Introduction to Google Collab, Keras, Tensorflow, and Pytorch

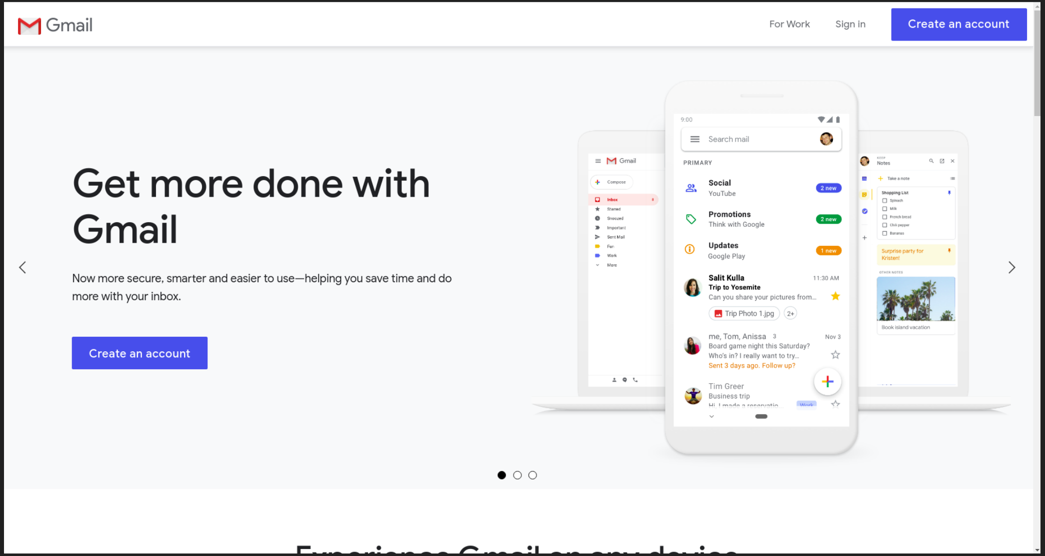
CMPSC 448

Outline

- Create Google account
- Access Google Colab
- Upload your document to Google Colab

Create Google Account

- Goto www.google.com/gmail
- Click on the Create an account


Gmail

For Work Sign in Create an account

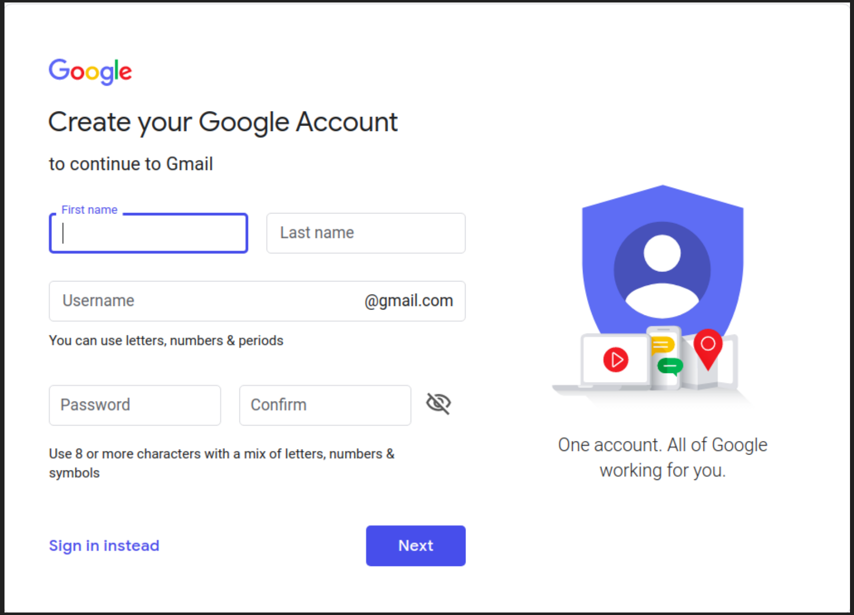
Get more done with Gmail

Now more secure, smarter and easier to use—helping you save time and do more with your inbox.

