Introduction

Ramin Zarebidoky (LiterallyTheOne)

18 Oct 2025

Introduction

Keras

Keras is a high-level API for building and training **Deep Learning Models**. It is designed to be a stand-alone project. But with the help of **TensorFlow**, **PyTorch**, and **Jax**, It can run on top of different hardware (e.g., **CPU**, **GPU**).

The fascinating thing about **Keras** is that it is super easy to get started with. You can train and test a model with only a few lines of code. It is a perfect way to learn **Deep Learning** concepts by practically seeing their effects.

Google Colab

There are so many ways available to run a **Deep Learning** code. One of the fastest and easiest way that doesn't require any installation, is **Google Colab**. Google colab is a free could-based platform that is powered by `jupyter notebook`. All the packages that we want for this tutorial is already installed in **Google Colab**. Also, every code that we run in this tutorial can be run on this platform. So, I highly recommend you to start with **Google Colab**. After you have become more comfortable with the packages and concepts, switch to a local platform like your personal computer.

All the codes that we talk about in this tutorial is available in the **GitHub**. Each tutorial has a link to its respective code, which you can find it at the bottom of each page. To load and run the codes in **Google Colab**, you can follow these steps.

- Open Google colab
- From **files** select **Open Notebook**
- Go to the **GitHub** section
- Copy the **URL** of the code
- Select the **.ipynb** file that you want

Here is an example of loading this tutorial's code:

