Artificial Intelligence 2: AI and Your App

This is a **bonus** lesson that will expand your knowledge about AI and how it works. It will **not** be a scored part of your submission but it could be helpful to learn how you could integrate AI with your app. If you need a refresher on how AI works, check out **Artificial Intelligence 1: All About AI.**

In this lesson, you will…

- Consider the ethics of AI
- Decide if AI is right for your app
- Learn how to integrate some AI elements with App Inventor and Thunkable

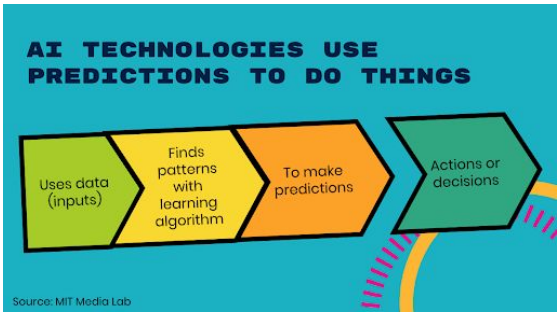
Key Terms and Concepts

- **Ethics** - a set of moral principles that affect how people decide what's right or wrong
- **Bias** - preconceived ideas somebody has that are often unfair to some people or groups

Inspiration

We'll be taking a deeper dive into AI application and your app today. If you need a refresher, review **Artificial Intelligence 1: All About AI.**

Most importantly, remember the 4 basic parts of AI:



By this point you likely have ideas about what you want your app to do. Will AI be important for your app? Let's consider a few more things before we answer that.

Ethics and Actions

Ethics is a part of philosophy that has to do with what is right or wrong. This is an important topic in the world of AI today for many reasons! You'll want to make sure your app is making decisions that help people and society instead of causing harm, even if the harm is by accident.

Gender Shades Example of bias

One place to start is by thinking about "healthy" data. Removing **bias** in your data helps make your model healthier, so that it can make better decisions. We've already learned about some details around bias in training data, here's a video that outlines more:

Code.org Data Bias

But data isn't the only area in AI in which we can think about ethics . We can also think about the ethics of the use or the results of the AI. Imagine an AI technology called the *Weed Puller* that predicts what plants are weeds and pulls them out. Using the *Weed Puller* as a base, consider the following:

- How can you remove **bias** from your datasets?
    - Data should **reflect the characteristics of the population** on which the app is being used. If your data has bias, the AI might make mistakes.
    - Imagine if the *Weed Puller* has a dataset of images of weeds found only in **the desert**. What if we want to use *Weed Puller* **near water**? It might not work, because it was not trained with data to help it recognize weeds found in near water. The *Weedpuller* has a bias for desert plants!
    -

A weed found in the desert	A weed found near the water
 *Prickly Lettuce*	 *Cattails*

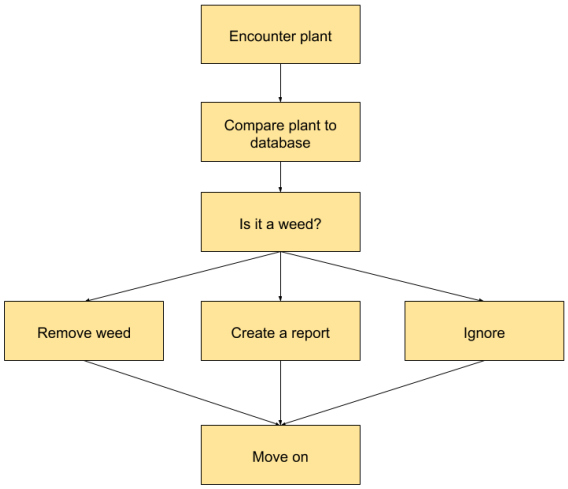
- See how different they look! The characteristics of weeds in the desert are drastically different than those near the water.

