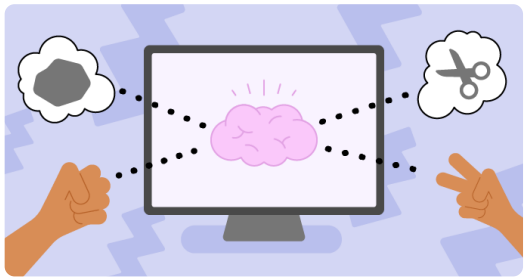
Rock, paper, scissors by hand

Use Google's Teachable Machine to build a rock, paper, scissors game controlled by hand gestures



Step 1 Introduction

What you will make

Create a game of rock, paper, scissors that you can play against the computer: you will make the gestures with your hands and have the computer recognise them. In this version of the game, the computer is going to cheat — it will recognise the player's move and choose the object that will beat it!

Essentially, you'll create a simpler version of the brain of this awesome robot.

What you should already know

This project assumes you already know some Python. Specifically, it assumes you know how to use:

- Variables
- Functions, including how to create your own function that accepts arguments
- The structure of a machine learning model — see Teach a computer to read (https://projects.raspberrypi.org/en/projects/teach-a-computer-to-read) for details
- How to use the command line interface on your computer to navigate to a directory — see Amazing image identifier (https://projects.raspberrypi.org/en/projects/amazing-image-identifier) for details

What you will need

- A computer with a camera
- An internet connection
- A Google account

What you will learn

- How to create and train a machine vision model quickly using Teachable Machine
- What you need to consider when you create a model to make sure that it works in practice
- How to include a model you've created in a desktop application

Additional information for educators

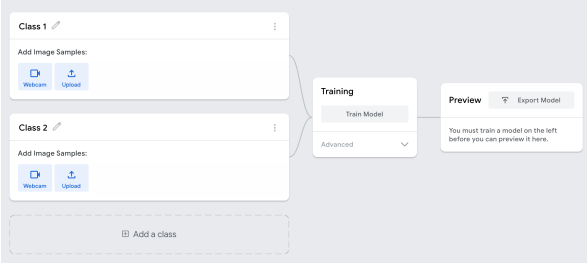
If you need to print this project, please use the printer-friendly version (https://projects.raspberrypi.org/en/projects/rock-paper-scissors-by-hand/print).

Here is a link to the resources for this project (https://rpf.io/p/en/rock-paper-scissors-by-hand-go).

Step 2 Get started with Teachable Machine

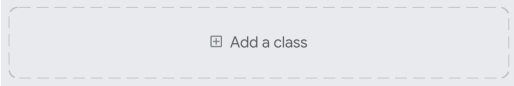
This project has two parts to it: first you build a model on Teachable Machine, and then you use that model in a game you'll build on your computer. Teachable Machine is a tool made by Google that creates machine learning models in your browser. You'll create a model to recognise images, but it can also create models that recognise sounds, or body poses.

Open the Teachable Machine image model training page (https://teachablemachine.withgoogle.com/train/image) in a new tab in your browser.



This is an untrained model, with two classes. Of course, since you need to recognise 'rock', 'paper', and 'scissors', you're going to need three classes.

Click the Add a class button to add a third class to your model.



Next, give your classes proper labels that match what they're going to be trained to recognise.

Use the pencil beside each class name to rename them as follows:

- Change 'Class 1' to 'rock'
- Change 'Class 2' to 'paper'
- Change 'Class 3' to 'scissors'

It's important to match this order, because some of the code provided to help your game run assumes that the classes are ordered like this.

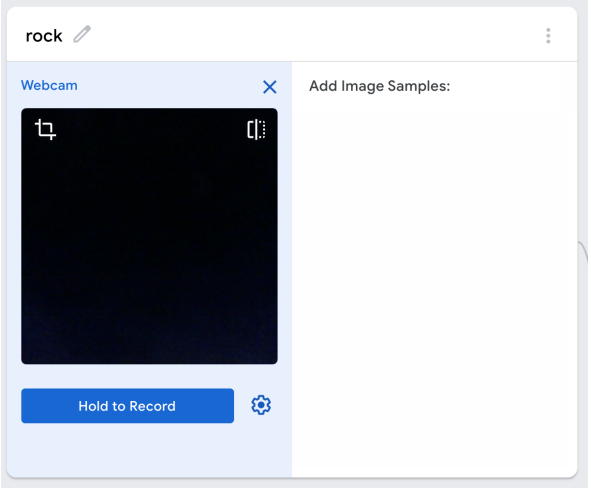
Now you're ready to record some training data.