Create an account

Create Google Account

- Enter your information



Google

Create your Google Account

to continue to Gmail

First name

Last name

Username @gmail.com

You can use letters, numbers & periods

Password

Confirm

Use 8 or more characters with a mix of letters, numbers & symbols

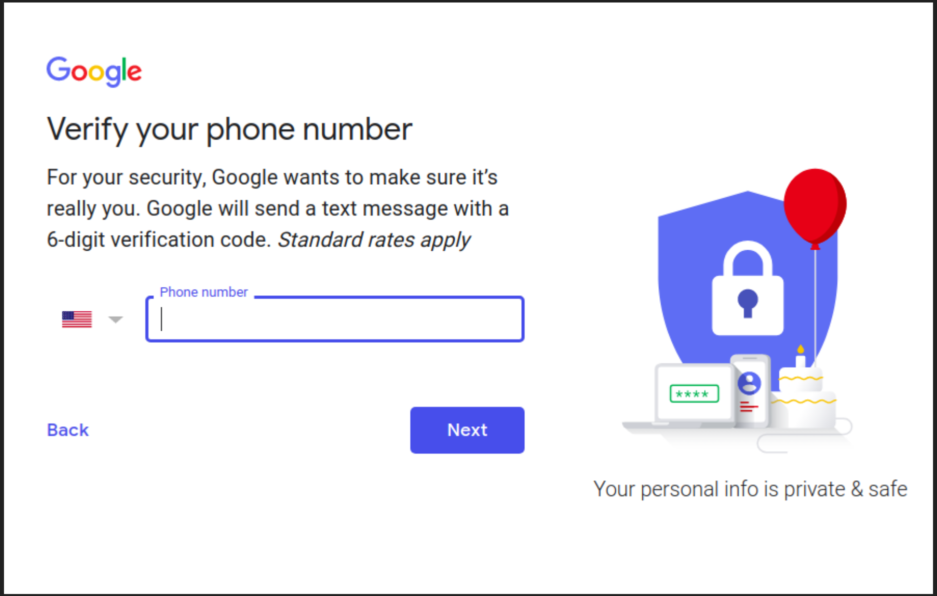
Sign in instead

Next

One account. All of Google working for you.

Create Google Account

- Enter your phone number



Verify your phone number

For your security, Google wants to make sure it's really you. Google will send a text message with a 6-digit verification code. *Standard rates apply*

Phone number

Create Google Account

- Enter your personal information
- Verify your phone number
- Then, scroll down and click on **I Agree** button.

What is Google Colaboratory?

- Google Colaboratory known as google collab is a cloud service, and support TPU and GPU.
- Enhance your Python programming language coding skills
- Develop excellent deep learning models using most popular libraries like TensorFlow, Keras, PyTorch, and OpenCV.
- Do anything without much worrying about packages, libraries, and their installation.

Use Google Colab

- Let’s try!

Loading Data

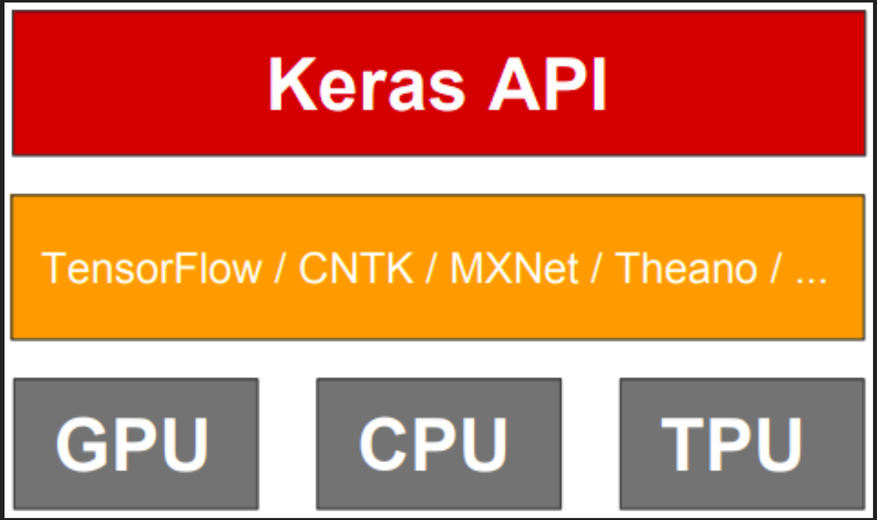
- Colab saves all your Jupyter Notebook to Google Drive
- Colab provides many ways to upload your data into virtual machine on which the code is running.
- When you got disconnected all of your data is lost.

Avoid Data Loading Problem

- Upload your data in your google drive and load the data from google drive
- Let’s try

Keras

- Open-source neural network library written in Python.
- It is running on top of Tensorflow, R, Theano
- Fast experimentation with deep neural network.



Keras is the official high-level API of TensorFlow

- tensorflow.keras (tf.keras) module
- Part of core TensorFlow since v1.4
- Full Keras API
- Better optimized for TF
- Better integration with TF-specific features
    - Estimator API
    - Eager execution
    - etc.

What’s special about Keras?

