Welcome to Colab!

If you're already familiar with Colab, check out this video to learn about interactive tables, the executed code history view, and the command palette.

What is Colab?

Colab, or "Colaboratory", allows you to write and execute Python in your browser, with

- Zero configuration required
- Access to GPUs free of charge
- Easy sharing

Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab can make your work easier. Watch Introduction to Colab to learn more, or just get started below!

Getting started

The document you are reading is not a static web page, but an interactive environment called a **Colab notebook** that lets you write and execute code.

For example, here is a **code cell** with a short Python script that computes a value, stores it in a variable, and prints the result:

```
seconds_in_a_day = 24 * 60 * 60
seconds_in_a_day
```

```
86400
```

To execute the code in the above cell, select it with a click and then either press the play button to the left of the code, or use the keyboard shortcut "Command/Ctrl+Enter". To edit the code, just click the cell and start editing.

Variables that you define in one cell can later be used in other cells:

```
seconds_in_a_week = 7 * seconds_in_a_day
seconds_in_a_week
```

```
604800
```

Colab notebooks allow you to combine **executable code** and **rich text** in a single document, along with **images**, **HTML**, **LaTeX** and more. When you create your own Colab notebooks, they are stored in your Google Drive account. You can easily share your Colab notebooks with co-workers or friends, allowing them to comment on your notebooks or even edit them. To learn more, see Overview of Colab. To create a new Colab notebook you can use the File menu above, or use the following link: create a new Colab notebook.

Colab notebooks are Jupyter notebooks that are hosted by Colab. To learn more about the Jupyter project, see jupyter.org.

Data science

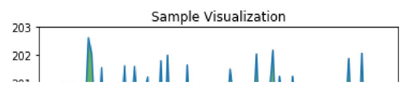
With Colab you can harness the full power of popular Python libraries to analyze and visualize data. The code cell below uses **numpy** to generate some random data, and uses **matplotlib** to visualize it. To edit the code, just click the cell and start editing.

```
import numpy as np
from matplotlib import pyplot as plt

ys = 200 + np.random.randn(100)
x = [x for x in range(len(ys))]

plt.plot(x, ys, '-')
plt.fill_between(x, ys, 195, where=(ys > 195), facecolor='g', alpha=0.6)

plt.title("Sample Visualization")
plt.show()
```



You can import your own data into Colab notebooks from your Google Drive account, including from spreadsheets, as well as from Github and many other sources. To learn more about importing data, and how Colab can be used for data science, see the links below under Working with Data.

Machine learning

With Colab you can import an image dataset, train an image classifier on it, and evaluate the model, all in just a few lines of code. Colab notebooks execute code on Google's cloud servers, meaning you can leverage the power of Google hardware, including GPUs and TPUs, regardless of the power of your machine. All you need is a browser.

Colab is used extensively in the machine learning community with applications including:

- Getting started with TensorFlow
- Developing and training neural networks
- Experimenting with TPUs
- Disseminating AI research
- Creating tutorials

To see sample Colab notebooks that demonstrate machine learning applications, see the machine learning examples below.

More Resources

Working with Notebooks in Colab

- Overview of Colaboratory
- Guide to Markdown
- Importing libraries and installing dependencies
- Saving and loading notebooks in GitHub
- Interactive forms
- Interactive widgets

Working with Data

- Loading data: Drive, Sheets, and Google Cloud Storage
- Charts: visualizing data
- Getting started with BigQuery

Machine Learning Crash Course

These are a few of the notebooks from Google's online Machine Learning course. See the full course website for more.

