GOOGLE COLLABORATORY

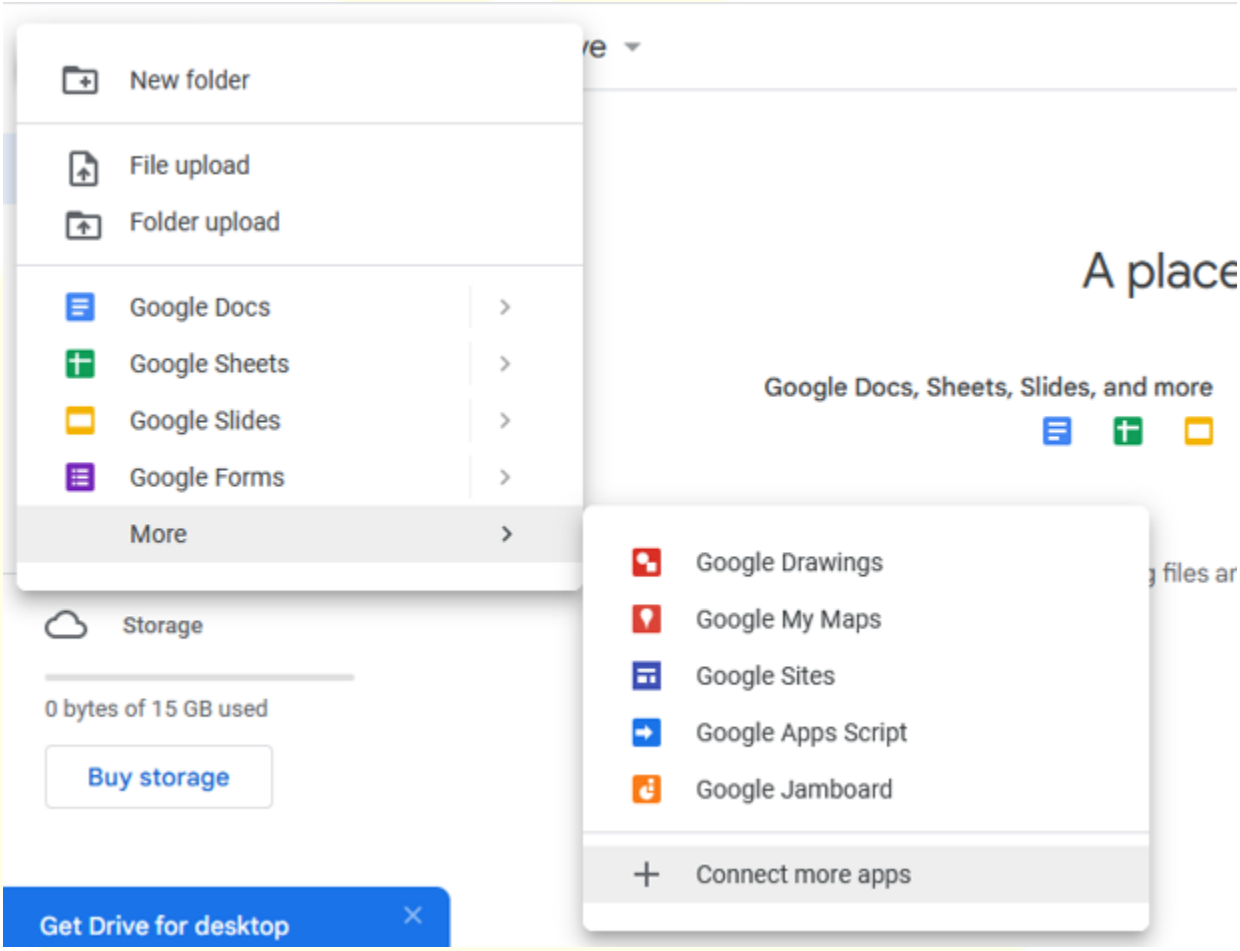
How to configure Google collab with our email drive