- What **decisions** does your AI make?
  - Some AI models make a single decision. Others make many more! Think about each decision you will program your app to make. Are there any decisions it could make that you **did not originally intend**?
  - Imagine if the Weed Puller tries to predict if it's seeing a weed. Is there **only one type** of weed? Can weeds be **hidden or camouflage their appearance**? It might make a **bad decision** like pulling a good plant.
  - Can you tell which one of these plants is a weed? What would you do if you weren't sure?

  -

- What are all the **actions** your AI could take?
  - Make a map of all the actions that your app could take (or not). What effects do you **intend** for these actions to have? Can you think of other consequences that you **haven't considered** yet?

- Weed Puller is trying to pull weeds. With enough training, this AI could do this as well as a person, but **without the contextual understanding from a human**. The AI only knows to do what it's programmed to do. It might make a **bad decision** like pulling out weeds that were part of an ecosystem protecting endangered animals.
- 

- What if your AI makes a **mistake**?
  - Can you **reduce bias** in the data and help **make better decisions**…
    - Finding a more representative dataset
    - Changing your app's actions based on how sure the AI is of its decision
    - Keeping your app's decisions private and sharing the decision only with the people who absolutely need to know it.
  - If the Weed Puller made a **mistake** and decided that a tomato plant was a weed, it would pull out the tomato plant. That could **negatively impact people** who were planning to eat it or insects in the garden who lived there.

- Does your app use or share **sensitive data**?
  - Data is important to the success of AI. While it's important to have a working AI model, the information shared needs to be carefully considered. If users are providing you information, they are trusting

you to respect their information and not misuse it. How would you feel if a stranger somehow knew your secrets?
    - Weed Puller is designed to use data to make decisions about weeds. Perhaps it collects data about the **locations of peoples' houses** to help with lawn work. If asked where it goes, it might share sensitive information about the location of peoples' homes; that would be a problem!

- How does your AI model interact with or make predictions **connected to people**?
    - Predictions about people or that affect people through AI should be carefully considered before being implemented. Think about yourself. You are a complex person, it wouldn't be easy for a stranger or AI to really know you without a lot of time!
    - http://www.envisioningcards.com/envision_pdfs/EC_Sample_Cards_Set.pdf

This may be a lesson focused on AI, but these ethical questions are **applicable to your entire app**.

As creators, **we bear responsibility on how our technology interacts with people** and we should always be mindful of the **potential impact** our creations can have.

Which solutions need AI?
Not every problem requires technology or AI. Think about what sort of problems are best suited for AI such as:

- Situations that are not programmable (ex - Google Maps)
- Situations that can predict something when given new information (ex- when given a new x-ray, an AI tech could identify cancer)
- Often connected to actions (ex - smart vacuum)

Carefully consider if your app should use AI. Does it add significant value to your core design functions?

AI Features in App Inventor and Thunkable

Here are the main features in App Inventor and Thunkable that utilize AI:

	Image Recognition	Speech Recognition	Language Translation
App Inventor*	LookExtension PersonalImageClassifier	PersonalAudioClassifier	
Thunkable	Microsoft Image and Emotion Recognizer	Speech Recognition	Language Translation

