable Machine app, as shown in the Component Designer:



Screen1

AboutScreen

AccentColor: Default

AlignHorizontal: Center : 3

AlignVertical: Top : 1

AppName: Rock, Paper, Scissors

BackgroundColor: Default

BackgroundImage: None...

CloseScreenAnimation: Default

Icon: None...

OpenScreenAnimation: Default

PrimaryColor: Default

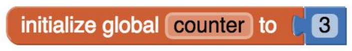
PrimaryColorDark: Default

The notable differences between the user interface in this app and the user interface in Teachable Machine Part 1 include:

- a new Clock component
- a new Notifier component
- a new Start Playing button
- a new GameCountdown label (below the Start Playing button)

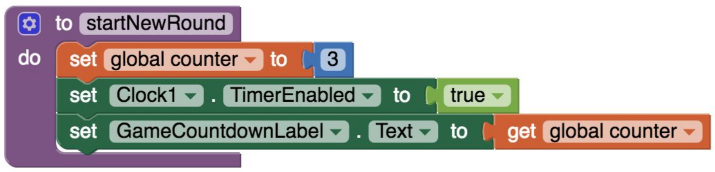
The App's Behavior

Here is a walkthrough of the blocks that provide the app's functionality:



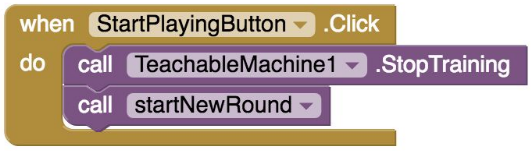
```
initialize global counter to 3
```

counter is a global variable that keeps track of the time left before the rock–paper–scissors round takes place. This counter is decreased by 1 every time the clock event handler fires.



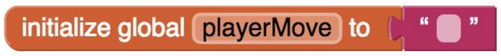
```
to startNewRound
do  set global counter to 3
    set Clock1 . TimerEnabled to true
    set GameCountdownLabel . Text to get global counter
```

startNewRound is a procedure that is called to start a new round of rock–paper–scissors. It resets the counter and countdown label and enables the clock.



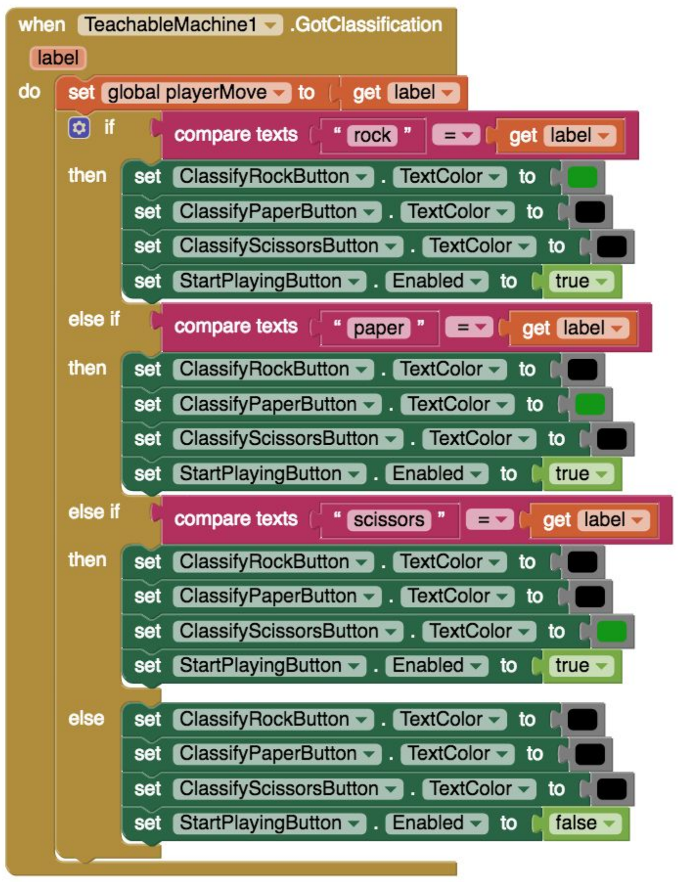
```
when StartPlayingButton .Click
do  call TeachableMachine1 .StopTraining
    call startNewRound
```

When the StartPlayingButton is clicked, we stop training TeachableMachine1 and start a new round of rock–paper–scissors.