2015 E.C
ETHIOP COMPUTING
mail@ethioptec.com or ethiomisgic@gmail.com

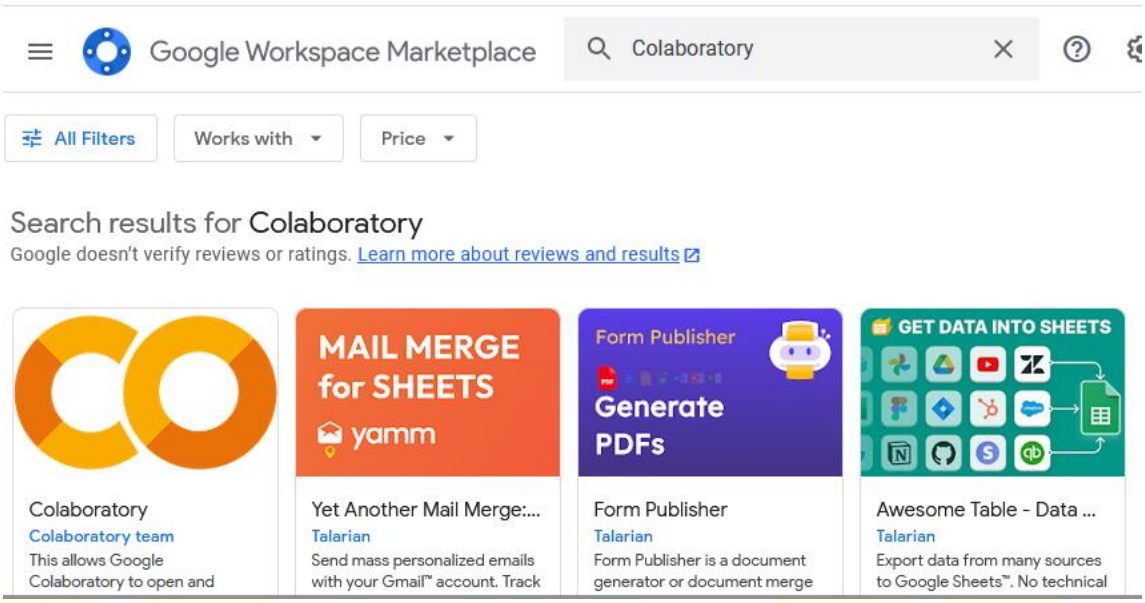
Introduction to Google Collaborator

Setting Up Collaborator in Google drive

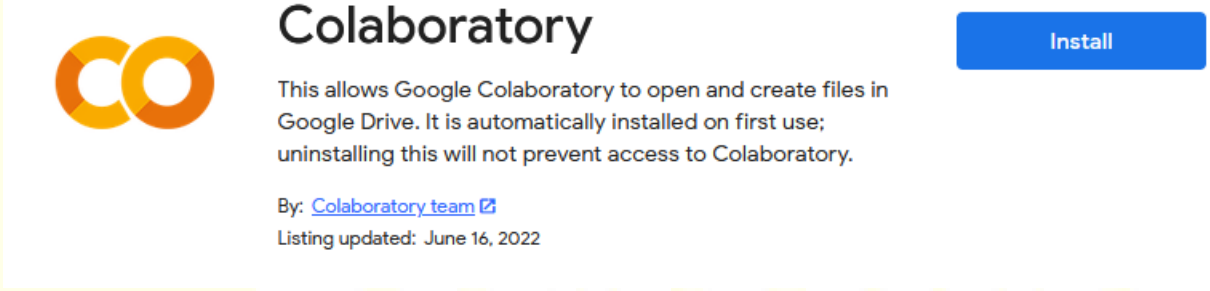
1. Go to your google drive. And click on "**new**" at the top of left side navigator. Click on more as shown