- Intro to Pandas DataFrame
- Linear regression with tf.keras using synthetic data

Using Accelerated Hardware

- TensorFlow with GPUs
- TensorFlow with TPUs

Featured examples

- NeMo Voice Swap: Use Nvidia's NeMo conversational AI Toolkit to swap a voice in an audio fragment with a computer generated one.
- Retraining an Image Classifier: Build a Keras model on top of a pre-trained image classifier to distinguish flowers.
- Text Classification: Classify IMDB movie reviews as either *positive* or *negative*.
- Style Transfer: Use deep learning to transfer style between images.
- Multilingual Universal Sentence Encoder Q&A: Use a machine learning model to answer questions from the SQuAD dataset.
- Video Interpolation: Predict what happened in a video between the first and the last frame.

```
!pip install -q tensorflow-datasets tensorflow matplotlib
```

```
import numpy as np
import tensorflow as tf

import tensorflow_datasets as tfds
```

```
from tensorflow.keras.utils import to_categorical
```

```
## Loading images and labels
(train_ds, train_labels), (test_ds, test_labels) = tfds.load("tf_flowers",
    split=["train[:70%]", "train[:30%]"], ## Train test split
    batch_size=-1,
    as_supervised=True,  # Include labels
)
```

```
     Downloading and preparing dataset 218.21 MiB (download: 218.21 MiB, generated: 221.83 MiB, total: 440.05 MiB) to /root/tensorflow_datasets/tf_flowers/3.0.1...
     Dl Completed...: 100%                                     5/5 [00:02<00:00, 2.01 file/s]
     Dataset tf_flowers downloaded and prepared to /root/tensorflow_datasets/tf_flowers/3.0.1. Subsequent calls will reuse this data.
```

```
## check existing image size
train_ds[0].shape
```

```
     TensorShape([442, 1024, 3])
```

```
## Resizing images
train_ds = tf.image.resize(train_ds, (150, 150))
test_ds = tf.image.resize(test_ds, (150, 150))
```

```
train_labels
```

```
     <tf.Tensor: shape=(2569,), dtype=int64, numpy=array([2, 3, 3, ..., 0, 2, 0])>
```

```
## Transforming labels to correct format
train_labels = to_categorical(train_labels, num_classes=5)
test_labels = to_categorical(test_labels, num_classes=5)
```

```
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input
```

```
train_ds[0].shape
```

```
     TensorShape([150, 150, 3])
```

```
## Loading VGG16 model
base_model = VGG16(weights="imagenet", include_top=False, input_shape=train_ds[0].shape)
```

```
     Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
     58889256/58889256 [==============================] - 0s 0us/step
```

```
## will not train base mode
# Freeze Parameters in model's lower convolutional layers
base_model.trainable = False
```

```
## will not train base mode
# Freeze Parameters in model's lower convolutional layers
base_model.trainable = False
```

```
## Preprocessing input
train_ds = preprocess_input(train_ds)
test_ds = preprocess_input(test_ds)
```

```
## model details
base_model.summary()
```

```
     Model: "vgg16"
     _________________________________________________________________
      Layer (type)                Output Shape              Param #
     =================================================================
      input_1 (InputLayer)        [(None, 150, 150, 3)]     0

      block1_conv1 (Conv2D)       (None, 150, 150, 64)      1792

      block1_conv2 (Conv2D)       (None, 150, 150, 64)      36928

      block1_pool (MaxPooling2D)  (None, 75, 75, 64)        0

      block2_conv1 (Conv2D)       (None, 75, 75, 128)       73856

      block2_conv2 (Conv2D)       (None, 75, 75, 128)       147584

      block2_pool (MaxPooling2D)  (None, 37, 37, 128)       0

      block3_conv1 (Conv2D)       (None, 37, 37, 256)       295168
```

```
 block3_conv2 (Conv2D)       (None, 37, 37, 256)       590080

 block3_conv3 (Conv2D)       (None, 37, 37, 256)       590080

 block3_pool (MaxPooling2D)  (None, 18, 18, 256)       0

 block4_conv1 (Conv2D)       (None, 18, 18, 512)       1180160

 block4_conv2 (Conv2D)       (None, 18, 18, 512)       2359808

 block4_conv3 (Conv2D)       (None, 18, 18, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 9, 9, 512)         0

 block5_conv1 (Conv2D)       (None, 9, 9, 512)         2359808

 block5_conv2 (Conv2D)       (None, 9, 9, 512)         2359808

 block5_conv3 (Conv2D)       (None, 9, 9, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 4, 4, 512)         0

=================================================================
Total params: 14714688 (56.13 MB)
Trainable params: 0 (0.00 Byte)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

