Personal Image Classifier

MIT
APP INVENTOR

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. Students will train an image classification model using a web-based interface. Students will then upload the model and use it in a pre-made App Inventor app to see how the model performs in classifying images. Finally, students will create a new mobile app using App Inventor. This app will be a game that allows users to earn points by matching a given facial expression. Students will again use their trained model as the classifier within the app. Students will use MIT App Inventor's machine learning extension called the PersonalImageClassifier when creating this app.

2. Learning Objectives

After completing this unit, students will be able to:

1. Gain a basic understanding of machine learning.
2. Train, test, and revise an image classification model.
3. Use the PersonalImageClassifier extension and their trained model in an image classification app using App Inventor.
4. Use their model in a more interesting game app.

3. Mapping with the CSTA Standards

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

Middle School 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-DA-09	Refine computational models based on the data they have generated. [C] DA: Inference & Models [P] Creating (5.3), Abstraction (4.4)	Students train a model to classify images.
2-AP-11	Create clearly named variables that represent different data types and perform operations on their values. [C] AP: Variables [P] Creating (5.1, 5.2)	Students use several variables to hold information in the app.
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)	PIC extension is used in mobile app to classify images.
2-IC-20	Discuss issues of bias and accessibility in the design of existing technologies. [C] IC: Culture [P] Inclusion (1.2)	Bias is discussed as part of training an image classification model.

High School Standards

3A-CS-01	Explain how abstractions hide the underlying implementation details of computing systems embedded in everyday objects. [C] CS: Devices [P] Abstraction (4.1)	Students should understand how the Personal Image Classifier and PIC extension hide underlying complex code.
3A-DA-13	Create computational models that represent the relationships among different elements of data collected from a phenomenon or process. [C] DA: Inference & Models [P] Abstraction (4.4)	Students train an image classification model.
3A-AP-19	Create artifacts by using procedures within a program, combinations of data and procedures, or independent but interrelated programs. [C] AP: Modularity [P] Creating (5.2)	Simple procedure is used in the program.
3A-IC-26	Test and refine computational artifacts to reduce bias and equity deficits. [C] IC: Culture [P] Inclusion (1.2)	Students refine image classification model to potentially reduce bias.
3B-AP-08	Describe how artificial intelligence drives many software and physical systems. [C] AP: Algorithms [P] Communicating (7.2)	Discussion of AI in society.

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 your students won't be creating apps until Lesson 2, it is important to test whether the apps will work at all beforehand. The extension requires the machine learning code to execute directly on the mobile device, and some older devices cannot handle it due to hardware limitations. Follow the instructions below under Lesson 2 to test whether the PIC and Expression Match apps will work on your classroom mobile devices. Here is a list of device/OS pairs that are known to work with this extension.

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 open and interact with Personal Image Classifier in a browser. Note that this first lesson needs neither App Inventor nor mobile devices.

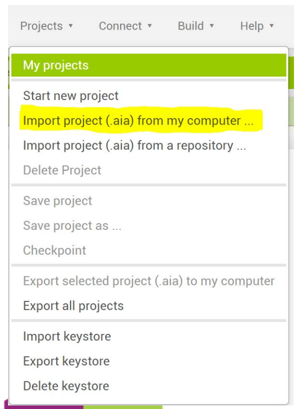
Lesson 2

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 PIC Extension (noted in previous section). Certain Chromebooks may also be used to run both App Inventor and the AI Companion in lieu of mobile devices. Here is a list of device/OS pairs that are known to work with this extension.

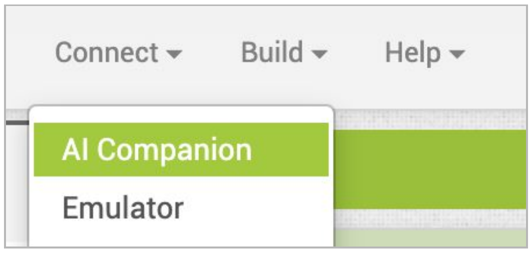
Make sure that the mobile devices have the latest updates and have the MIT AI2 Companion installed so that students can run the app on the device (Google Play MIT AI2 Companion link).

If your device is not on our compatibility list, you should check beforehand if the PIC extension will work on your device. (Older devices might not be able to run the extension.) You can download this LookTest.apk file and install it on your device(s). Skip to the **Test App** instructions below to see how to test the app.

You can also test using the completed project file and the AI Companion. Start by downloading the completed project (link) onto your computer. Then, open App Inventor and import the aia file 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.

Test App

Click the Classify Video button. You should see video appear in the app. Click again and you should see the classification appear in the label at the top of the screen. If video does not appear, it may mean that your device’s hardware is not sufficient to run the extension. If that is the case, you may not be able to run Lessons 2 and 3 with your students, because you won’t be able to run the apps in class.

6. Lesson Plan (3 x 45 minutes)

This unit consists of three 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. If students have already learned about the App Inventor Look extension, some of this may be skipped.
25 min	**Train a Model with the Personal Image Classifier** Students open Web browsers on their computers and use https://classifier.appinventor.mit.edu/oldpic/ to create a model of different facial expressions. Students follow *PIC Student Guide Lesson 1*.
10 min	**Wrap-up Discussion** Discuss how students think their models performed. Ask about what made their models more or less accurate. What was a good number of images to train the model with?

Lesson 2

Time	Activity
10 min	**Introduction to Activity** Explain to students they will use the model made in the previous lesson in a prebuilt app. If students need more time to train their model, give them another 5-10 minutes.
15 min	**Run and Test PIC App** 1. Download and import the PIC template aia in App Inventor, or use the sidebar tutorial to open the template automatically. 2. Students upload their model from Lesson 1 to the template. 3. Students test the app and their model using the PIC app.
15 min	**Wrap-up Discussion** 1. Discuss how their app worked based on accuracy, limitations, and discuss ways to improve their models. 2. Students may go back to the Classifier website to update their models based on their success in the apps.

Lesson 3

Time	Activity
5 min	**Introduction to Lesson** Explain to students they will create a game app that uses their trained model to assess users' ability to match a given expression.

30 min	**Coding Expression Match App** 1. Students may follow the tutorial using the pdf student guide, or using the sidebar tutorial. 2. Students will create a game app called Expression Match, where the user tries to match a given expression (happy, sad, surprised) and gains points depending on how well they can match the model. 3. Students test the app to assess how well the model works as a game.
10 min	**Wrap-up Discussion** 1. Discuss how their game app worked based on accuracy and different users. a. How might bias in the training model affect the outcome of a game? b. Is it fair? c. Does it matter who is playing the game? 2. Students may go back to the Classifier website to update their models based on how well the model performs.