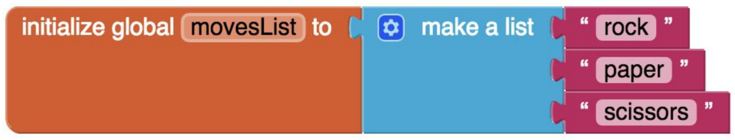
```
initialize global playerMove to " "
```

playerMove is a global variable that stores the move of the player (“rock”, “paper”, or “scissors”) as determined by the TeachableMachine component. playerMove is updated every time the GotClassification event handler is fired.



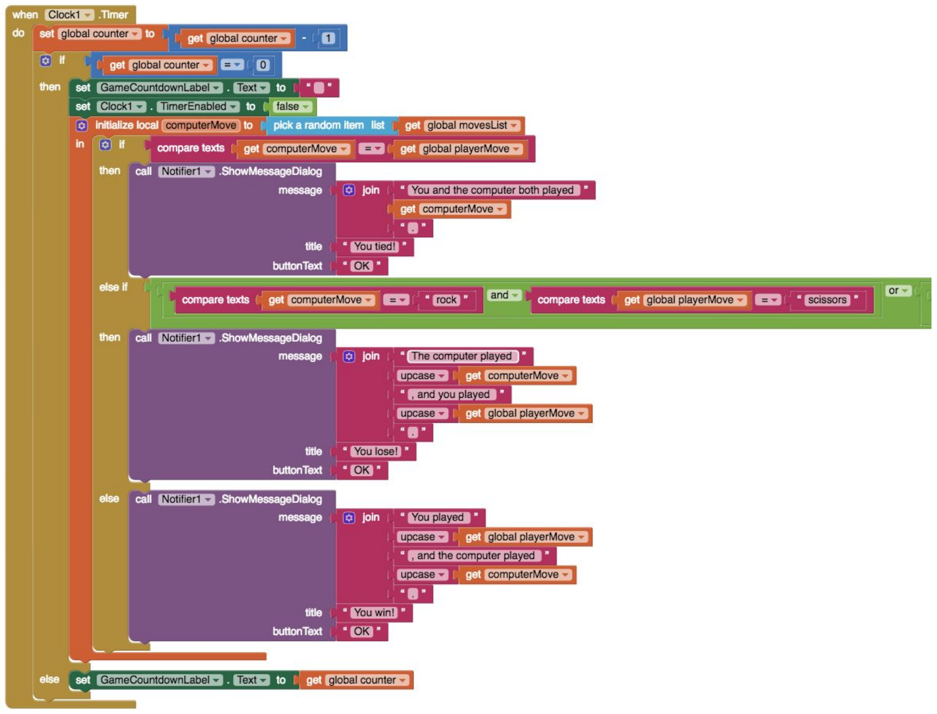
```
when TeachableMachine1 .GotClassification
  label
do  set global playerMove to  get label
    if  compare texts  " rock "  =  get label
    then  set ClassifyRockButton . TextColor to 🟩
          set ClassifyPaperButton . TextColor to ⬛
          set ClassifyScissorsButton . TextColor to ⬛
          set StartPlayingButton . Enabled to true
    else if  compare texts  " paper "  =  get label
    then  set ClassifyRockButton . TextColor to ⬛
          set ClassifyPaperButton . TextColor to 🟩
          set ClassifyScissorsButton . TextColor to ⬛
          set StartPlayingButton . Enabled to true
    else if  compare texts  " scissors "  =  get label
    then  set ClassifyRockButton . TextColor to ⬛
          set ClassifyPaperButton . TextColor to ⬛
          set ClassifyScissorsButton . TextColor to 🟩
          set StartPlayingButton . Enabled to true
    else  set ClassifyRockButton . TextColor to ⬛
          set ClassifyPaperButton . TextColor to ⬛
          set ClassifyScissorsButton . TextColor to ⬛
          set StartPlayingButton . Enabled to false
```