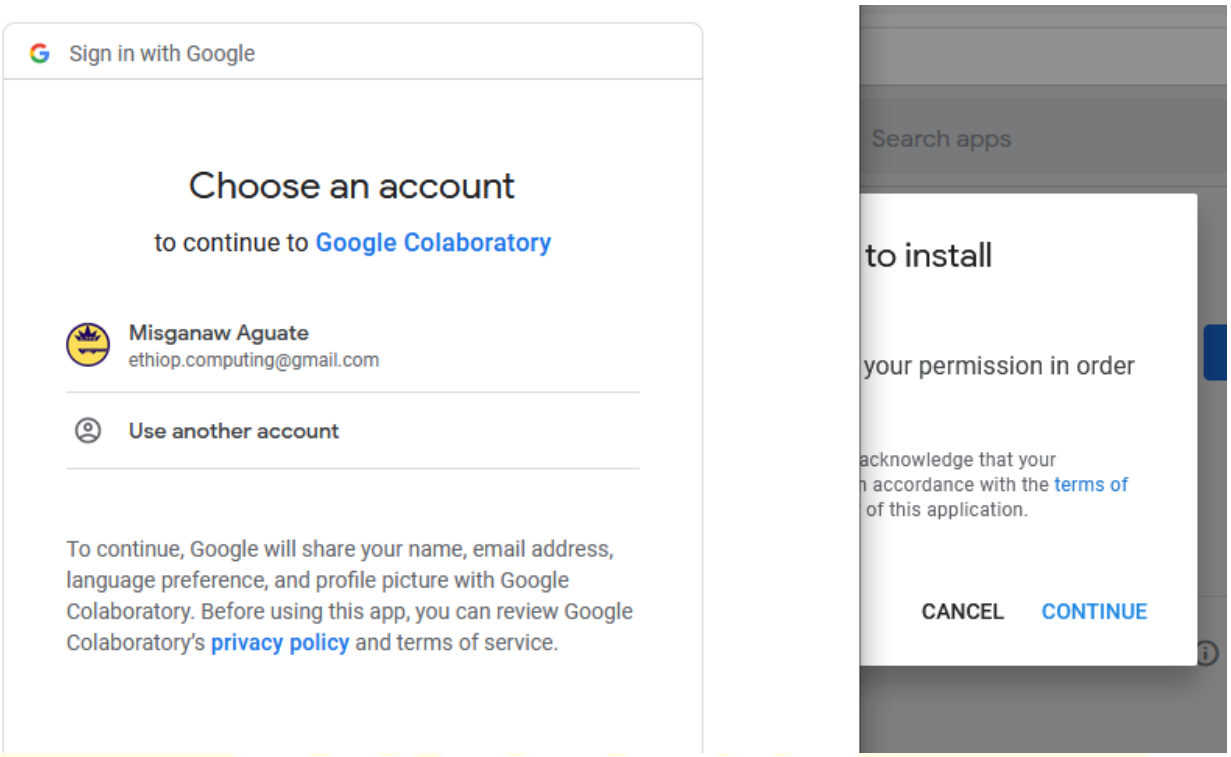
2. Click on "**connect mor apps**".
3. On the search space search "Collaboratory" and click on **collaborator** as shown