- A focus on user experience.
- Large adoption in the industry and research community.
- Multi-backend, multi-platform.
- Easy productization of models.

The Keras user experience

Keras is an API designed for human beings, not machines. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error.

This makes Keras easy to learn and easy to use. As a Keras user, you are more productive, allowing you to try more ideas than your competition, faster -- which in turn helps you win machine learning competitions

This ease of use does not come at the cost of reduced flexibility: because Keras integrates with lower-level deep learning languages (in particular TensorFlow), it enables you to implement anything you could have built in the base language. In particular, as tf.keras, the Keras API integrates seamlessly with your TensorFlow workflows.

Keras is multi-backend, multi-platform

- Develop in Python, R
  - On Unix, Windows, OSX
- Run the same code with…
  - TensorFlow
  - CNTK
  - Theano
  - MXNet
  - PlaidML
- CPU, NVIDIA GPU, AMD GPU, TPU...

Largest array of options for productizing models

- TF-Serving
- In-browser, with GPU acceleration (WebKeras, Keras.js, WebDNN…)
- Android (TF, TF Lite), iPhone (native CoreML support)
- Raspberry Pi
- JVM

How to use Keras

Three API styles

- The Sequential Model
  - Dead simple
  - Only for single-input, single-output, sequential layer stacks
  - Good for 70+% of use cases
- The functional API
  - Like playing with Lego bricks
  - Multi-input, multi-output, arbitrary static graph topologies
  - Good for 95% of use cases
- Model subclassing
  - Maximum flexibility
  - Larger potential error surface

The Sequential API

```
from __future__ import absolute_import, division, print_function, unicode_literals
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers


model = keras.Sequential();
model.add(layers.Dense(20, activation='relu', input_shape=(10,)))
model.add(layers.Dense(20, activation='relu'))
model.add(layers.Dense(20, activation='softmax'))


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


model.fit(x, y, epochs=10, batch_size=32)
```

The functional API

```
from __future__ import absolute_import, division, print_function, unicode_literals


import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers


inputs = keras.Input(shape=(10,))
x = layers.Dense(20, activation='relu')(x)
x = layers.Dense(20, activation='relu')(x)
outputs = layers.Dense(10, activation='softmax')(x)


model = keras.Model(inputs, outputs)
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x, y, epochs=10, batch_size=32)
```

Model subclassing

```
from __future__ import absolute_import, division, print_function, unicode_literals
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers


class MyModel(keras.Model):
  def __init__:
    super(MyModel, self).__init__()
    self.dense1 = layers.Dense(20, activation='relu')
    self.dense2 = layers.Dense(20, activation='relu')
    self.dense3 = layers.Dense(10, activation='softmax')


  def call(self, inputs):
    x = self.dense1(x)
    x = self.dense2(x)
    x = self.dense3(x)
    return x


model = MyModel()
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x, y, epochs=10, batch_size=32)
```

Example

Let’s see a real example.

Remember:
use the right tool (API) for the job!

