Introduction to Gradio

Estimated Reading Time: 15 minutes

Objectives

After completing this reading, you will be able to:

- Explain the basics of Gradio
- Demonstrate an example of implementing image classification in PyTorch

Introduction

Gradio is the way to demonstrate your machine learning model with a user-friendly web interface so that everyone can use it anywhere. Gradio is an open-source Python package that allows you to quickly build a demo or web application for your machine learning model, API, or any arbitrary Python function. You can then share a link to your demo or web application using Gradio's built-in sharing features. No JavaScript, CSS, or web hosting experience is needed.

Why Use Gradio?

Gradio is useful for several reasons:

- Ease of use: Gradio enables the creation of interfaces for models with just a few lines of code.
- Flexibility: Gradio supports various inputs and outputs, such as text, images, files, and more.
- Sharing and collaboration: Interfaces can be shared with others through unique URLs, facilitating easy collaboration and feedback collection.

Task 1: Getting Started with Gradio

To begin using Gradio, you first need to install the library. You can install Gradio using pip:

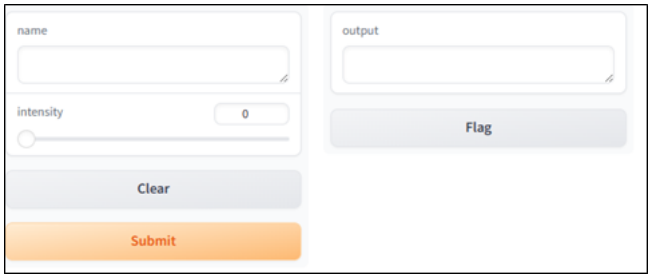
```
pip install gradio
```

Creating Your First Gradio Interface

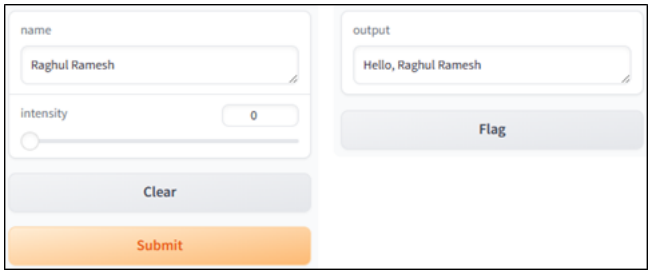
You can run Gradio in your favorite code editor, Jupyter notebook, Google Colab, or anywhere else you write Python. Let's write your first Gradio app:

```
import gradio as gr
def greet(name, intensity):
  return "Hello, " + name + "!" * int(intensity)
demo = gr.Interface(
  fn=greet,
  inputs=["text", "slider"],
  outputs=["text"],
)
demo.launch(server_name="127.0.0.1", server_port= 7860)
```

If running from a file, the demo below will open in a browser on http://127.0.0.1:7860. If you are running within a notebook, the demo will appear embedded within the notebook.



Type your name in the textbox on the left, drag the slider, and then press the Submit button. You should see a friendly greeting on the right.



Understanding the Interface class

Note that to make your first demo, you created an instance of the `gr.Interface` class. The `Interface` class is designed to create demos for machine learning models that accept one or more inputs and return one or more outputs.

The `Interface` class has three core arguments:

- **fn:** The function to wrap a user interface (UI) around
- **inputs:** The Gradio component(s) to use for the input. The number of components should match the number of arguments in your function.
- **outputs:** The Gradio component(s) to use for the output. The number of components should match the number of return values from your function.
  The **fn** argument is flexible — you can pass any Python function you want to wrap with a UI. In the example above, you saw a relatively simple function, but the function could be anything from a music generator to a tax calculator to the prediction function of a pretrained machine learning model.

The **input** and **output** arguments take one or more Gradio components. As we'll see, Gradio includes more than 30 built-in components (such as the `gr.Textbox()`, `gr.Image()`, and `gr.HTML()` components) that are designed for machine learning applications.