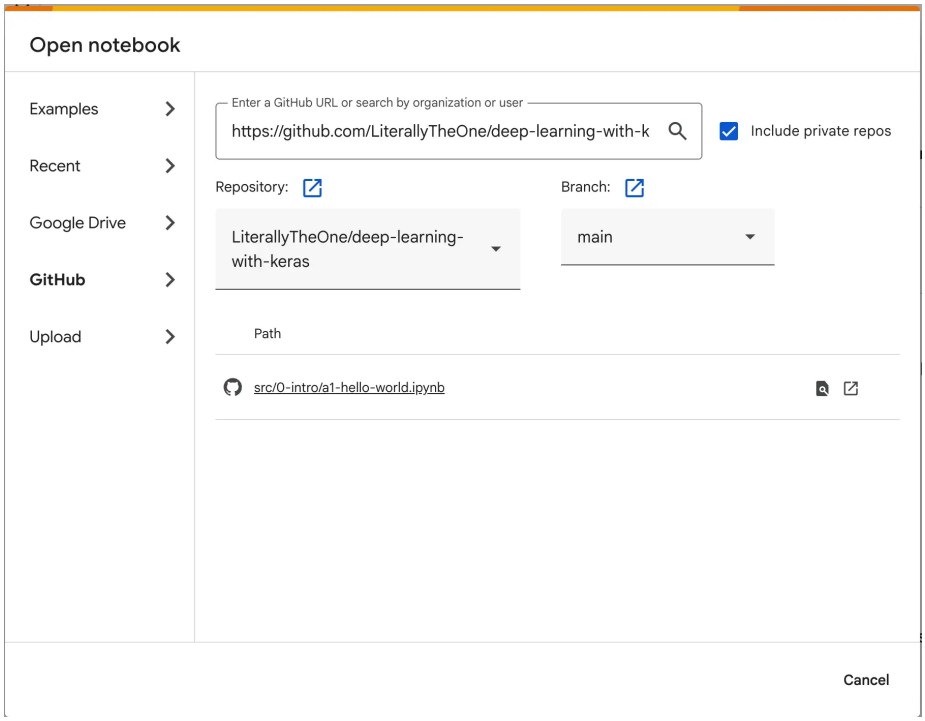

Figure 1: Colab GitHub

Hello World

Here is a **Hello World** example that we are gradually going to complete it step by step.

```
# Setup
import os

os.environ["KERAS_BACKEND"] = "torch"

# Imports
from keras.datasets import mnist
import keras
from keras import layers

# Prepare the Data
(train_images, train_labels), (test_images, test_labels) =
 ↪  mnist.load_data()

train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype("float32") / 255
test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype("float32") / 255

# Define the model
model = keras.Sequential([
    layers.Dense(512, activation="relu"),
    layers.Dense(10, activation="softmax")
])

model.compile(optimizer="adam",
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])

# Train the model
model.fit(train_images, train_labels, epochs=5, batch_size=128)

# Test the model
test_loss, test_acc = model.evaluate(test_images, test_labels)

"""
--------
output:

Epoch 1/5
469/469             2s 5ms/step - accuracy: 0.9259 - loss: 0.2622
Epoch 2/5
```

```
469/469             2s 5ms/step - accuracy: 0.9685 - loss: 0.1092
Epoch 3/5
469/469             2s 5ms/step - accuracy: 0.9797 - loss: 0.0710
Epoch 4/5
469/469             2s 5ms/step - accuracy: 0.9852 - loss: 0.0515
Epoch 5/5
469/469             2s 5ms/step - accuracy: 0.9901 - loss: 0.0363
313/313             1s 3ms/step - accuracy: 0.9801 - loss: 0.0616
"""
```

In the code above, we have trained and tested a model on a dataset called **MNIST**. In the future, we are going deeper into each step, but for now, here is a simple explanation of each of them. At first, we set up the backend of our **Keras**. We set that to `torch`, but you can set that to either `tensorflow` or `jax`. Then we imported the necessary modules. After that, we downloaded MNIST. MNIST contains of 28×28 images of handwritten digits between 0 and 9. Then, we normalize our data. After that, we defined a simple model and compiled the model with the proper `optimizer`, `loss`, and `metrics`. With the `fit` function, we train our model. And finally, we test our model with the evaluate function. As you can see in the output, our model's `accuracy` and `loss` are shown in each training step and testing step. We have gotten `99%` accuracy on our training data and `98%` accuracy on our test data.

Kaggle

Kaggle is one of the biggest platforms for data science and machine learning enthusiasts. It contains a huge number of datasets and a variety of competitions. In this tutorial, we are going to select an **Image Classification Dataset** from Kaggle. One of the simplest ways to do that is to go to the **Datasets** section in **Kaggle**, and select **Image Classification** tag in the **filters**. The dataset that we have to choose should have stored its images in a format like below:

```
class_a/
...a_image_1.jpg
...a_image_2.jpg
class_b/
...b_image_1.jpg
...b_image_2.jpg
```

As you can see, in the format above, we have some directories with images. Each directory represents a class, and the images in each directory belong to that class.

You can see the format of a **Dataset** by scrolling down to **Data Explorer**. For example, in Tom and Jerry Image classification We have a format as below:

As you can see, we have 4 directories (*jerry*, *tom*, *tom_jerry_0*, *tom_jerry_1*),