Step 3 Collect your training data

Now that you have your classes set up, you need to add some data to use to train them. Each class has two options under 'Add Image Samples': Webcam and Upload. If you had a large collection of photographs of people making the 'rock', 'paper', and 'scissors' gestures on your computer, you could upload them. However, as you probably don't have that, you're going to use the Webcam option to record some images now. If you are using a Raspberry Pi with a camera, once you have set up the camera (https://projects.raspberrypi.org/en/projects/getting-started-with-picamera) it will function as a webcam.

Click on the Webcam button for the 'rock' class.

You will see the class expand to show a preview from your computer's camera (you may need to grant the Teachable Machine website permission to access your camera) and a Hold to Record button.



While you hold down the Hold to Record button with one hand, make the 'rock' gesture with the other, and make sure that it appears in the preview. Move your hand around and change the shape of the gesture, to ensure there is some variety in the training data.

If you stop recording, don't worry, just hold down the Hold to Record button to start again. Any new images will be added to the existing ones.

You should see your images appear in the 'Add Image Samples' section on the right-hand side. Keep recording until you have at least 300 of them. More images will usually improve the quality of the model you create, so recording several hundred is a good idea, if you have time.

If you want to be very thorough in making sure your training data is varied, you could:

- Switch hands.
- Have other people record themselves making the gesture — but make sure to have them record all three gestures, or the model may learn to associate them with only one!
- Change your facial expression.
- Move so you have a different background.
- Change the level of lighting in the room, or moving between bright and dark areas.

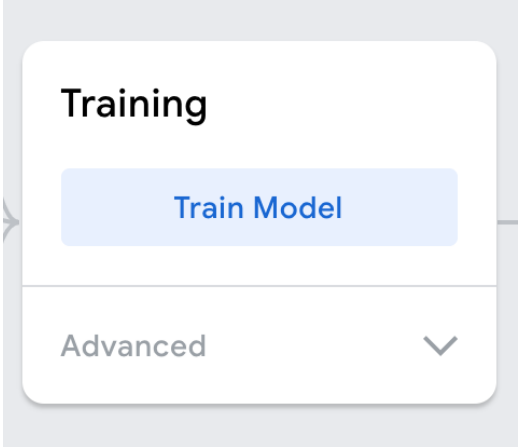
These are all suggestions to avoid the model learning the wrong rules. For example, if there's someone in the background of your 'rock' images who has left by the time you are recording 'paper', it may learn to associate that person with the 'rock' class of images. Or, since you're only recording for a few seconds, it may learn to use the expression on your face. So a good idea might be not to record all of the images for any class all at once, but rather to do some 'rock', then some 'paper', and 'scissors', then back to 'rock' and so on. Can you think of anything else you should consider?

Like you did for 'rock', record training images for 'paper' and 'scissors' using the appropriate gestures.

Step 4 Create your model

Now that your data is recorded, it's time to train your model!

Train the model by clicking the Train Model button. That was easy!



The training will take a short time, during which you can watch Teachable Machine count the epochs it's running through. You need to leave your browser open on the Teachable Machine tab, or the model will stop training. Once the training is complete, you'll see a preview of the model where you can test it by making gestures at your camera and seeing how likely your model thinks that gesture is to be each of 'rock', 'paper', and 'scissors'. The highest percentage indicates the 'guess' that your game would make if it saw that gesture.

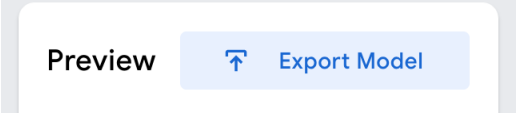
Spend some time testing your model's preview, to see if it usually predicts the gesture you make correctly.

If your model doesn't work as you'd like, there are a few things you can try:

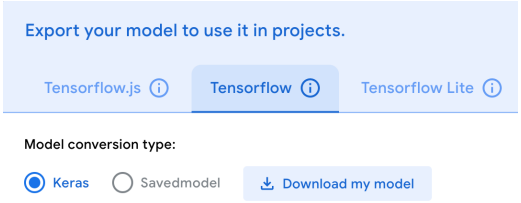
- Add more data by recording some new pictures for each class, in addition to the ones that you already have
- Open the 'Advanced' section of the 'Training' section and increase the number of epochs, so the model will train for longer

Once your model works well enough — it will never be right all the time, but being right most of the time is pretty good — it's time to export the model so you can use it as part of your game.