7. Assessment

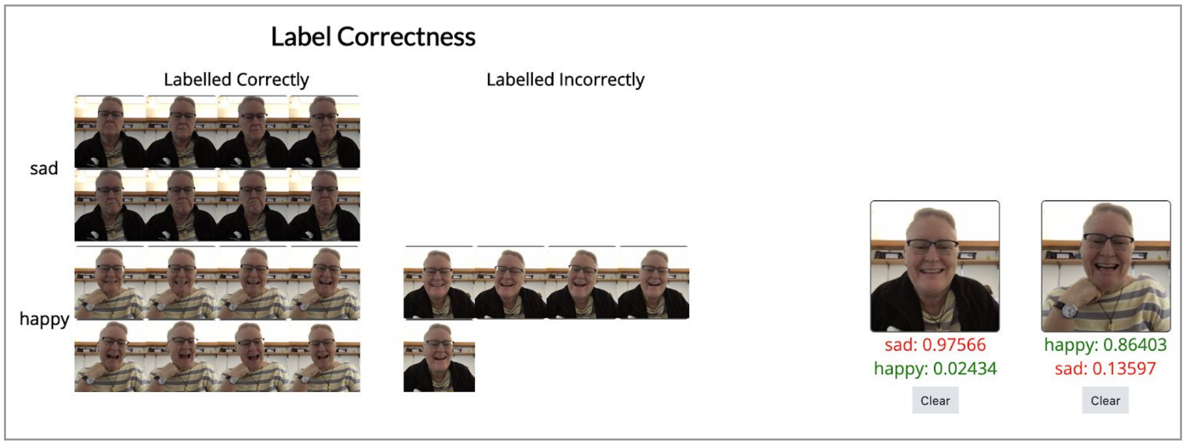
Multiple-choice questions to test understanding after the unit

1. You train a model on facial expressions like happy, sad, and surprised with on https://classifier.appinventor.mit.edu/oldpic/, and build an expression matching game with your model. You play the game and score very highly. You give the app to a friend, and they score very low. Why might that happen?

   A. The model might need more training to be more accurate.
   B. The model is biased because you trained it with pictures of yourself.
   C. Your friend might not be very skilled at making expressions.
   D. All of the above.

   (Answer: **D**. A and B especially … and C is always a possibility!)

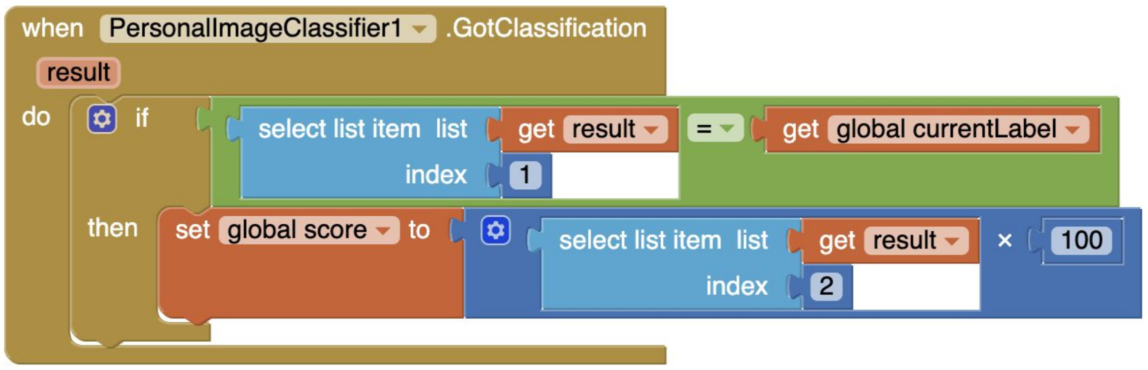
2. You train a model for 2 labels (happy, sad) using https://classifier.appinventor.mit.edu/oldpic/and get several incorrect labels during the testing phase. Looking at the results below, it labels several "happy" faces incorrectly. What might be the problem?



A. Not enough test cases.
B. You need a third label in addition to happy and sad.
C. All sad images include a black jacket; all happy images do not, so it classifies a black jacket as sad.
D. There is no problem. You can expect some incorrect labelling during testing.

Answer: **C**

3. Your code for an expression matching game includes the following. However, your score never changes from zero. Why are you never getting any score?



```
when PersonalImageClassifier1 .GotClassification
  result
do  if    select list item  list  get result    =    get global currentLabel
                           index  1
    then  set global score to    select list item  list  get result    ×  100
                                                  index  2
```

A. In the set global score block, the index for select list item should be 1, not 2.

B. You are missing the for each item block, so it is comparing the currentLabel to a list, and it will never be equal.

C. You should just be comparing if result=currentLabel, and not using select list item.

D. Your model needs better training.

(Answer: **B**; the if statement should be inside a for each item in list, and result in the if statement should be changed to item.)

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.

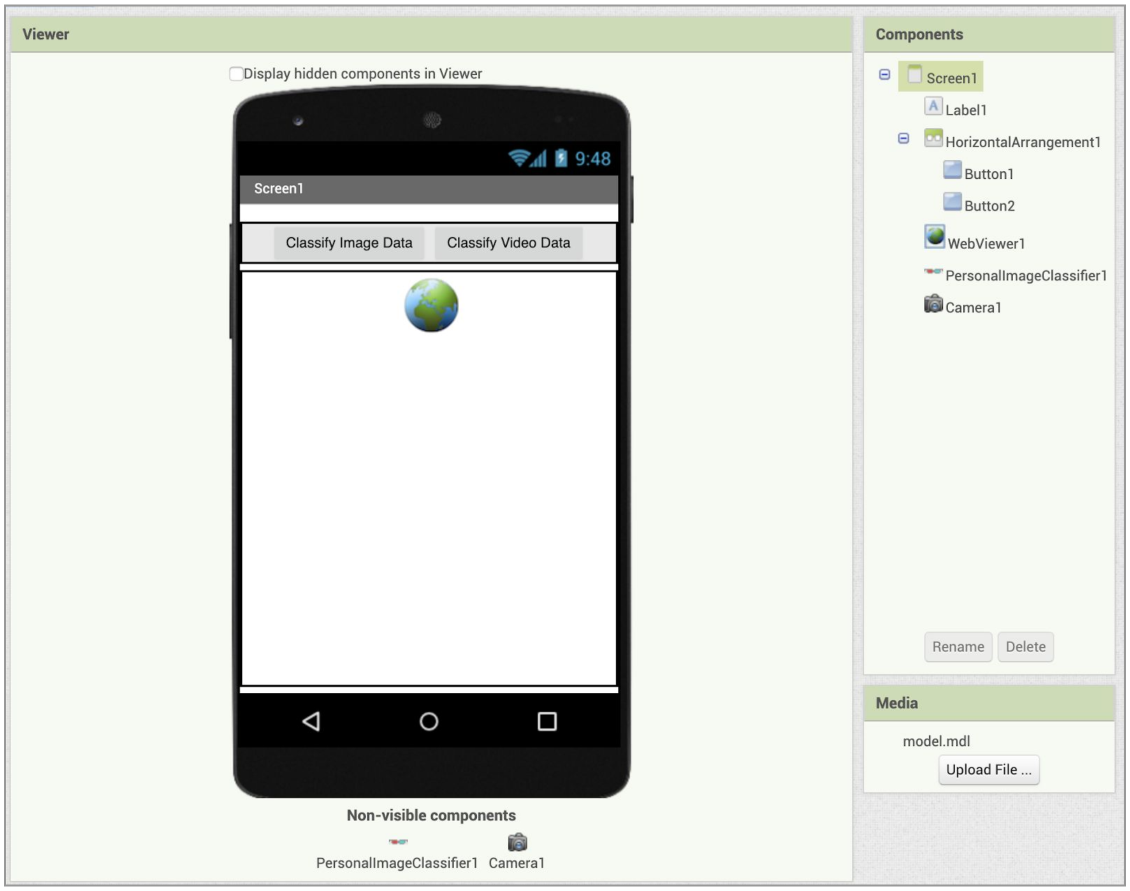
After completion of this unit, I think…	**Disagree**	**Somewhat disagree**	**Neutral**	**Somewhat agree**	**Agree**
Learning how to make apps makes me want to learn more about coding.					
I feel more connected to the technology around me when I make apps.					
I am excited to share this app with friends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