Our GotClassification handler from Part 1 of the tutorial should be modified to enable or disable the StartPlayingButton depending on if any classes have been categorized. GotClassification also sets playerMove to the current label.



```
initialize global movesList to  make a list  " rock "
                                             " paper "
                                             " scissors "
```

movesList is a list containing the three possible moves in game of rock–paper–scissors.



```
when Clock1 .Timer
do  set global counter to  get global counter - 1
    if  get global counter = 0
    then  set GameCountdownLabel . Text to " "
          set Clock1 . TimerEnabled to false
          initialize local computerMove to  pick a random item list  get global movesList
          in  if  compare texts  get computerMove = get global playerMove
              then  call Notifier1 .ShowMessageDialog
                        message  join  " You and the computer both played "
                                       get computerMove
                                       " . "
                        title  " You tied! "
                        buttonText  " OK "
              else if  compare texts  get computerMove = " rock "  and  compare texts  get global playerMove = " scissors "  or
              then  call Notifier1 .ShowMessageDialog
                        message  join  " The computer played "
                                       upcase  get computerMove
                                       " , and you played "
                                       upcase  get global playerMove
                                       " . "
                        title  " You lose! "
                        buttonText  " OK "
              else  call Notifier1 .ShowMessageDialog
                        message  join  " You played "
                                       upcase  get global playerMove
                                       " , and the computer played "
                                       upcase  get computerMove
                                       " . "
                        title  " You win! "
                        buttonText  " OK "
    else  set GameCountdownLabel . Text to  get global counter
```

Each time the Clock1.Timer goes off, we decrease the counter by 1. If the counter value is equal to 0, we disable the timer. We then choose a random move for the computer by picking a random item from movesList. We then compare the computerMove to the playerMove to determine the winner of the game.

We use Notifier1.ShowMessageDialog to display the winner of the game, as well as information about what move the computer played and what move the user played.

If the counter has not reached 0, we simply update GameCountdownLabel.

Variations

- Keep track of the score between you and the computer.
- Use the TextToSpeech component to announce the winner of the round.
- Make a variation of rock–paper–scissors with additional classes.
- Save and load the models for rock–paper–scissors.

Review

Here are some of the ideas covered in this tutorial:

- The **TeachableMachine** component can collect training data from live video input and learn to associate different video images with different labels. These labels can be used for a wide variety of applications, including playing games.

Scan the Sample App to your Phone

Download Source Code

Tutorial Version:

- App Inventor 2

Tutorial Difficulty:

- Advanced

Tutorial Type:

- Computer Vision

Emoji Scavenger Hunt Tutorial

Getting Started

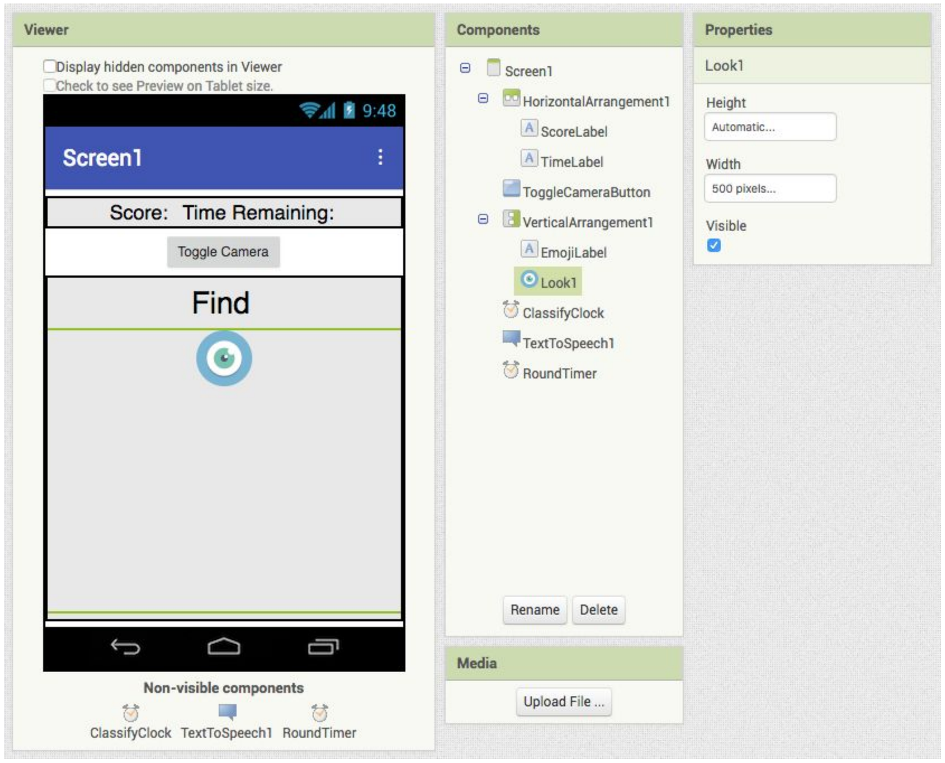
The app demonstrates how to use the **Look** component to recognize objects in an image taken by the phone camera.

It makes use of the following App Inventor components:

- Look

User Interface

Here are the components for the Emoji Scavenger Hunt app, as shown in the Component Designer:



The user interface consists of a label for the score, a label for the remaining time, a button to toggle the camera, a label for the emoji to look for, and the Look component.

There are also three non-visible components: two Clocks and one TextToSpeech component.

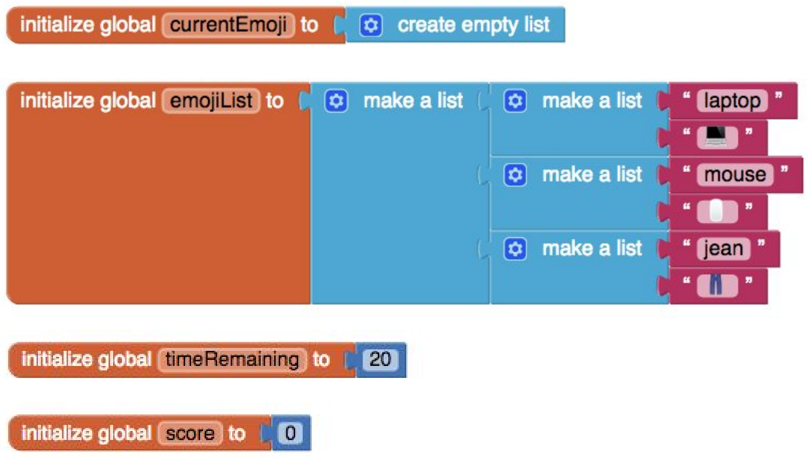
The width of the Look component is set to 500 pixels instead of “Automatic” to prevent it from filling up the entire screen.

The ClassifyClock component should be fired every 3 seconds to classify the video data, so the TimerInterval of ClassifyClock is set to 3000. The RoundTimer component should be fired every second to update the time remaining, so the TimerInterval of RoundTimer is set to 1000.