```
#add our layers on top of this model
from tensorflow.keras import layers, models

flatten_layer = layers.Flatten()
dense_layer_1 = layers.Dense(50, activation='relu')
dense_layer_2 = layers.Dense(20, activation='relu')
prediction_layer = layers.Dense(5, activation='softmax')


model = models.Sequential([
    base_model,
    flatten_layer,
    dense_layer_1,
    dense_layer_2,
    prediction_layer
])
```

```
from tensorflow.keras.callbacks import EarlyStopping

model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy'],
)
```

```
es = EarlyStopping(monitor='val_accuracy', mode='max', patience=5,  restore_best_weights=True)
```

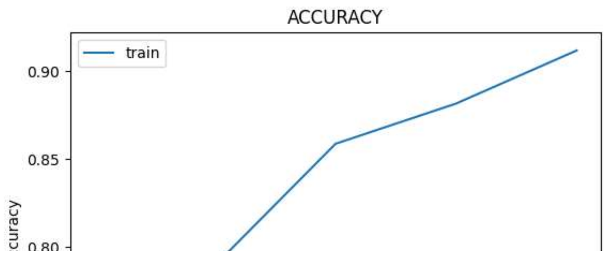
```
history=model.fit(train_ds, train_labels, epochs=5, validation_split=0.2, batch_size=32, callbacks=[es])
```

```
Epoch 1/5
65/65 [==============================] - 657s 10s/step - loss: 0.8456 - accuracy: 0.7071 - val_loss: 1.2420 - val_accuracy: 0.6304
Epoch 2/5
65/65 [==============================] - 647s 10s/step - loss: 0.5961 - accuracy: 0.7898 - val_loss: 1.0110 - val_accuracy: 0.6790
Epoch 3/5
65/65 [==============================] - 630s 10s/step - loss: 0.3979 - accuracy: 0.8589 - val_loss: 1.1255 - val_accuracy: 0.6984
Epoch 4/5
65/65 [==============================] - 630s 10s/step - loss: 0.3205 - accuracy: 0.8818 - val_loss: 1.1503 - val_accuracy: 0.7101
Epoch 5/5
65/65 [==============================] - 631s 10s/step - loss: 0.2371 - accuracy: 0.9119 - val_loss: 1.3278 - val_accuracy: 0.6946
```

```
los,accurac=model.evaluate(test_ds,test_labels)
print("Loss: ",los,"Accuracy: ", accurac)
```

```
35/35 [==============================] - 291s 8s/step - loss: 0.1887 - accuracy: 0.9292
Loss:  0.18868598341941833 Accuracy:  0.9291552901268005
```

```
import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'])
plt.title('ACCURACY')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train'],loc='upper left')
plt.show()
```



```
import numpy as np
import pandas as pd
y_pred = model.predict(test_ds)
y_classes = [np.argmax(element) for element in y_pred]
#to_categorical(y_classes, num_classes=5)
#to_categorical(test_labels, num_classes=5)
print(y_classes[:10])
print("\nTest")
print(test_labels[:10])
```

```
    35/35 [==============================] - 262s 7s/step
    [2, 3, 3, 2, 3, 0, 0, 0, 0, 4]

    Test
    [[0. 0. 1. 0. 0.]
     [0. 0. 0. 1. 0.]
     [0. 0. 0. 1. 0.]
     [0. 0. 0. 0. 1.]
     [0. 0. 0. 1. 0.]
     [1. 0. 0. 0. 0.]
     [1. 0. 0. 0. 0.]
     [1. 0. 0. 0. 0.]
     [1. 0. 0. 0. 0.]
     [0. 1. 0. 0. 0.]]
```