MIT APP INVENTOR

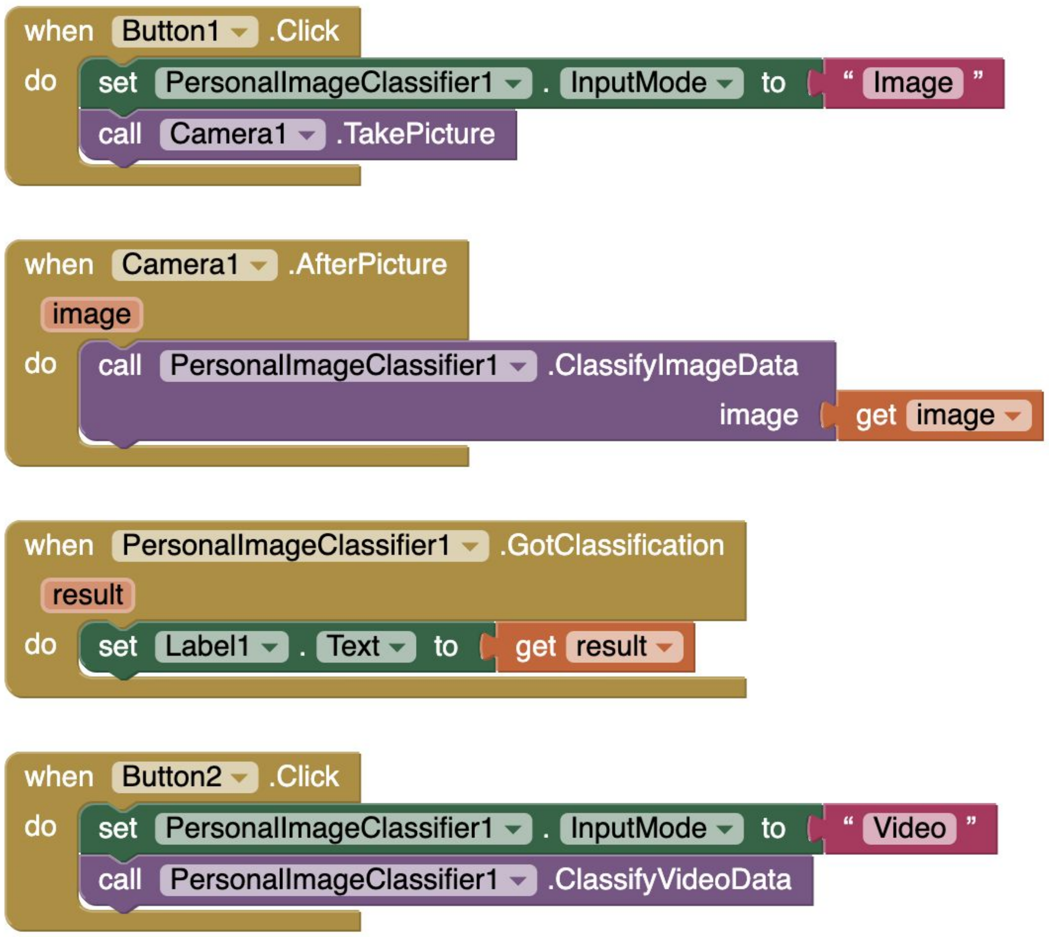
8. Screen Design and Code Blocks

Below are images of the completed PIC App (Lesson 2).

Designer:



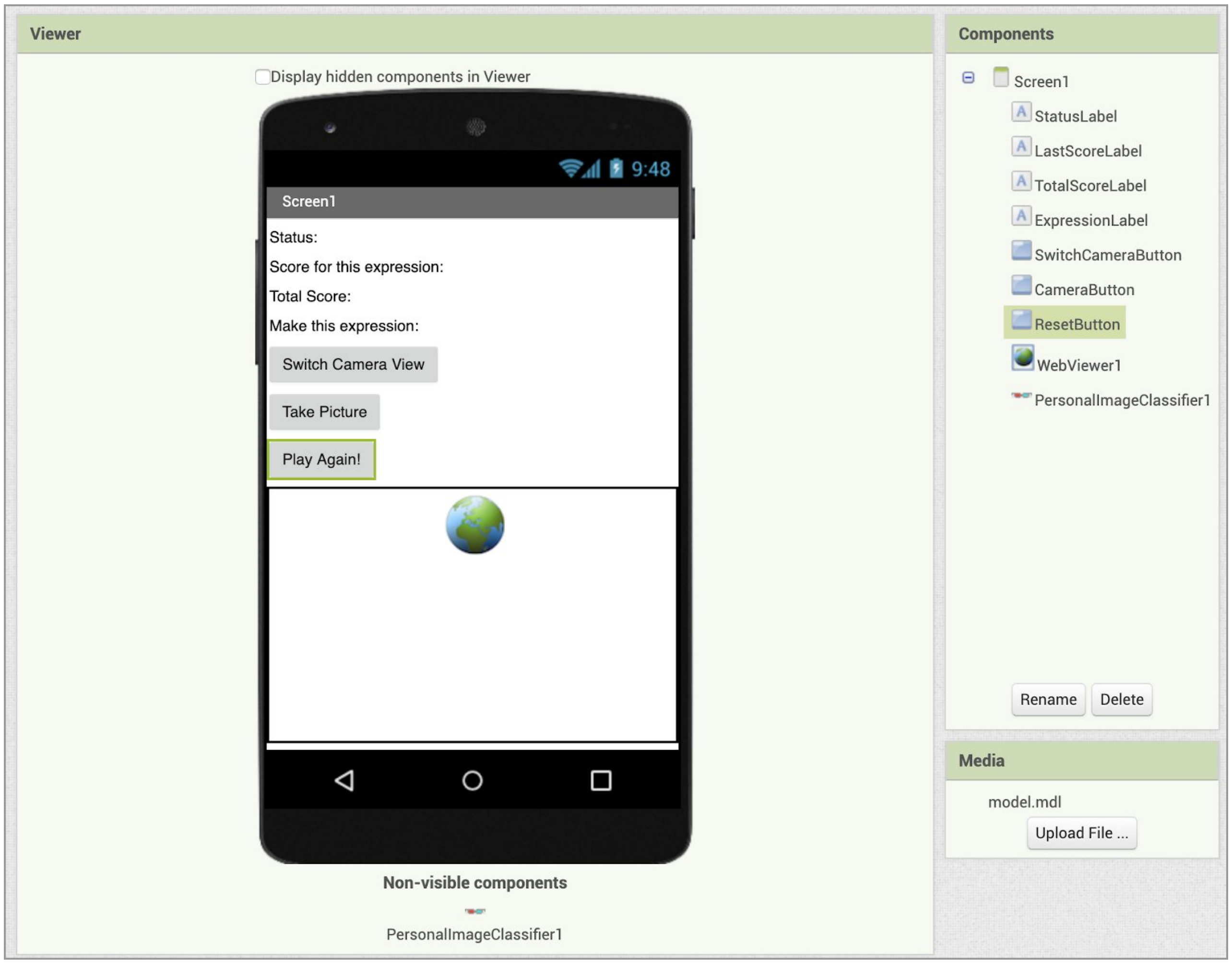
Blocks:



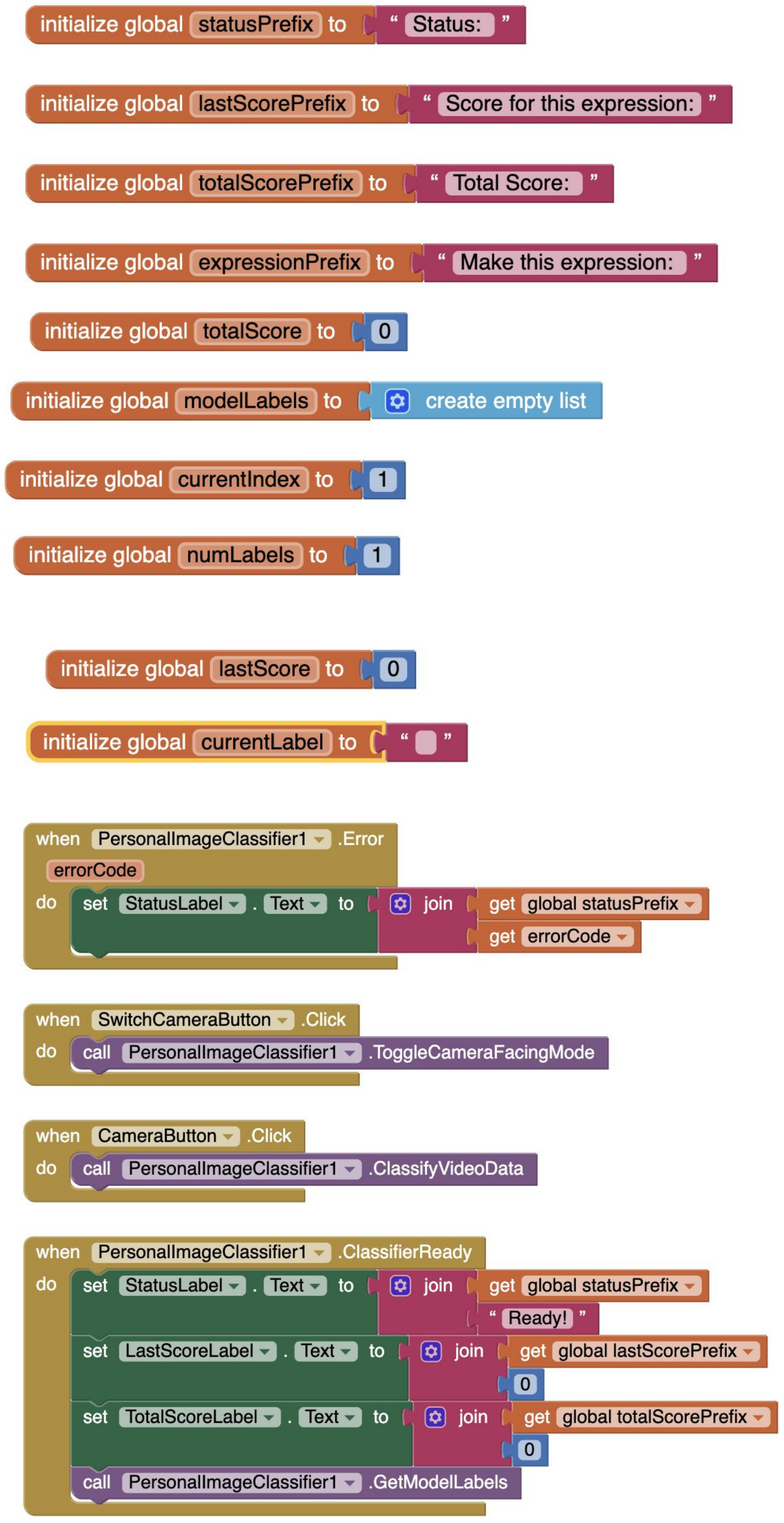
```
when Button1 .Click
do  set PersonalImageClassifier1 . InputMode to " Image "
    call Camera1 .TakePicture

when Camera1 .AfterPicture
 image
do  call PersonalImageClassifier1 .ClassifyImageData
                                       image  get image

when PersonalImageClassifier1 .GotClassification
 result
do  set Label1 . Text to  get result

when Button2 .Click
do  set PersonalImageClassifier1 . InputMode to " Video "
    call PersonalImageClassifier1 .ClassifyVideoData
```

Below are images of the completed Expression Match App (Lesson 3).