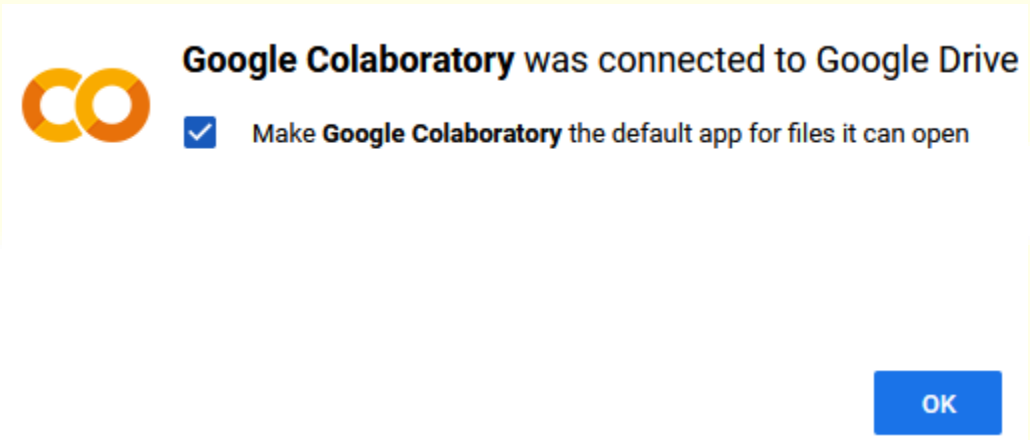
4. Click on **install** as shown



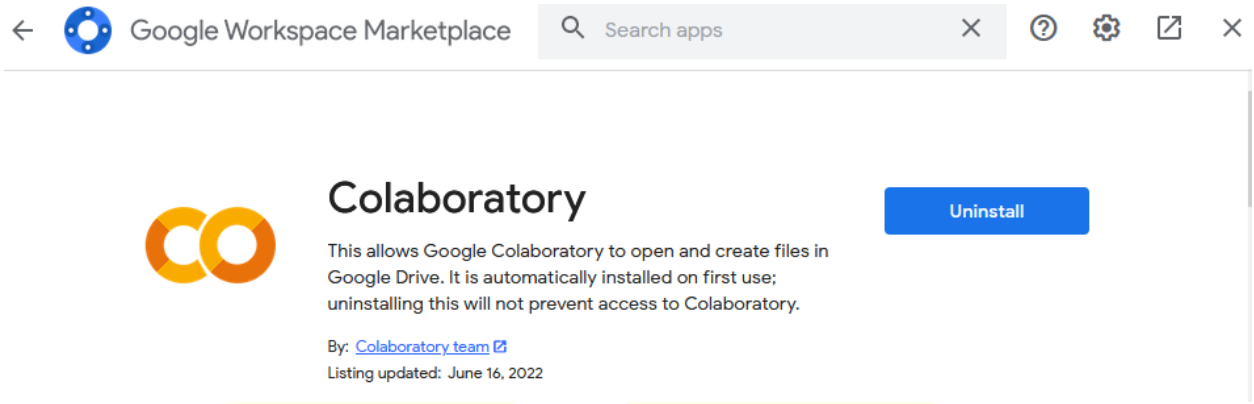
5. When click on install **continue** button will appear and while click on **continue**, it will redirect you to select the email on which the collaborator will installed.