The App’s Behavior

Here are the blocks for the Emoji Scavenger Hunt app:

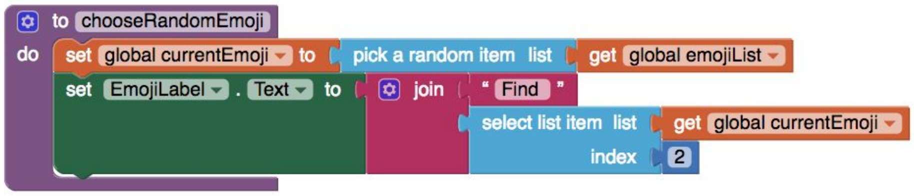
Classify live video



```
initialize global currentEmoji to  create empty list

initialize global emojiList to  make a list  make a list  " laptop "
                                                          " 💻 "
                                             make a list  " mouse "
                                                          " 🖱 "
                                             make a list  " jean "
                                                          " 👖 "

initialize global timeRemaining to  20

initialize global score to  0
```

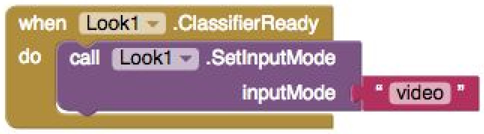
These blocks are used to initialize the global variables.

- **currentEmoji** = a list whose first element is the name of the current round’s emoji and whose second element is the emoji itself (e.g. 💻, 🖱, or 👖)
- **emojiList** = the list of all emojis and their corresponding names
- **timeRemaining** = the number of seconds remaining for the current round
- **score** = the player’s score



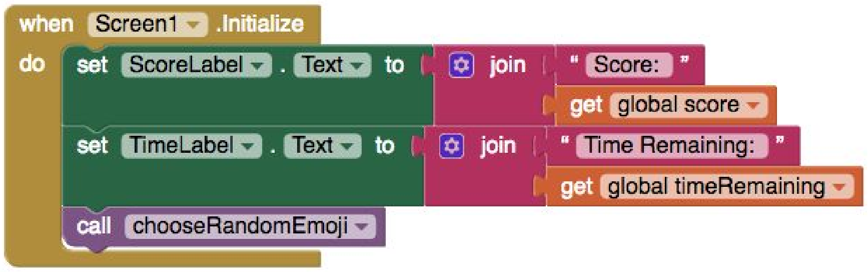
```
to chooseRandomEmoji
do  set global currentEmoji to  pick a random item  list  get global emojiList
    set EmojiLabel . Text to  join  " Find "
                                    select list item  list  get global currentEmoji
                                                      index  2
```

This procedure picks a random list out of emojiList and sets it to currentEmoji. The EmojiLabel is then set to be “Find [the second element of emojiList]”, which is the emoji that the user should look for.



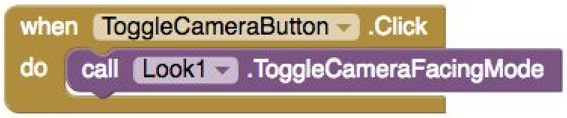
```
when Look1 .ClassifierReady
do  call Look1 .SetInputMode
                   inputMode  " video "
```

This block changes the input mode of the Look component to video. By default, the Look component takes in image data, so to classify live video, the input mode needs to be changed. The “ClassifierReady” event is called when the the Look component is ready to be used.



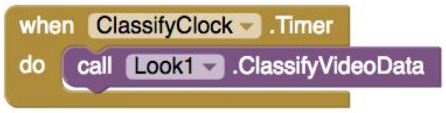
```
when Screen1 .Initialize
do  set ScoreLabel . Text to  join  " Score: "
                                    get global score
    set TimeLabel . Text to  join  " Time Remaining: "
                                   get global timeRemaining
    call chooseRandomEmoji
```

When the screen is initialized, we want the score label and the time remaining label to display the correct values.