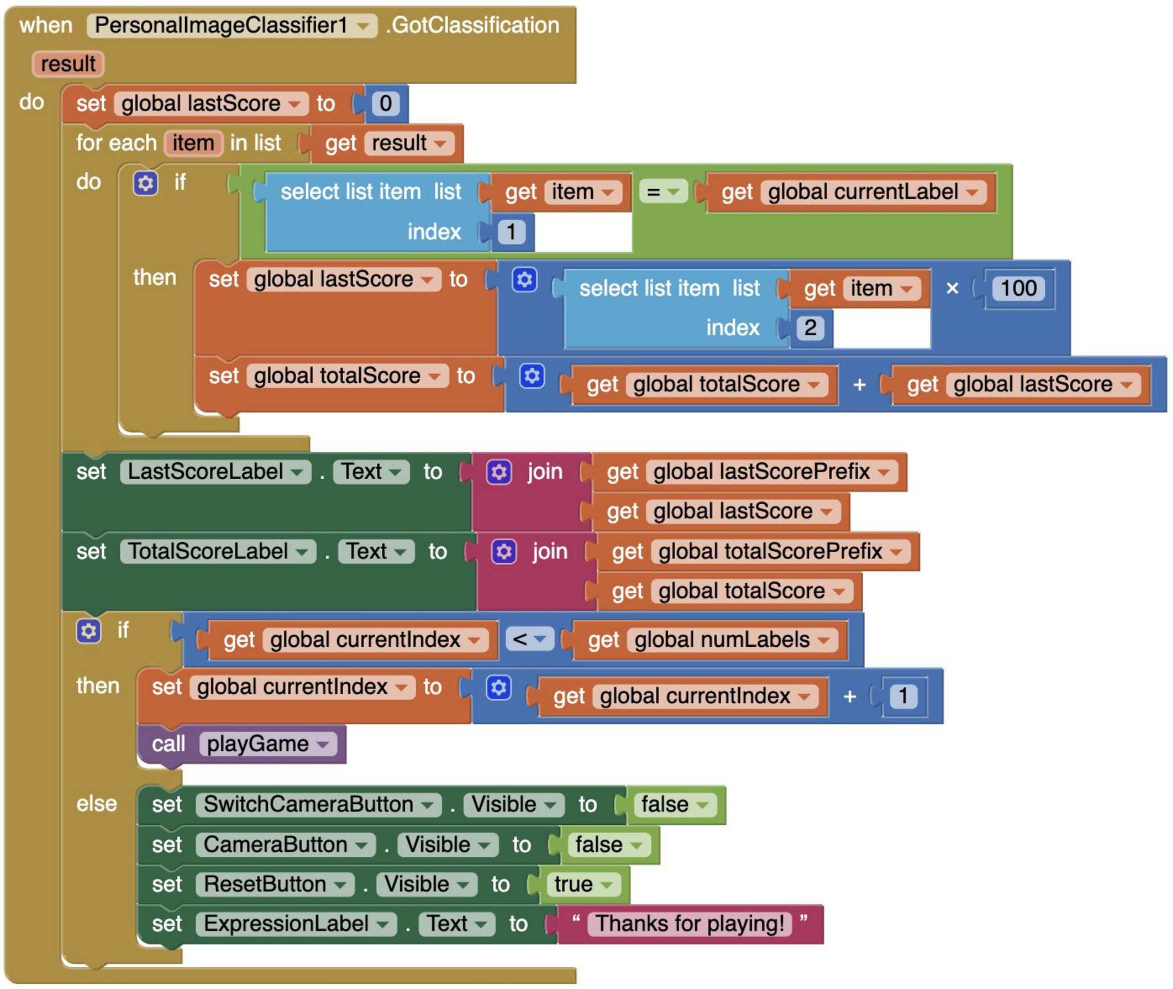
Designer:



Blocks:

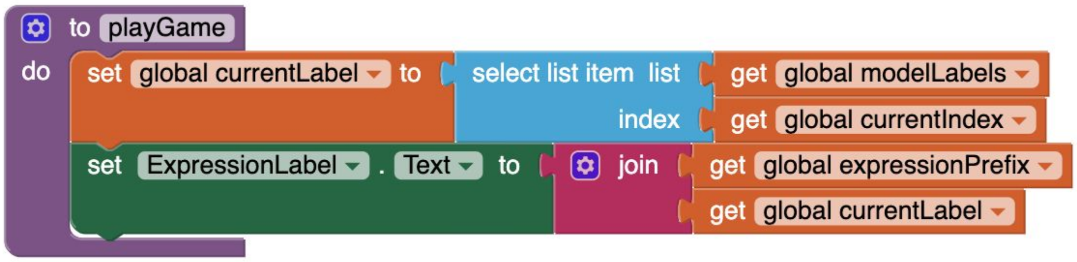


```
initialize global statusPrefix to " Status: "

initialize global lastScorePrefix to " Score for this expression: "

initialize global totalScorePrefix to " Total Score: "

initialize global expressionPrefix to " Make this expression: "

initialize global totalScore to 0

initialize global modelLabels to create empty list

initialize global currentIndex to 1

initialize global numLabels to 1

initialize global lastScore to 0

initialize global currentLabel to " "

when PersonalImageClassifier1 .Error
  errorCode
do  set StatusLabel . Text to  join  get global statusPrefix
                                     get errorCode

when SwitchCameraButton .Click
do  call PersonalImageClassifier1 .ToggleCameraFacingMode

when CameraButton .Click
do  call PersonalImageClassifier1 .ClassifyVideoData

when PersonalImageClassifier1 .ClassifierReady
do  set StatusLabel . Text to  join  get global statusPrefix
                                     " Ready! "
    set LastScoreLabel . Text to  join  get global lastScorePrefix
                                        0
    set TotalScoreLabel . Text to  join  get global totalScorePrefix
                                         0
    call PersonalImageClassifier1 .GetModelLabels
```

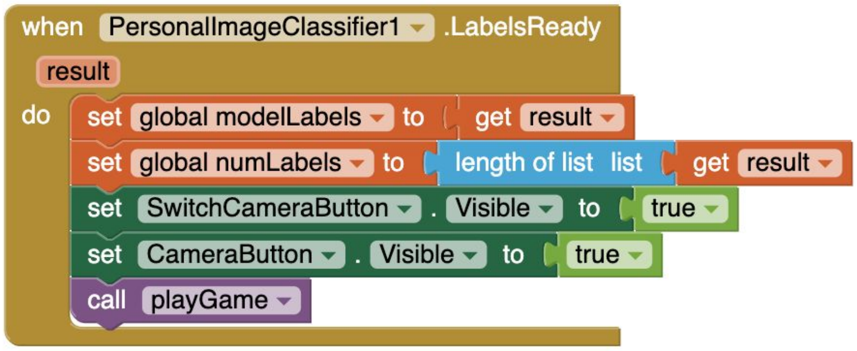


```
when PersonalImageClassifier1 .GotClassification
  result
do  set global lastScore to 0
    for each item in list  get result
    do  if  select list item  list  get item     = get global currentLabel
                              index 1
        then  set global lastScore to  select list item  list  get item   × 100
                                                         index 2
              set global totalScore to  get global totalScore + get global lastScore
    set LastScoreLabel . Text to  join  get global lastScorePrefix
                                        get global lastScore
    set TotalScoreLabel . Text to  join  get global totalScorePrefix
                                         get global totalScore
    if  get global currentIndex < get global numLabels
    then  set global currentIndex to  get global currentIndex + 1
          call playGame
    else  set SwitchCameraButton . Visible to false
          set CameraButton . Visible to false
          set ResetButton . Visible to true
          set ExpressionLabel . Text to " Thanks for playing! "
```