6. Select your mail. And then finally click on **Ok**.

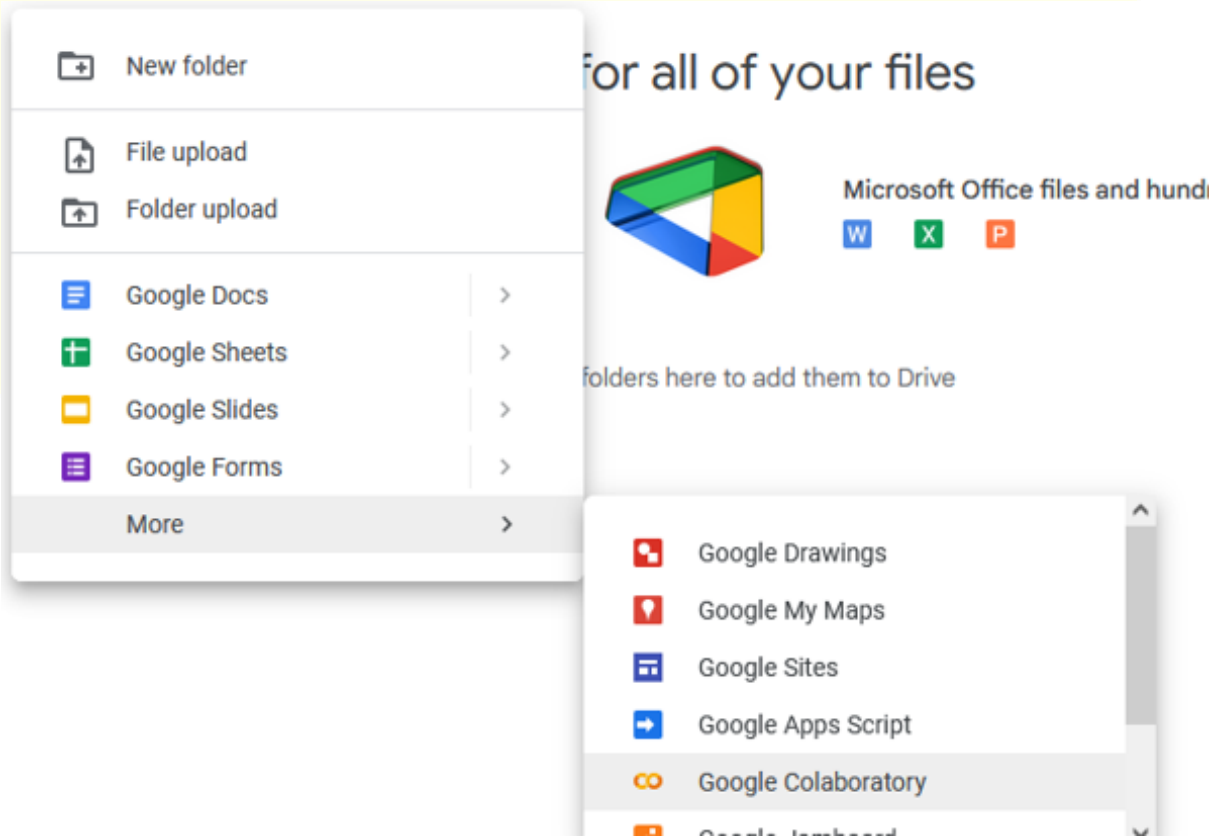


7. After pressing OK. You have to click on **Done**. At the end the following dialog box will appear and never click on uninstall button. Instead close the dialog box and go to your drive to start with colab.