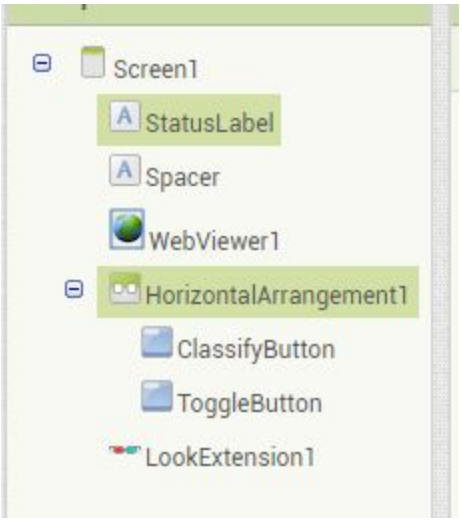
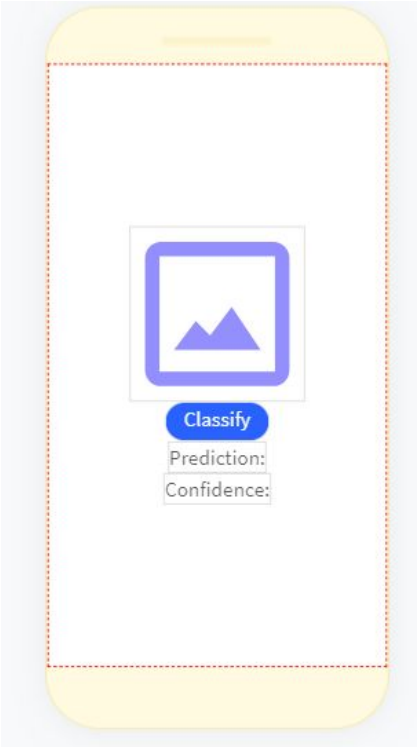
*Artificial intelligence blocks are not natively on the App Inventor platform and need to be downloaded as extensions. Official extensions are available here. Learn how to integrate extensions in your App Inventor platform here.

As you can see, the functions available in App Inventor and Thunkable are a bit basic; you won't be training your own AI model. **However, in Coding 13: Cloud Storage and APIs, you'll learn about what APIs are and how to connect other resources to your own app. You can find other programs out there that will give you more control of your AI model and integrate them to your app that way instead**. More resources can be found in the **Additional Resources** section of this lesson.

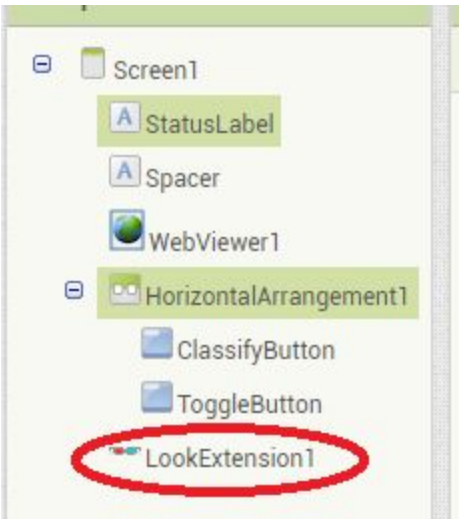
Activity: Image Recognition

Let's code a simple image recognition app:

App Inventor	Thunkable
1. First, let's get all our screen components in place.	1. First, let's get all our screen components in place.

 	   We have an Image, a Button, two Labels, a Camera, and an Image Recognizer.
2. You'll notice that there's a	2. Now we can get into the coding.

LookExtension1 that's not normally available in App Inventor. Refer to the above sections to learn how to import extensions into your app. **LINK**



There is one button available to the user: **Classify**.

When the **Classify** button is clicked, the **Camera** should first take a picture.

Then **Image Recognizer** should analyze the image from the camera we got from the camera.

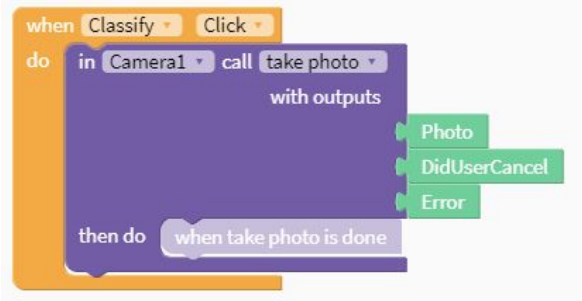
Finally, the app should show the user the photo, inform the user what **Image Recognizer** predicts the photo is, and how confident the app is in its prediction. We want to inform the user by setting the **Image** displayed and changing the text **Labels**.

--

*When the **Classify** button is clicked, the **Camera** should first take a picture.*

Hint: Where should we look when we want something to happen when a button is clicked?