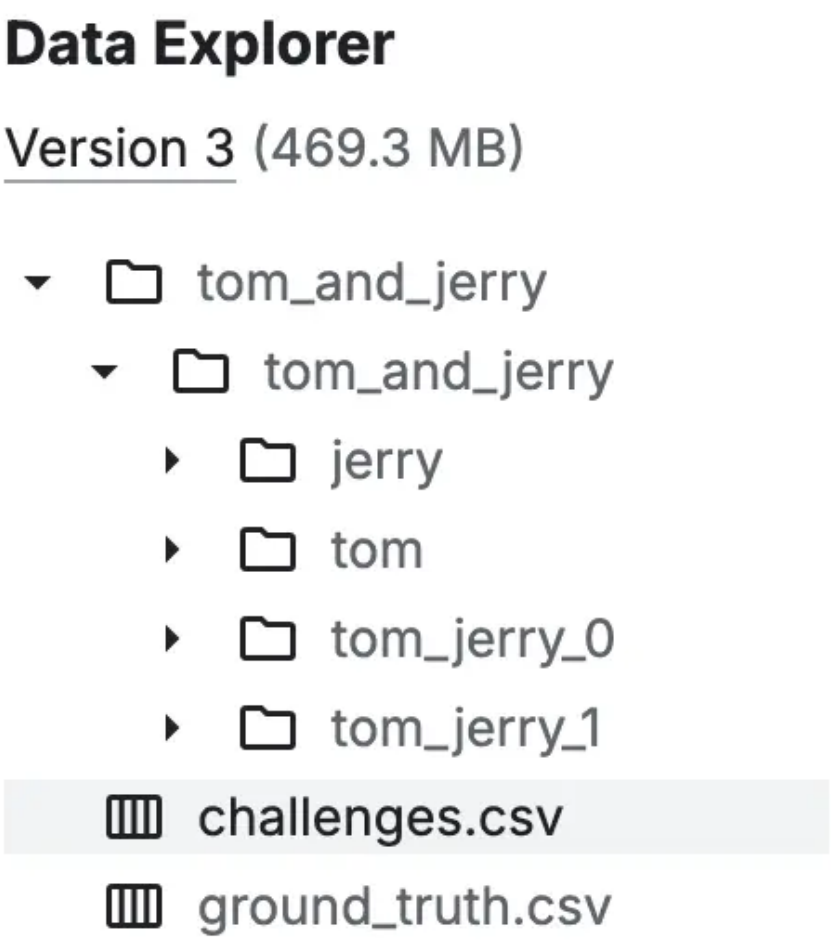

Figure 2: Tom and Jerry data format

and each directory has its own images. So, we have 4 classes. Another example is Facial Emotion Recognition Dataset. Its data structure is as below:

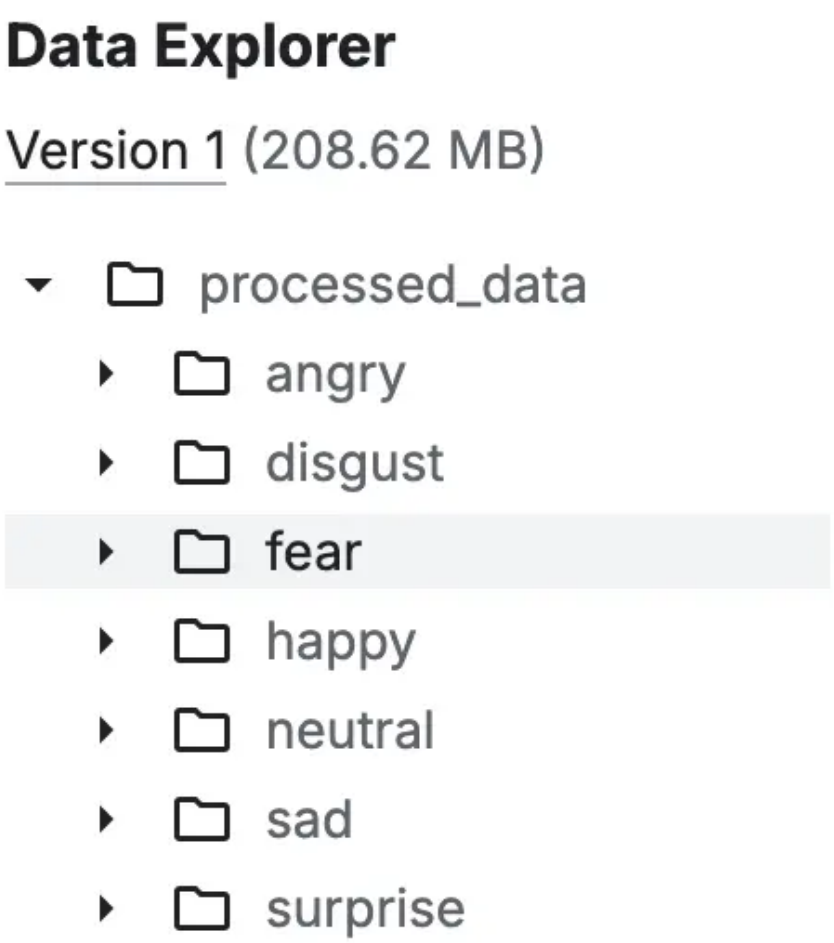

Figure 3: Facial Emotion data format

As you can see, in the image above, we have 7 directories (*angry*, *disgust*, *fear*, *happy*, *neutral*, *sad*, and *surprise*). So, we have 7 classes.

Now, you should select a dataset with these criteria:

- Image classification
- Each class has its own directory, and its images are in that directory
- It's better for our dataset size not to exceed 5GB.

Conclusion

In this tutorial, we have introduced **Keras**. Then, we explained about **Google Colab** and how to load a notebook from **GitHub**. After that, we provided a **Hello World** example that we are going to complete it overtime. Finally, we introduced **Kaggle** and explained how to get a suitable **Dataset** from it for this tutorial.