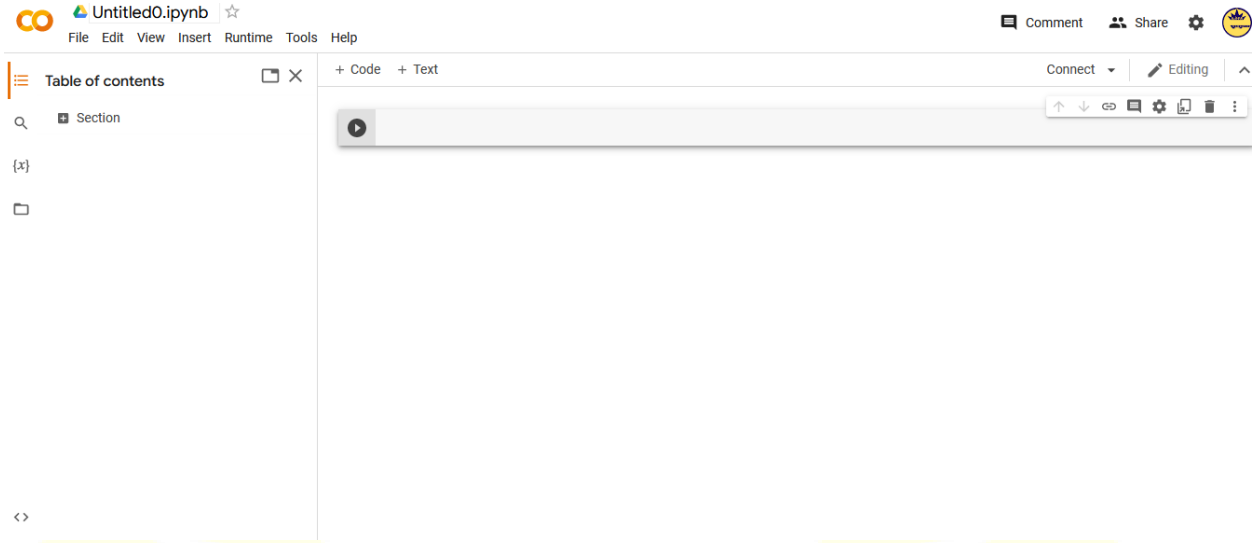


Start with colab

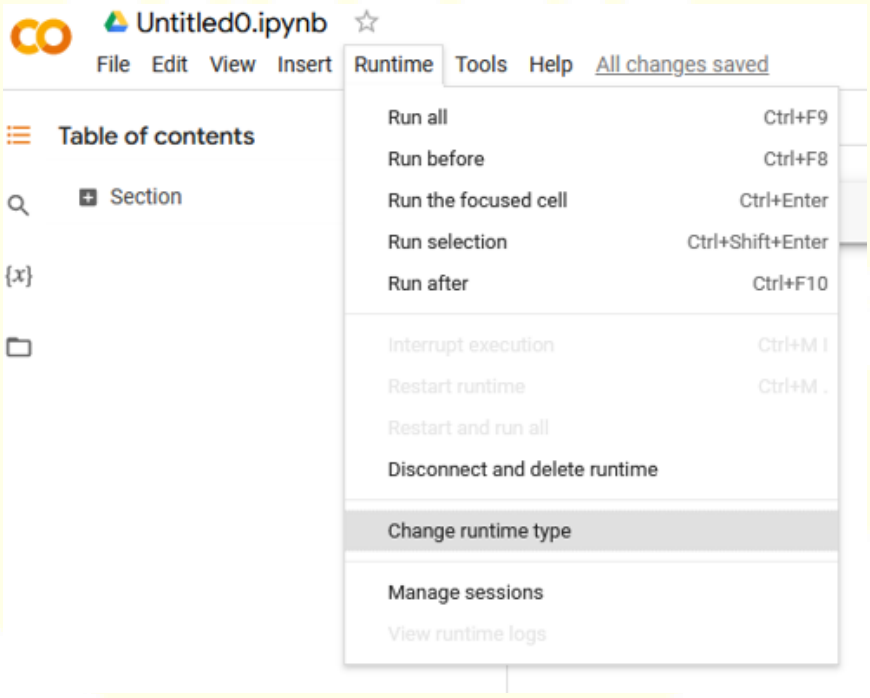
1. Right click on your google drive. Click on **More** and go to Google **Collaboratory** as shown.



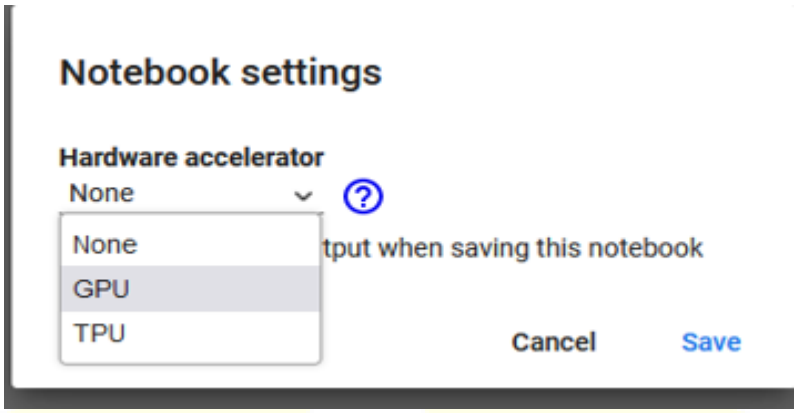
2. Your colab environment will look like the following