If your function accepts more than one argument, as is the case above, pass a list of input components to **inputs**, with each input component corresponding to one of the function's arguments in order. The same applies if your function returns more than one value: simply pass a list of components to **outputs**. This flexibility makes the `Interface` class a very powerful way to create demos.
Let's create a simple interface for an image captioning model. The BLIP (Bootstrapped Language Image Pretraining) model can generate captions for images. Here's how you can create a Gradio interface for the BLIP model.

```
pip install transformers
pip install torch
```

```
import gradio as gr
from transformers import BlipProcessor, BlipForConditionalGeneration
from PIL import Image
processor = BlipProcessor.from_pretrained("Salesforce/blip-image-captioning-base")
model = BlipForConditionalGeneration.from_pretrained("Salesforce/blip-image-captioning-base")
def generate_caption(image):
    # Now directly using the PIL Image object
    inputs = processor(images=image, return_tensors="pt")
    outputs = model.generate(**inputs)
    caption = processor.decode(outputs[0], skip_special_tokens=True)
    return caption
def caption_image(image):
    """
    Takes a PIL Image input and returns a caption.
    """
    try:
        caption = generate_caption(image)
        return caption
    except Exception as e:
        return f"An error occurred: {str(e)}"
iface = gr.Interface(
    fn=caption_image,
    inputs=gr.Image(type="pil"),
    outputs="text",
    title="Image Captioning with BLIP",
    description="Upload an image to generate a caption."
)
iface.launch(server_name="127.0.0.1", server_port= 7860)
```

Here, we use the `BlipProcessor` and `BlipForConditionalGeneration` from the `transformers` library to set up an image captioning model. This example demonstrates creating a web interface using Gradio, where the input parameter specifies an image and the output is the generated text caption. The title and description parameters enhance the interface by providing context and instructions for users.

Use Case: Image Captioning

Image captioning models like BLIP are incredibly powerful tools in various domains, from helping visually impaired individuals understand image content to efficiently organizing and searching through large photo libraries. Creating a Gradio interface for such a model makes it accessible for non-technical users to interact with and benefit from this technology. For example, photographers or digital asset managers could use your application to automatically generate descriptive names for their images, enhancing the usability and searchability of their digital libraries.

Task 2: Image Classification in PyTorch

Now let`s explore a different kind of computer vision task — **Image Classification**. Image classification is a central task in computer vision. Building better classifiers to classify what object is present in a picture is an active area of research, as it has applications stretching from autonomous vehicles to medical imaging.

Such models are perfect to use with Gradio's image input component. In this tutorial, we will build a web demo to classify images using Gradio. We can build the whole web application in Python.

Step 1: Setting up the image classification model

First, we will need an image classification model. For this tutorial, we will use a pretrained Resnet-18 model, as it is easily downloadable from PyTorch Hub. You can use a different pretrained model or train your own.

```
import torch
model = torch.hub.load('pytorch/vision:v0.6.0', 'resnet18', pretrained=True).eval()
```

Step 2: Defining a predict function

Next, we will need to define a function that takes in the user input, which in this case is an image, and returns the prediction. The prediction should be returned as a dictionary whose keys are class name and values are confidence probabilities. We will load the class names from this text file.

In the case of our pretrained model, it will look like this:

```
import requests
from PIL import Image
from torchvision import transforms
# Download human-readable labels for ImageNet.
response = requests.get("https://git.io/JJkYN")
labels = response.text.split("\n")
def predict(inp):
 inp = transforms.ToTensor()(inp).unsqueeze(0)
 with torch.no_grad():
  prediction = torch.nn.functional.softmax(model(inp)[0], dim=0)
  confidences = {labels[i]: float(prediction[i]) for i in range(1000)}
 return confidences
```

Let's break this down. The function takes one parameter:

`inp: the input image as a PIL image`

The function converts the input image into a PIL Image and subsequently into a PyTorch tensor. After processing the tensor through the model, it returns the predictions in the form of a dictionary named confidences. The dictionary's keys are the class labels, and its values are the corresponding confidence probabilities.

