How to Deploy a Machine Learning UI on Heroku in 5 Steps

Gradio lets you build a UI for your machine learning model. Heroku lets you host it. Here's how to use them together.

Abubakar Abid
Jul 19 · 4 min read ★

In this tutorial, I'm going to show you, step by step, how to create and deploy your machine learning model and UI on Heroku. I'll use this drag-and-drop image interface that I created for the Inception Net model as an example:

You can try out the GUI at: https://arrane-tundra-87ra8.herokuapp.com/ (might take a minute for the dyno to start up)

The Gradio Piece

I'm assuming that you've already somewhat familiar with the Gradio library (https://github.com/gradio-app/gradio). If not, this MNIST tutorial is a good place to get started. The quick overview is: Gradio is a Python library that lets you create UIs around your machine learning model by specifying 3 things: a Python function, input component, and an output component.

Step 1: Create your Gradio app

We'll start with a script that creates a drag-and-drop image interface around the Inception model. Create a brand new directory, and copy-paste the code below into a python file that we'll call `app.py`:

```
import gradio as gr
import tensorflow as tf
import numpy as np
import requests

inception_net = tf.keras.applications.InceptionV3() # load the model

# Download human-readable labels for ImageNet.
response = requests.get("https://git.io/JJkYN")
labels = response.text.split("\n")

def classify_image(inp):
  inp = inp.reshape((-1, 299, 299, 3))
  inp = tf.keras.applications.inception_v3.preprocess_input(inp)
  prediction = inception_net.predict(inp).flatten()
  return {labels[i]: float(prediction[i]) for i in range(1000)}

image = gr.inputs.Image(shape=(299, 299, 3))
label = gr.outputs.Label(num_top_classes=3)

gr.Interface(fn=classify_image, inputs=image, outputs=label).launch()
```


What's happening in this script is that we're loading the Inception Net image classifier, using Tensorflow Keras. Since this is an image classification model, we will use the `Image` input interface. We'll output a dictionary of labels and their corresponding confidence scores with the `Label` output interface. *(To deploy your own model, swap this out with your own code).*

Step 2: Write your `requirements.txt` file

We need to make sure that we designate all of the Python libraries that we will need as dependencies. So create a `requirements.txt` file with the dependencies in the same directory as `app.py`. In this case, we'll have:

```
setuptools==41.0.0
gradio
tensorflow==2.0.0
numpy
requests
```


Getting the package versions to work on Heroku can be a little tricky, but the packages and versions above work for me. **Important:** Heroku limits the size of your build to be 500 MB, which can fill up easily given the size of the Tensorflow package. **I suggest using the Tensorflow 2.0.0 library, which is smaller than some of other versions of the library.**

If you want to test the script locally (which you should!), run from a Terminal: `pip install -r requirements.txt` to install the dependencies, and then `python app.py` to launch the UI. At this point, you should see that your interface is running on `localhost:7860` (or another port), which if you navigate to in your browser should show your GUI!

This URL is locally accessible. How do we share this with the world? Gradio comes with a built-in `share` parameter that can create public URLs, but those expire after a certain amount of time. To create permanent public links, we'll be using Heroku!

The Heroku Piece

To deploy your web app on Heroku, you'll need to have a Heroku account, and it's very convenient to have the Heroku CLI as well. So go ahead and create a Heroku account and download the CLI, if you haven't done that already.

Now, on to the deployment!

Step 3: Create a `setup.sh` file

In order to deploy our app correctly on Heroku, we need to make sure it's served on the right URL & port. The commands below do that, so put them inside a file called `setup.sh` in the same directory as your app.

```
export GRADIO_SERVER_NAME=0.0.0.0
export GRADIO_SERVER_PORT="$PORT"
```


Step 4: Create a `Procfile`

Now, we're going to create a file called `Procfile` (that's it, there's no extension for this file!) whose job it is to tell Heroku what commands to run to start the Gradio app. There are only two commands: to run the bash script we created in the previous step, and then to launch our app. So our Heroku `Procfile` looks like this:

```
web: source setup.sh && python app.py
```

If you're wondering why we're using `source` instead of `sh`, it's because we need the environment variables to be saved after the script is finished executing.

Step 5: Deploy!

At this point, your directory should look like this:

Just 4 files in 1 directory — super simple!

If you haven't already added these files to a git repo, do it by running the following in your terminal:

```
git init
git add -A
git commit -am "commit message here"
```

We just need to push this to a Heroku dyno. First, let's allocate a Heroku dyno for our app by running:

```
heroku create
```

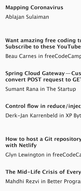
Heroku will automatically spin up and assign a random name to your app. Now, we push our changes to Heroku:

```
git push heroku master
```

Give this a few minutes to install all of the dependencies on your dyno. It's possible to run into dependency issues here, which you can resolve by reading the error messages and installing the right version of each library *(see my note about Heroku's size limits)*. If you don't see any errors, then you should be able to open up your app by running:

```
heroku psiscale web=1
heroku open
```

That last command will open up your default browser and bring you to the app. You'll notice the web address is the automatically generated instance name plus the Heroku domain.



That's it! You have a live link that you can share with anyone. Time to start running some predictions :)

Thanks to Ali Abid.

Machine Learning · Artificial Intelligence · Data Science · Heroku · User Interface