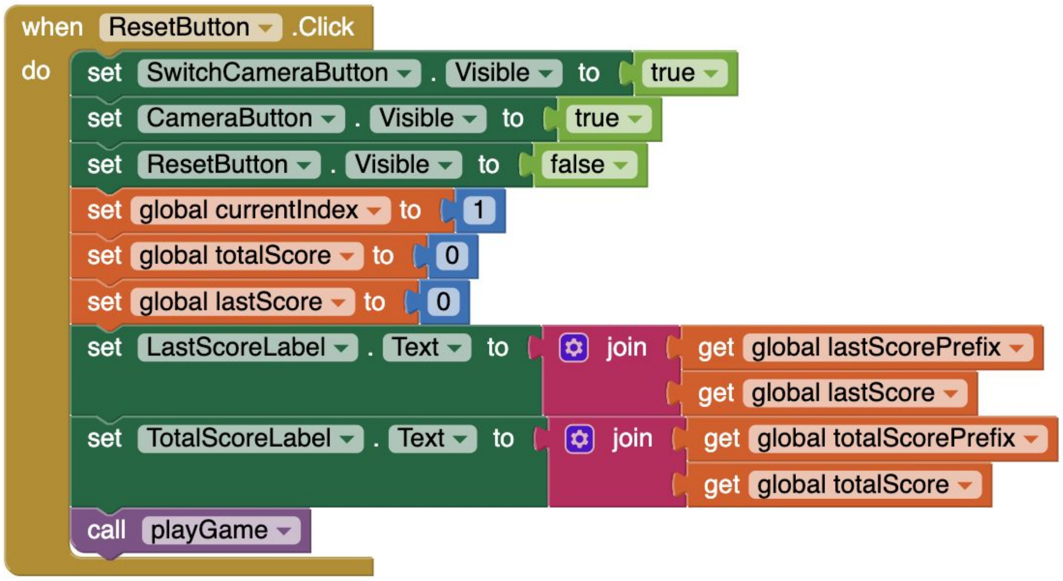
MIT
APP INVENTOR



```
to playGame
do  set global currentLabel to  select list item  list  get global modelLabels
                                                   index get global currentIndex
    set ExpressionLabel . Text to  join  get global expressionPrefix
                                         get global currentLabel
```



```
when PersonalImageClassifier1 .LabelsReady
  result
do  set global modelLabels to  get result
    set global numLabels to  length of list  list  get result
    set SwitchCameraButton . Visible to  true
    set CameraButton . Visible to  true
    call playGame
```



```
when ResetButton .Click
do  set SwitchCameraButton . Visible to  true
    set CameraButton . Visible to  true
    set ResetButton . Visible to  false
    set global currentIndex to  1
    set global totalScore to  0
    set global lastScore to  0
    set LastScoreLabel . Text to  join  get global lastScorePrefix
                                        get global lastScore
    set TotalScoreLabel . Text to  join  get global totalScorePrefix
                                         get global totalScore
    call playGame
```

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 training a model in image classification.

Lesson 1 Outline

Introduction to Machine Learning (10 minutes)

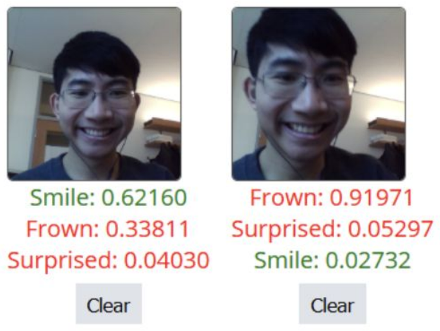
Using the presentation, give a general overview of what machine learning is. Additional explanations and resource links are in Appendix 4.

It is suggested that you have students work in pairs on this unit. They will be training a model of different facial expressions, and it could be helpful to have multiple people as input images for the model. It can also help to bring up biases that can happen when training a model on certain images (gender, race, etc).

Train a Model (25 minutes)

Have your students, working in pairs, load https://classifier.appinventor.mit.edu/oldpic/ in a browser on their computers using *PIC Guide Part 1*. Support students as they work through the training model:

1. **Add training data** - ask students to add 3-4 facial expressions (i.e. happy, sad, surprised) as “labels”, in preparation for Lessons 2 and 3 where they will use the trained model in two different apps. Ask them to “train” for each label with 10-20 pictures of themselves (and their partner). They can also upload images they find on the internet to introduce more varied images to their model.
2. **Select Model** - this allows you to tweak the parameters for the training. This is beyond the scope of this lesson. Students are instructed to keep the default values, and click on the Train Model button to continue. Note that the *Loss* number is a measure of how well the model performs with the given images. Students should note how high or low that number is.
3. **Add Testing Data** - Students should go through each label and test with several more images (another 10-20 per label). Encourage them to try different angles, or introduce things like their hands, or objects, to see how their model performs.
4. **View Results** - Students should focus on the images labelled incorrectly, and try to figure out why those images performed poorly. Is it a different angle, an ambiguous expression, a different person, background, coloring? Students can compare two different images on the right side of the screen by clicking on two images. This might help them to determine why


Smile: 0.62160
Frown: 0.33811
Surprised: 0.04030
Clear

Frown: 0.91971
Surprised: 0.05297
Smile: 0.02732
Clear

one image was labelled correctly, and another incorrectly. The confidence graph allows you to compare several images that perform well against those that did not.

5. **Improve the model -** Encourage students to continue to train their model by going back and adding more images to their model. If they want, they may also start over, now that they have a better idea of how the tool works.
6. **Download the model** - Once students are satisfied, they should download the model file (model.mdl) and save it on their computers for use in Lesson 2 and 3.

Wrap-up Discussion (10 minutes):

Once students have completed their model (or are in a place where they can continue to train before Lesson 2), lead the class in a discussion. Below are some possible questions:

1. Now that you've trained a machine learning model, how has your understanding of what machine learning is changed?
2. How did you improve upon your model between iterations?
3. Did you notice any bias in the model? What did it relate to?
4. Do you think your experience training the model will make you approach using image recognition tools in your life differently?

Appendix 2

Teacher's Guide

Lesson 2

Learning Objectives

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

1. Integrate a trained machine learning model into a mobile app.
2. Demonstrate understanding of performance issues of machine learning models.

Lesson 2 Outline

Introduction to the PIC App (10 minutes)