Solution:

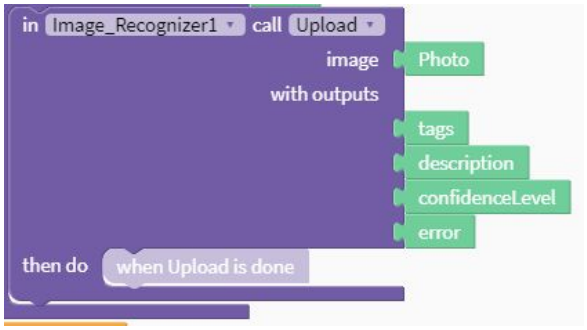


--

*Then **Image Recognizer** should analyze the image from the camera we got from the camera.*

Hint: Look in the blocks under **Image Recognizer**. Which one would be useful here? What input does it need?

Solution:

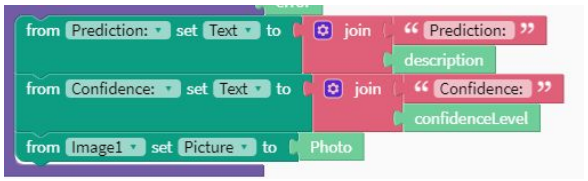


--

*Finally, the app should show the user the photo, inform the user what **Image Recognizer** predicts the photo is, and how confident the app is in its prediction. We want to inform the user by setting the **Image** displayed and changing the text **Labels**.*

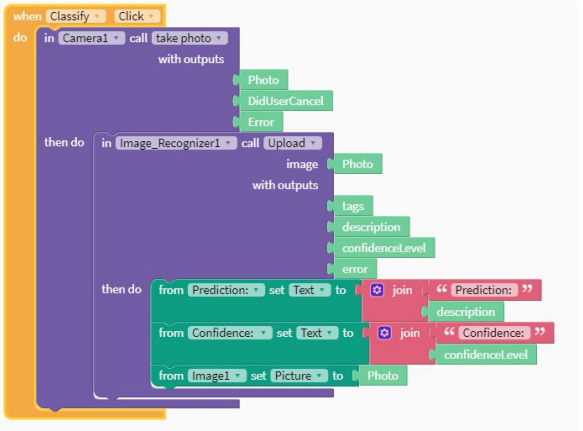
Hint: We're changing two **labels** and one **image** here. Use blocks from the appropriate section.

Solution:

<table>
<tr>
<td></td>
<td> 

This solution makes it a little more user friendly when reading labels by joining “Prediction: “ and “Confidence: “ to their respective fields. Note the space after the colon!

--

All together, your code should now look like this:
 
</td>
</tr>
<tr>
<td>3. Now we can get into the coding.

There are two buttons available to the user: Classify and Toggle.

When the Classify button is clicked, it should call LookExtension to analyze what the camera sees.

However, you should know that the LookExtension is that it needs a bit of</td>
<td>4. You’re ready to test!
Connect your mobile device to App Inventor and try it out!</td>
</tr>
</table>

time to load and *ready up*.
So, we should prevent the user from using the **Classify** button until **LookExtension** is ready.

When the **Toggle** button is clicked, the camera that the app uses should switch to another available camera on the device.

Then finally, once **LookExtension** has analyzed the image from the camera, the app should tell through the **StatusLabel** what its prediction is.

--

Let's code in this order:
*Prevent the user from using the **Classify** button until **LookExtension** is ready*

Hint: Look through the blocks available in **LookExtension** and see if there are any that would be useful to you.

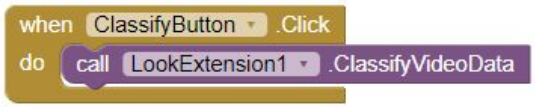
Solution:

--

*When the **Classify** button is clicked, it should call **LookExtension** to analyze what the camera sees.*

Hint: Nothing too fancy here. “When a button is clicked, do something”.

Solution:



You may have noticed we used “ClassifyVideoData” instead of “ClassifyImageData”. What do you think the difference is? You can experiment with both options later! For now, go ahead and stick with “ClassifyVideoData”.

--

*When the **Toggle** button is clicked, the camera that the app uses should switch to another available camera on the device.*

Hint: Another button! Remember to look in the available blocks under **LookExtension**.