Distributed, multi-GPU, & TPU training

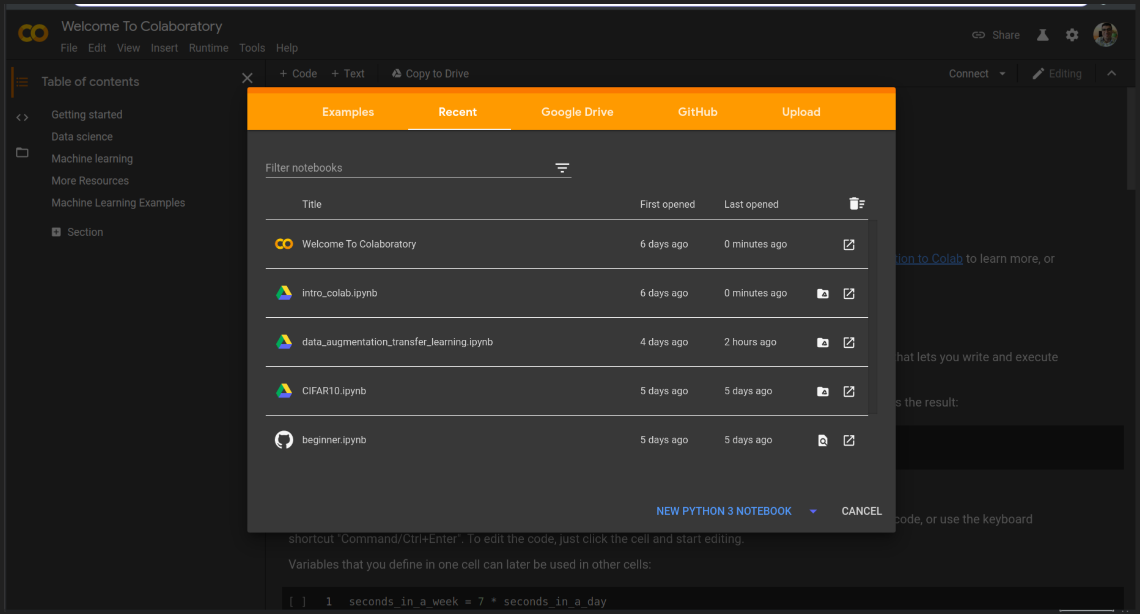
Built-in multi-GPU support

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.applications import Xception
from tensorflow.keras.utils import multi_gpu_model


# Instantiate the model base (we do it on CPU)
with tf.device('/cpu:0'):
  model = Xception(weight=None,
                   input_shape=(height, width, 3),
                   classes=num_classes)

# Replicates the model on 8 GPUS
# This code is assuming your machine has 8 available gpus
parallel_model = multi_gpu_model(model, gpus=8)
parallel_model.compile(loss='categorical_crossentropy', optimizer='rmsprop')


# This `fit` call will be distributed on 8 GPUs
# Since the batch size is 256, each GPU will process 32 samples.
parallel_model.fit(x, y, epochs=20, batch_size=256)
```

Upload Notebook to Colab

Upload Notebook

- Open Google Colab
- Click on Upload tab



Title	First opened	Last opened
Welcome To Colaboratory	6 days ago	0 minutes ago
intro_colab.ipynb	6 days ago	0 minutes ago
data_augmentation_transfer_learning.ipynb	4 days ago	2 hours ago
CIFAR10.ipynb	5 days ago	5 days ago
beginner.ipynb	5 days ago	5 days ago

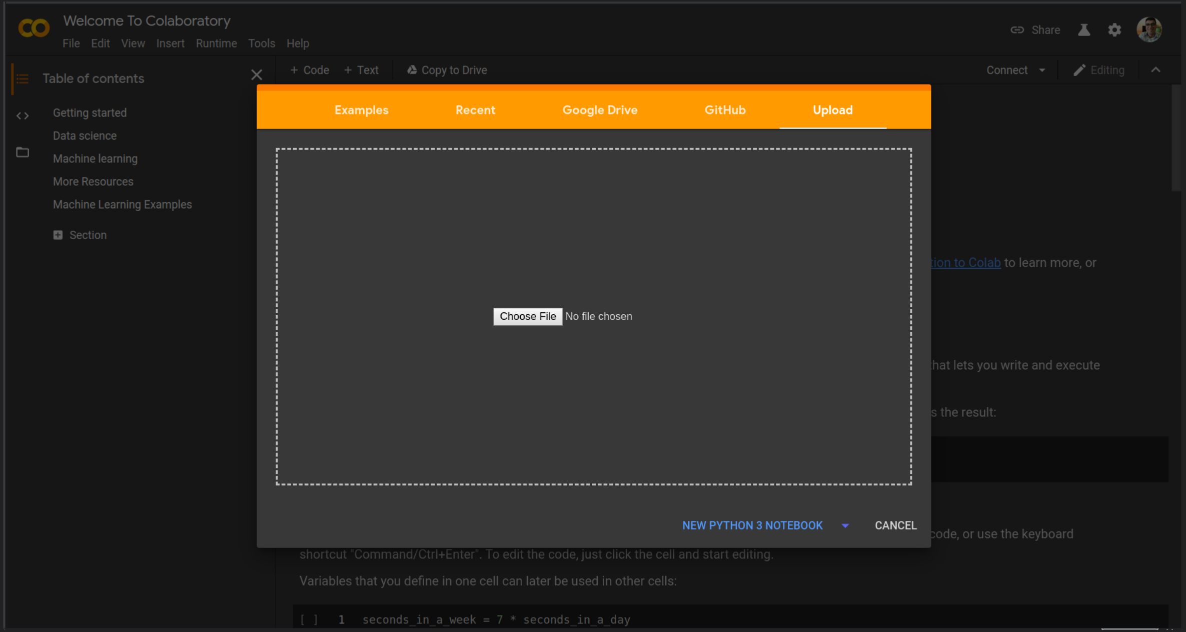
shortcut "Command/Ctrl+Enter". To edit the code, just click the cell and start editing.

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

```
[ ]   seconds_in_a_week = 7 * seconds_in_a_day
```

Upload Notebook

- Click on Choose File
- Select your notebook



ion to Colab to learn more, or

hat lets you write and execute

s the result:

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
```