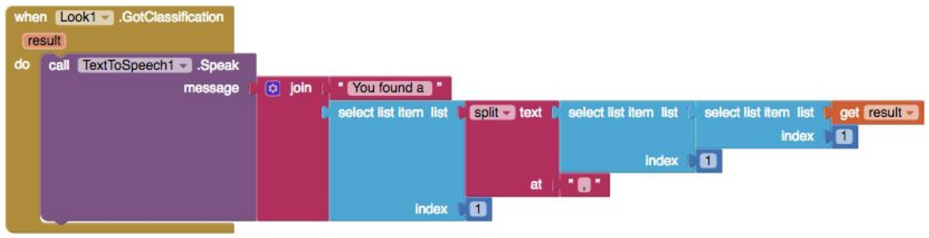
```
when ToggleCameraButton .Click
do  call Look1 .ToggleCameraFacingMode
```

When the ToggleCameraButton is clicked, the Look component should toggle the direction of the camera.



```
when ClassifyClock .Timer
do  call Look1 .ClassifyVideoData
```

Every three seconds when the ClassifyClock timer fires, the Look component should classify the current live video frame.

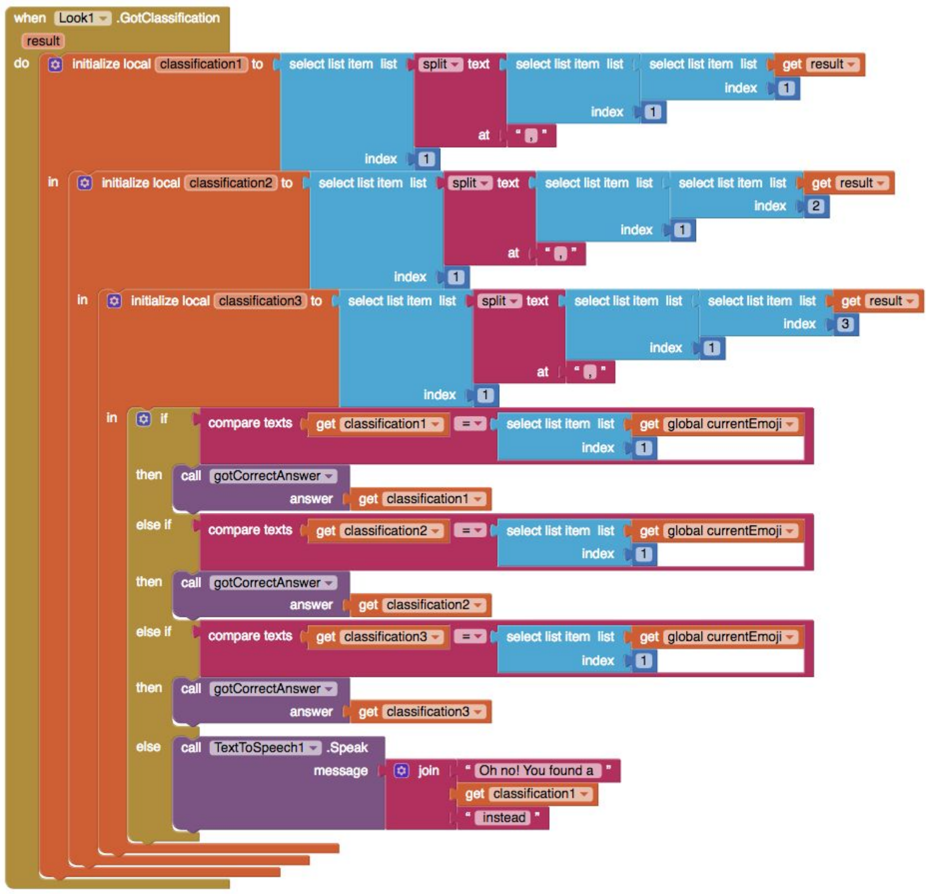


```
when Look1 .GotClassification
  result
do call TextToSpeech1 .Speak
     message  join  " You found a "
                    select list item  list  split text  select list item  list  select list item  list  get result
                                                                                                   index  1
                                                                          index  1
                                           at  " , "
                                      index  1
```

The result of “GotClassification” is returned as a list in the form [[class1, confidence1], [class2, confidence2], …, [class10, confidence10]]. class1, etc. are of the form “cellular telephone, cellular phone, cellphone, cell, mobile phone” (a list of one or more items separated by commas). A complete list of possible classes can be found here[link].