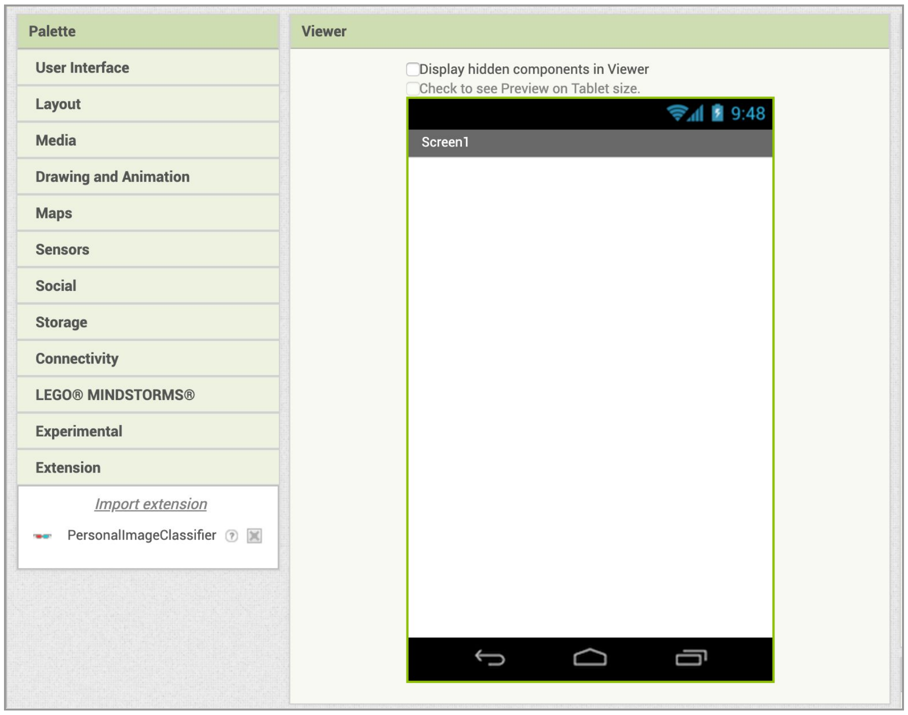
1. Explain that students will be using their model in a prebuilt App Inventor project and will be able to see how their model performs within the context of a mobile app.
2. Some students may need more time to train their model. If so, given the 5-10 minutes to do so.
3. All students should have a downloaded model file on their computer.

Run and Test PIC app (15 minutes):

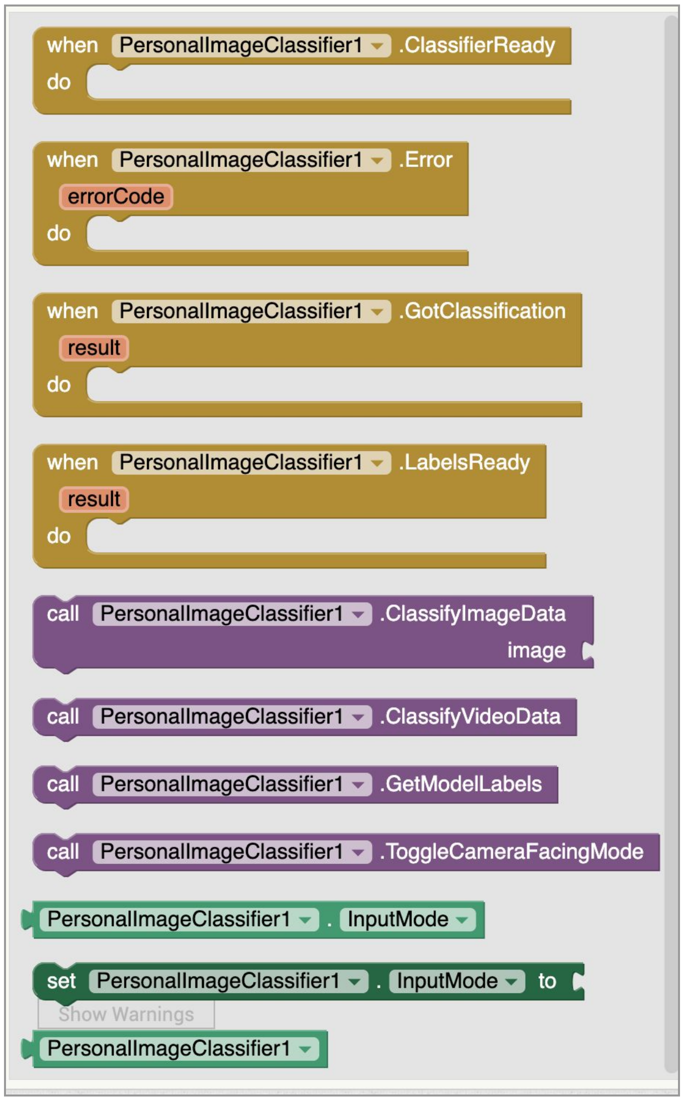
Personal Image Classifier Extension:

1. PersonalImageClassifier 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 *PersonalImageClassifier* extension will have already been imported to the PIC app template your students will complete.



Above: The PIC Extension imported into the PIC project.



```
when PersonalImageClassifier1 .ClassifierReady
do

when PersonalImageClassifier1 .Error
  errorCode
do

when PersonalImageClassifier1 .GotClassification
  result
do

when PersonalImageClassifier1 .LabelsReady
  result
do

call PersonalImageClassifier1 .ClassifyImageData
                                           image

call PersonalImageClassifier1 .ClassifyVideoData

call PersonalImageClassifier1 .GetModelLabels

call PersonalImageClassifier1 .ToggleCameraFacingMode

PersonalImageClassifier1 . InputMode

set PersonalImageClassifier1 . InputMode to
Show Warnings
PersonalImageClassifier1
```

Above: Blocks for the PersonalImageClassifier extension in the Blocks Editor.

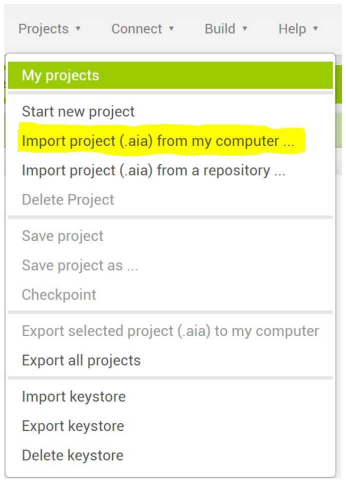
Make sure to review the student guide and/or the sidebar tutorial beforehand to help out students as best as possible.

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/2Nusctv) 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 PIC 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 PIC 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.



Upload the Model:

1. Following the instructions in the tutorial, students upload their model.mdl file from Lesson 1, and set it as the *Model* for the **PersonalImageClassifier** component in the project.

Test the App

1. Students can now experiment with the app. Students can try taking pictures of themselves with the app and seeing what label (facial expression) the app thinks they are making. The app will return the classification as a list of lists. For example:

    ((happy .79124) (surprised .58254) (sad .32198))

    The app thinks the expression is happy with a confidence level of .79124, or approximately 79%. Next it thinks it could be surprised, with a confidence level of .58254, and then it thinks it could be sad, with a confidence level of .32198.

    The app classifies either images or video, depending on which button the user presses. If the user chooses to classify an image, the app opens the camera app of the mobile device. The user can control the direction of the camera (front or back) using the camera app, and take a picture of their face. If the user chooses to classify video, they can use the Toggle Camera button to change the direction of the camera.

2. Students should note how well (or not) their model performs within the app. They can decide to go back and improve their model with more training.

Wrap-up Discussion (10 minutes):

1. Once students have completed the app and tested it, discuss how their app worked based on accuracy, limitations, and discuss ways the might improve their models.
    a. Did they see any differences between how it performed in the Classifier web tool and in the app?
2. Students may go back to the Classifier website to update their models based on their success in the apps. If they feel the app did not classify expressions well enough, they can

try again to improve their models, or add more labels if they want.