In this section, we define a predict function that processes an input image to return prediction probabilities. The function first converts the image into a PyTorch tensor and then forwards it through the pretrained model. We use the softmax function in the final step to calculate the probabilities of each class. The softmax function is crucial because it converts the raw output logits from the model, which can be any real number, into probabilities that sum up to 1. This makes it easier to interpret the model's outputs as confidence levels for each class.

Step 3: Creating a Gradio interface

Now that we have our predictive function set up, we can create a Gradio Interface around it.

In this case, the input component is a drag-and-drop image component. To create this input, we use `Image(type="pil")` which creates the component and handles the preprocessing to convert that to a PIL image.

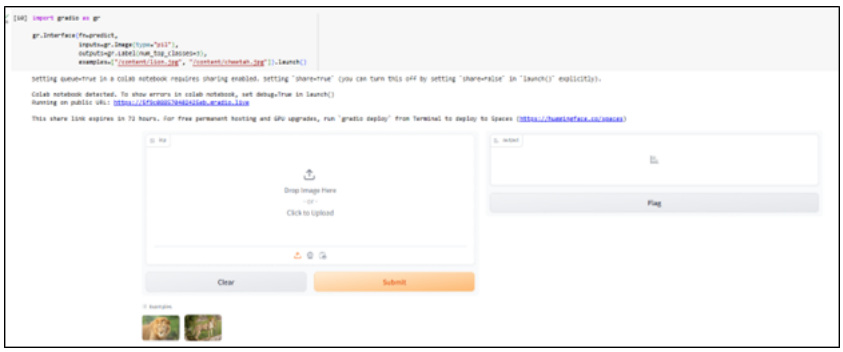
The output component will be a Label, which displays the top labels in a nice form. Since we don't want to show all 1,000 class labels, we will customize it to show only the top 3 classes by constructing it as `Label(num_top_classes=3)`.

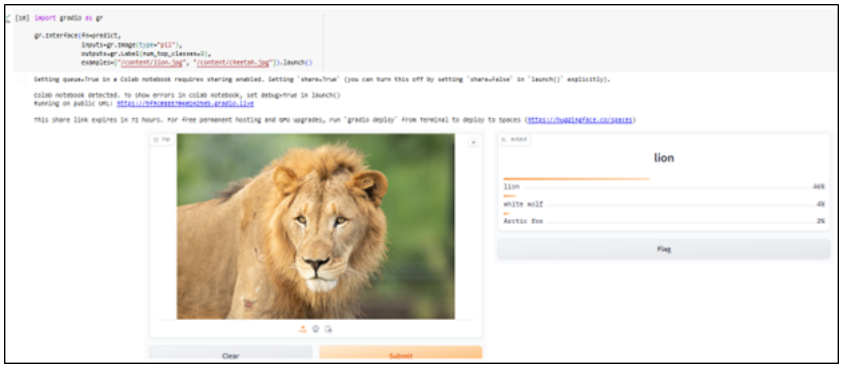
Finally, we'll add one more parameter, the examples parameter, which allows us to prepopulate our interfaces with a few predefined examples. The code for Gradio looks like this:

```
import gradio as gr
gr.Interface(fn=predict,
       inputs=gr.Image(type="pil"),
       outputs=gr.Label(num_top_classes=3),
       examples=["/content/lion.jpg", "/content/cheetah.jpg"]).launch()
```

The example paths provided, /content/lion.jpg and /content/cheetah.jpg, are placeholders. You should replace these with the actual paths to images on your system or server where you have saved the images you want to use for testing. This ensures that when you or others are using the Gradio interface, the examples are correctly loaded and can be used to demonstrate the functionality of your image classifier.

This produces the following interface, which you can try in a browser (try uploading your own examples).





And you're done! That's all the code you need to build a web demo for an image classifier. If you'd like to share with others, try setting `share=True` when you `launch()` the Interface!

Conclusion

Gradio simplifies the process of building interactive web demos for machine learning models. By integrating Gradio with models like BLIP for image captioning, you can create practical, user-friendly applications that leverage the power of AI to solve real-world problems. This tool not only aids in demonstrating the capabilities of your models but also in collecting valuable feedback for further improvement.