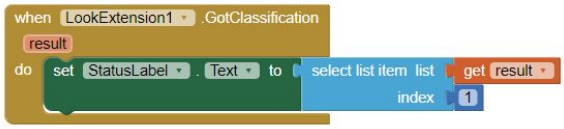
Solution:

--

*Once **LookExtension** has analyze the image from the camera,, the app should tell through the **StatusLabel** what its prediction is.*

Hint: **LookExtension** stores its predictions as a *list* of possible answers. This *list* is named “result” and its most confident prediction is *the first item in this list*.

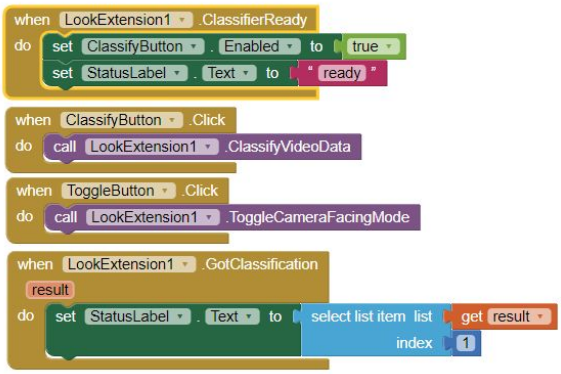
Solution:



This one is a bit hard to get right without some prior knowledge about **LookExtension**.

--

All together, your code should now look like this:

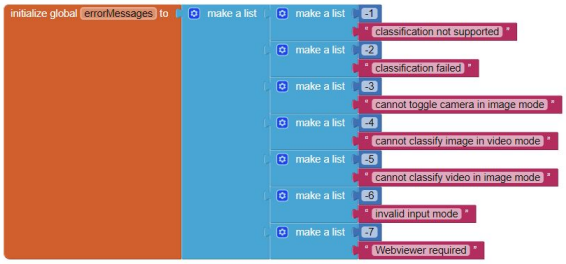


5. We also need to handle how this app deals with errors.

Many apps have a way to handle errors. For example, think of when you enter the incorrect password to a login. Most of the time, it’ll say “Invalid Password, try again”.

The text there is purely for humans to understand what the app thinks is wrong. Even when things don't go right, it's important to understand what's happening!

Copy the following code into your project. Read through it and make an effort to understand what's going on.



6. You're ready to test!

Connect your mobile device to App Inventor and try it out!

Reflection

Once you've taken a few pictures with your new image recognition app, take a moment and reflect:

- What improvements to this app could you make?
- Could image recognition be used to help solve a problem in your community? What about speech recognition or other types of AI?

- Using whatever you answered with in the previous question, consider the ethics behind your solution. Who would be affected by your solution? What good could it do? What harm could it do? How can you make it cause the most good and least harm to direct and indirect users?

Additional Resources

SolveIt Series by Technovation:

- Using Envisioning Cards to Create Ethical Technology with Batya Friedman
- How to Build Tech People Need and Want with Mike Burshteyn of ideas42!
- Batya Friedman Explains Moral and Technical Imagination

For more about ethics, check out the MIT AI Ethics Curriculum

Advanced AI Integrations

Remember to go over Coding 13: Cloud Storage and APIs to learn more about integrating outside services to your app. Some of these are only compatible with more advanced coding languages (such as Java or Swift) but they're definitely worth a look regardless if you intend to use it in your app.

- Dialogflow
  - Great for creating conversational AI apps
- TensorFlow
  - Lots of tools like transcribing handwritten numbers, pose guessing, and more!
- Google
  - Google has a large library of AI tools to use. Check out this overview video for a nice summary. *Note: If you decide to use these tools, be sure to double check the pricing. Some tools are free to use depending on how many users use your app.*

VianAI
https://www.youtube.com/watch?v=kzJI2vgOyhI

DialogFlow
https://www.youtube.com/watch?v=Fk6xtjceF5A