Click the Export Model button in the ‘Preview’ section.



In the screen that appears, select the ‘Tensorflow’ tab and, make sure that ‘Model conversion type’ is set to ‘Keras’, and click the Download my model button.



Make sure you remember where you save the model, you’ll need it in a few minutes!

Now that you have your model, you’re ready to build your game with it.

Step 5 Set up your game

Your completed model now needs to be inserted into a desktop Python application to make a working game. A simple version of rock, paper, scissors, where the computer will use the player's guess to cheat, has been written for you to use. You are focusing on the machine learning aspects of the project. However, there is a lot of room to improve the game's interface. If you want to learn about how to do this, check out the Getting started with GUIs project (https://projects.raspberrypi.org/en/projects/getting-started-with-guis). You can also make changes to the computer's method of choosing whether to play 'rock', 'paper', or 'scissors', if you want to create a game where the player always wins, or where the computer chooses at random.

First, you need to download and set up the starter file for the project.

For Windows

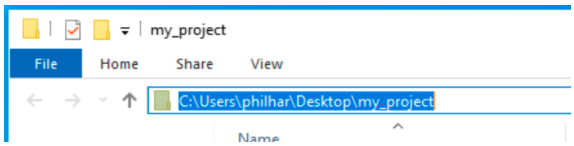
Download the starter zip file (https://rpf.io/p/en/rock-paper-scissors-by-hand-go) and unzip it somewhere you'll remember on your computer. If you can't think of a location, just put it on your desktop. This isn't the best place to keep things in the long term, but it's fine when you're working on them.

Next, you need to install the libraries you will use in this project. For this, you'll need to use the command line interface (CLI) — a program that controls your computer by typing text commands into a window. The command line interface is called 'command prompt' in Windows.

In the CLI, you don't access files by clicking to open them, or the directories (folders) they live in. You need to know the path to the file. It's like a set of directions, either from where you are currently located on the computer — called a relative path — or from the root of the computer's hard drive — called an absolute path. You'll need to find the path to the directory you've just unzipped for this next step.

Finding the path to a directory on Windows

The easiest way to find the path to a directory you know in Windows is to open the folder in Windows Explorer, as you would normally, and click into the navigation bar at the top of the window. The full path for the folder should become visible and you can then copy it.



In the CLI, navigate to the directory you just unzipped by entering the following command, and replace `[directory_path]` with the path to your directory.

```
cd [directory_path]
```

Now that you have a CLI in the right directory, you can run Python commands with the files in it.

The command to install the libraries you need uses pip, a tool to fetch Python code written by other people from the internet and set it up so you can use it in your projects. It's important to use pip to install libraries rather than just downloading them: some libraries need other libraries to work (these libraries are called their dependencies) and pip will automatically install those too.

Conveniently, pip can be given a list of all the libraries a project needs, and told to install them all at once. These are usually included in a file called `requirements.txt`, as they have been with the starter code provided here.

Run this command on your CLI to install the libraries you'll need. It may take a while to run, as it will have to download the libraries from the internet and some of them are quite large.

```
pip install requirements.txt
```

For macOS

To install the libraries and other files you're going to be using in this project, you'll need to use the command line interface (CLI) — a program that controls your computer by typing text commands into a window. The command line interface is called 'terminal' in macOS.

In the CLI, you don't access files by clicking to open them, or the directories (folders) they live in. You need to know the path to the file. It's like a set of directions, either from where you are currently located on the computer — called a relative path — or from the root of the computer's hard drive — called an absolute path. You need to find the path to the directory you've just unzipped for this next step.

There are also some special paths, that are sorts of shortcuts in the system, and you're going to be using one of them, called the home directory. Every user on a computer gets their own home directory to store their files, and it is accessed using a special character called the tilde (`~`).

Open the CLI on your computer and type the command below in:

```
cd ~
```

Now press the Return key.

You are now located in your home directory, and can install the files needed for this project there. Because this can be a complex process, a program to handle the installation for you has been created. Here are the details of this program (http://rpf.io/proj-rps), but be aware that it's written in a language called bash script, and won't look much like Python.

To download and run the program, type (or copy and paste) the command below into your CLI and press the Return key.

```
curl -L http://rpf.io/proj-rps | sudo bash -s $USER
```

The script may take several minutes, or more, to complete the setup depending on the speed of your computer and your internet connection. Once it has finished, you will have a new directory inside your home directory, called `amazing_image_identifier`. This is the directory you'll work in.