For now, we will just let the TextToSpeech component say what we’ve found. This component passes the text “You found is a [first item in class1]” to TextToSpeech.

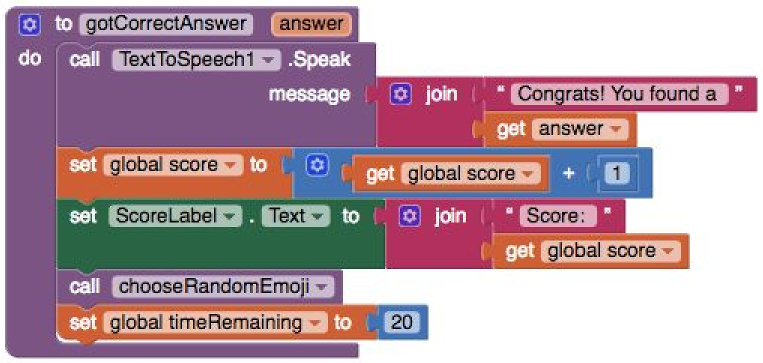
Add score and time remaining



```
when Look1 .GotClassification
  result
do  initialize local classification1 to  select list item  list  split text  select list item  list  select list item  list  get result
                                                                                                                      index  1
                                                                                                  index  1
                                                                              at  " , "
                                                            index  1
  in  initialize local classification2 to  select list item  list  split text  select list item  list  select list item  list  get result
                                                                                                                         index  2
                                                                                                     index  1
                                                                                 at  " , "
                                                               index  1
    in  initialize local classification3 to  select list item  list  split text  select list item  list  select list item  list  get result
                                                                                                                           index  3
                                                                                                       index  1
                                                                                   at  " , "
                                                                 index  1
      in  if  compare texts  get classification1  =  select list item  list  get global currentEmoji
                                                                         index  1
          then  call gotCorrectAnswer
                         answer  get classification1
          else if  compare texts  get classification2  =  select list item  list  get global currentEmoji
                                                                              index  1
          then  call gotCorrectAnswer
                         answer  get classification2
          else if  compare texts  get classification3  =  select list item  list  get global currentEmoji
                                                                              index  1
          then  call gotCorrectAnswer
                         answer  get classification3
          else  call TextToSpeech1 .Speak
                         message  join  " Oh no! You found a "
                                        get classification1
                                        " instead "
```

For this method, we replace the previous TextToSpeech block in “GotClassification” with some additional logic to test whether the result matches currentEmoji.

Three local variables are created, and they correspond to the top three classes from the result instead of only the first one. If any one of these equals the currentEmoji, we call gotCorrectAnswer; otherwise, TextToSpeech says what the classification is.

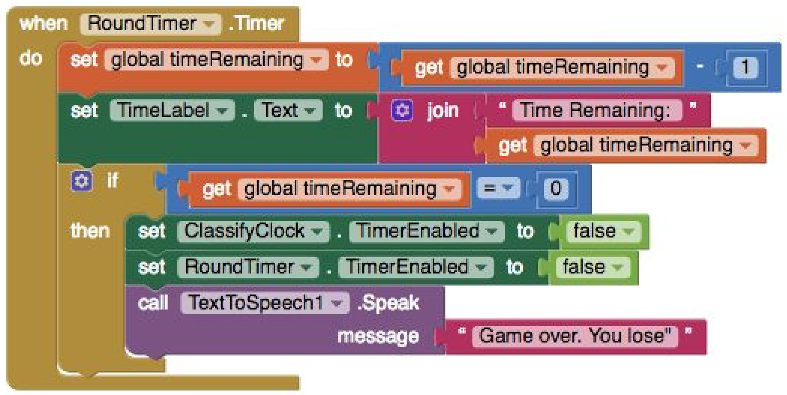


```
to gotCorrectAnswer answer
do  call TextToSpeech1 .Speak
                       message  join  " Congrats! You found a "
                                      get answer
    set global score to  get global score + 1
    set ScoreLabel . Text to  join  " Score: "
                                    get global score
    call chooseRandomEmoji
    set global timeRemaining to 20
```

This procedure is called when the image classification returns the correct answer (i.e. the classification matches the label of the current emoji). The correct answer is passed into the answer parameter.