Appendix 3

Teacher's Guide

Lesson 3

Learning Objectives

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

1. Integrate a trained machine learning model into a mobile game app.
2. Demonstrate understanding of potential bias in machine learning and image classification.

Lesson 3 Outline

Introduction to Lesson (5 minutes)

Explain to students they will create a game app that uses their trained model to assess users' ability to match a given expression. You can demo the app using the ExpressionMatch_completed.aia project.

The game will present each of the facial expressions (labels) from the trained model of the Classifier website created in Lesson 1. For each label, the user must try to match the expression to score points. As each expression is matched, the user's total score is incremented by the score for that given expression. After all labels have been presented and scored, the user is asked if they wish to play again.

Coding Expression Match App (30 minutes)

1. Students follow the tutorial using the pdf student guide, or using the sidebar tutorial.
2. Students will create a game app called Expression Match, where the user tries to match a given expression (happy, sad, surprised) and gains points depending on how well they can match the model.
    a. Students will build the entire UI for the game.
    b. Students will upload their model for the PersonalImageClassifier extension (similar to Lesson 2).
    c. Students will code the game features, adding scoring based on the confidence level of a given facial expression.
3. Students test the app to assess how well the model works as a game.
    a. Suggest that students try out each other's game to see how well they perform vs. the author(s) of the game.
    b. Ask students if they see patterns in how you can score well (or not) in the game.
    c. Ask students to try different facial expressions to try to confuse the app.

Wrap-up Discussion (10 minutes)

1. Discuss how their game app worked based on accuracy and different users.
    a. How might bias in the training model affect the outcome of a game?
    b. Is it fair?
    c. Does it matter who is playing the game?
    d. What other factors contribute to a high or low score in the game?
2. Ask students how their attitudes toward machine learning has changed after this lesson.
3. Students may go back to the Classifier website to update their models based on how well the model performs.
4. Students may expand on the game app with some expansion suggestions at the end of the tutorial.

Appendix 4
Machine Learning Resources

What is Machine Learning?

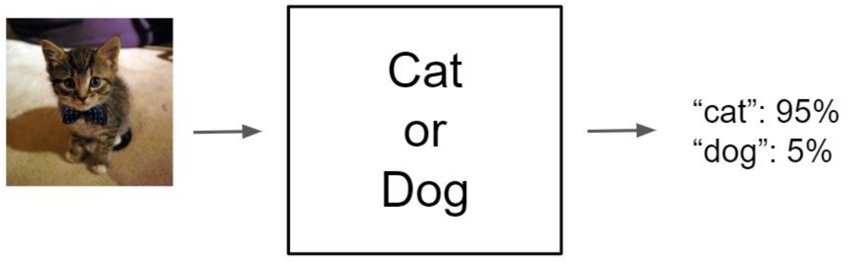
Machine Learning (ML) is a field of computation used to describe machines or systems that simulate human methods of cognition, such as learning or problem-solving. Examples of machine learning include Google's search engine, social media content suggestions, and the current development of self-driving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

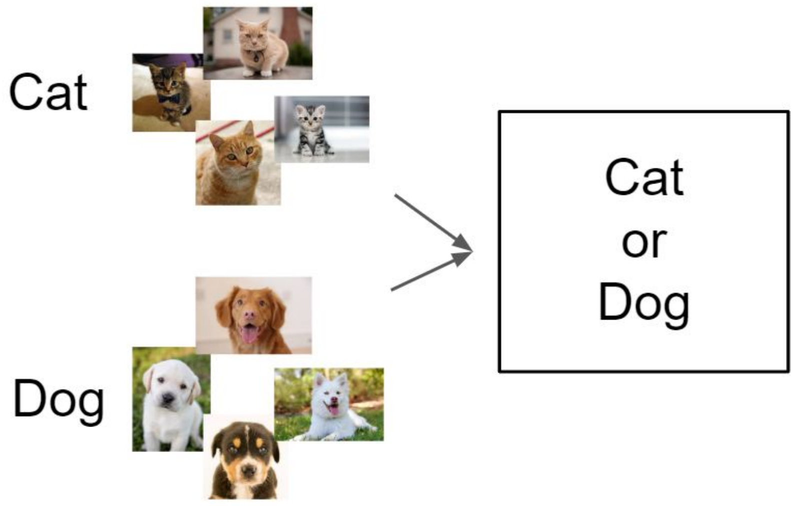
PersonalImageClassifier is an example of a **machine learning** tool that implements image classification. One popular way to create machine learning programs is by ***training*** them to classify images into some number of classes.

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



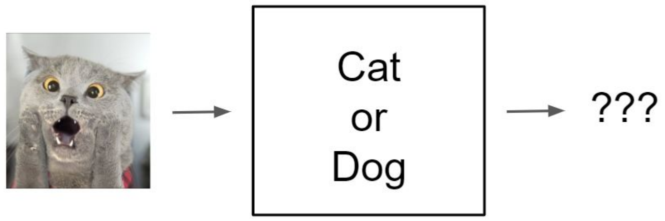
A lot of computation takes place in order to get from an image to the output text, and we will only describe the general principle here.

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.

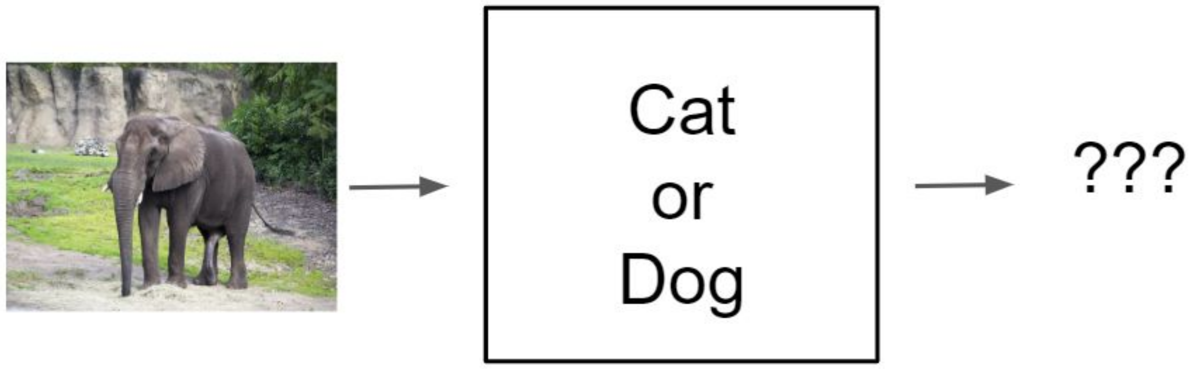


With enough examples, the program will be trained to "know" what types of images show dogs and what types of images show cats. It is important to note that this generally requires a huge number of examples from each class. Otherwise, the program will not have enough information to reliably classify new 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** or **labels** of image it has been taught. A **class** (or **label** as its referred to in the Classifier web tool) 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 PersonalImageClassifier 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 basically a series of computations, which could be random or 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 PersonalImageClassifier 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.