To change directory to the `amazing_image_identifier` directory, type the following command into your CLI and press the Return key.

```
cd amazing_image_identifier
```

For Linux (including Raspberry Pi)

To install the libraries and other files you'll use in this project, you need to use the command line interface (CLI) — a program that controls your computer by typing text commands into a window. The command line interface is called 'terminal' in Linux.

In the CLI, you don't access files by clicking to open them, or the directories (folders) they live in. You need to know the path to the file. It's like a set of directions, either from where you are currently located on the computer — called a relative path — or from the root of the computer's hard drive — called an absolute path. You need to find the path to the directory you've just unzipped for this next step.

There are also some special paths, that are sorts of shortcuts in the system, and you're going to be using one of them, called the home directory. Every user on a computer gets their own home directory to store their files, and it is accessed using a special character called the tilde (`~`).

Open the CLI on your computer and type the command below in:

```
cd ~
```

Now press the Return key.

You are now located in your home directory, and can install the files needed for this project there. Because this can be a complex process, a program to handle the installation for you has been created. Here are the details of this program (http://rpf.io/proj-rps), but be aware that it's written in a language called bash script, and won't look much like Python.

To download and run the program, type (or copy and paste) the command below into your CLI and press the Return key.

```
curl -L http://rpf.io/proj-rps | sudo bash -s $USER
```

The script may take several minutes, or more, to complete the setup depending on the speed of your computer and your internet connection. Once it has finished, you will have a new directory inside your home directory, called `rps_by_hand`. This is the directory you'll work in.

To change directory to the `rps_by_hand` directory, type the following command into your CLI and press the Return key.

```
cd rps_by_hand
```

There's a lot of code already provided for you, but it's all to create the game and capture an image from your computer's camera. First, run the program as it is to get an idea of how the game will work.

Go back to your CLI window and type the following command to run the program. Remember it, as you'll need to run the program several times.

If you're using Windows or Linux

```
python project.py
```

Then press the Return key to run the program.

If you're using macOS

```
python3 project.py
```

Then press the Return key to run the program.

It may take a moment, but when your program runs you should see something like the image below.

Now that you have the game running, it's time to connect your model to it.

Connect your model to the game

First, find the model you downloaded from Teachable Machine. It should be in a file called `converted_keras.zip`.

Unzip `converted_keras.zip` and open the resulting folder. Copy the `keras_model.h5` file to the project directory that contains the code for the game.

Note that the `labels.txt` file in the unzipped `converted_keras` directory contains the list of human-readable labels and the node numbers they match to on the model's output layer. For a larger number of categories, you would have Python read this file and automatically load them but, as there are only three categories in this model, the labels have already been included in the game code for you, in the list called `THROWS`.

Now that your game code and your model are in the same directory, you can load it into the game.

Find the function `get_player_throw`. On a line above it, add the following:

```
model = tf.keras.models.load_model('keras_model.h5', compile=False)
```

This will load your model into the game. Now you need to add a function that uses the model to get players' throws — their choice of 'rock', 'paper', or 'scissors'.

Update the `get_player_throw` function with the following code which will load the image the game has captured, then pass it to the model for a prediction. That prediction is then returned, so it can be used by the game code, instead of the default 'rock' that the game came with!

```
def get_player_throw():
    image = tf.keras.preprocessing.image.load_img(IMG_NAME, target_size=(IMAGE_SIZE, IMAGE_SIZE))
    image = tf.keras.preprocessing.image.img_to_array(image)
    image = np.expand_dims(image, axis=0)

    prediction_result = model.predict(image)

    best_prediction = THROWS[np.argmax(prediction_result[0])]

    return best_prediction
```

To understand each step in this process, you can review the Testing your computer's vision project [https://projects.raspberrypi.org/en/projects/testing-your-computers-vision/), particularly the 'Load your model and image' and 'Use the model to predict an image' steps. In short:

- All the lines starting with `image` work to convert the image to the right format for the model
- The `prediction_result` line gets the model's prediction in the form of numbers representing confidence in different guesses
- The `best_prediction` line takes the prediction with the highest value (`np.argmax` gets the index of the highest value in a list) and uses it to look up the label of that prediction in the THROWS list

Save your project

Run the program and play with it a few times! If you're not happy with the quality of its guesses, think about any differences between the training data and the inputs you give your game — maybe the light has changed, or you're wearing different clothes, etc. If you need to, you can record more training data in Teachable Machine, train your model again, and then download it and replace the model in the game with an updated version.