The TextToSpeech component says congratulations and reports that the player has found the emoji. The score variable is updated, and the score label is updated to reflect the new value.

A new emoji is randomly selected for the new round, and the time remaining is reset to 20 seconds.



```
when RoundTimer .Timer
do  set global timeRemaining to  get global timeRemaining - 1
    set TimeLabel . Text to  join  " Time Remaining: "
                                   get global timeRemaining
    if   get global timeRemaining = 0
    then set ClassifyClock . TimerEnabled to false
         set RoundTimer . TimerEnabled to false
         call TextToSpeech1 .Speak
                            message  " Game over. You lose" "
```

The RoundTimer fires every second. Every time it fires, we subtract one second from the timeRemaining variable and update the time remaining label. If timeRemaining has reached 0, then we stop ClassifyClock and RoundTimer and announce game over.

Variations

- Add more possible emojis!

- Add a button to restart the game.
- Add a button to start the next round instead of starting it immediately.
- Add levels so that when the user's score passes a certain number, the emojis become more difficult to find.

Review

Here are some of the ideas covered in this tutorial:

- The **Look** component can take an image or live video as input and classify the object in the provided image or live video. Its GotClassification event is triggered after ClassifyVideoData or ClassifyImage has been called, and the classification has successfully completed.

Scan the Sample App to your Phone

Download Source Code

Tutorial Version:

- App Inventor 2

Tutorial Difficulty:

- Advanced

Tutorial Type:

- Computer Vision

Bibliography

[1] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. O'Reilly Media, Inc. (2019).

[2] Scratch. `https://scratch.mit.edu/` (2019).

[3] Cognimates. `http://cognimates.me/home/` (2019).

[4] IBM Watson Products and Services. `https://www.ibm.com/watson/products-services/` (2019).

[5] Machine Learning for Kids. `https://machinelearningforkids.co.uk/` (2019).

[6] MIT App Inventor. `http://appinventor.mit.edu/explore/` (2019).

[7] About Us. `http://appinventor.mit.edu/explore/about-us.html` (2019).

[8] Deep Learning on AWS. `https://aws.amazon.com/deep-learning/` (2019).

[9] Cloud AI. `https://cloud.google.com/products/ai/` (2019).

[10] TensorFlow.js. `https://www.tensorflow.org/js/` (2019).

[11] TensorFlow. `https://www.tensorflow.org/` (2019).

[12] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. `https://arxiv.org/pdf/1704.04861.pdf` (2017).

[13] Teachable Machine. `https://teachablemachine.withgoogle.com/` (2019).

[14] Tesseract.js. http://tesseract.projectnaptha.com/ (2019).

[15] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, and Kevin Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. https://arxiv.org/pdf/1611.10012.pdf (2017).

[16] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single Shot MultiBox Detector. https://arxiv.org/pdf/1512.02325.pdf (2016).

[17] George Papandreou, Tyler Zhu, Liang-Chieh Chen, Spyros Gidaris, Jonathan Tompson, and Kevin Murphy. PersonLab: Person Pose Estimation and Instance Segmentation with a Bottom-Up, Part-Based, Geometric Embedding Model. https://arxiv.org/pdf/1803.08225.pdf (2018).

[18] Pete Warden. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. https://arxiv.org/pdf/1804.03209.pdf (2018).

[19] Emoji Scavenger Hunt. https://emojiscavengerhunt.withgoogle.com/ (2019).