3. To use GPU for fast training of the model, click on Runtime. And click on change Run time environment

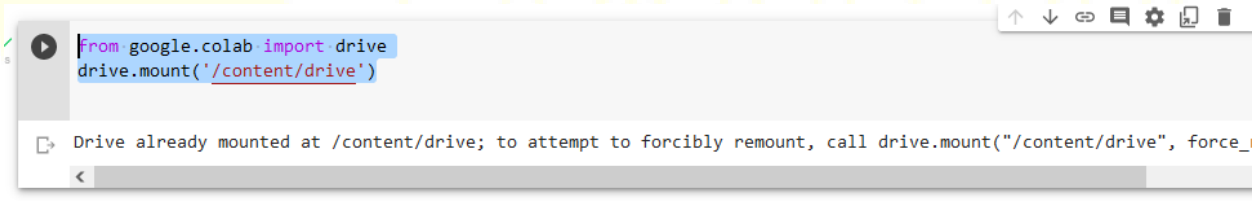


4. When pressing change run time type, the following dialog box will popup. Then select GPU and press Save button

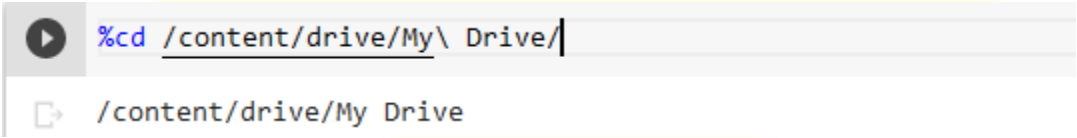


5. To use the selected GPU, you have to click on “connect” at the right top side of the working environment

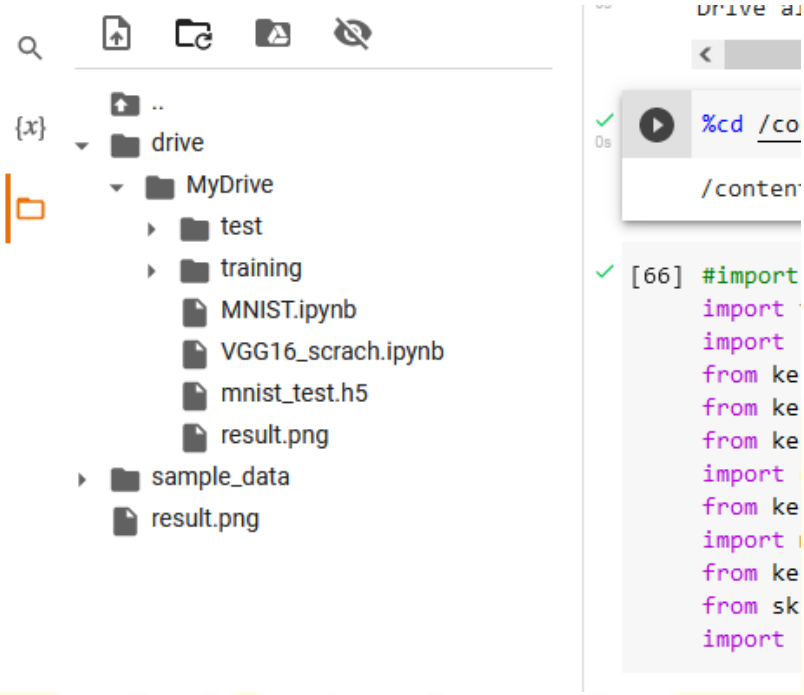
6. Mount your google drive to your colab environment. You can press SHIFT + ENTER to run each individual cell.



7. Enter into your current active directory in google drive



It looks as follows



8. Import necessary deep learning libraries and packages. It depends on your needs to code the algorithm.

```
#import important libraries and packages
import keras
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Activation,BatchNormalization
from keras.layers import Conv2D, MaxPool2D
# from keras.utils import to_categorical
from keras.preprocessing import image
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tqdm import tqdm
%matplotlib inline
```

```
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
from keras.applications.vgg19 import VGG19
from keras.applications.vgg19 import preprocess_input
from keras.applications.mobilenet import MobileNet
from keras.applications.mobilenet import preprocess_input
from keras.preprocessing.image import ImageDataGenerator
from keras import regularizers, optimizers
from keras.optimizers import Adam,Adamax,RMSprop,Adagrad,Adadelta,SGD
from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing import image
import seaborn as sns
import cv2
import os
```

```
from keras.applications import imagenet_utils
from sklearn.metrics import confusion_matrix
import itertools
import shutil
import random
import glob
import warnings
from keras.utils import plot_model
warnings.simplefilter(action ='ignore', category = FutureWarning)
```

Example1: training the model with MNIST dataset

Keep in mind the above code snipes and continue. That means add the following code

Load the MNIST data live as follows. Remember that to run each individual code in a code cell, you have to press SHIFT + ENTER.

```
with tf.device('/gpu:0'):
  (x_train,y_train),(x_test,y_test)=mnist.load_data()
  # Normalize Data
  x_train=x_train/255
  x_test-x_test/255
```

```
[68] print(x_train.shape, y_train.shape)
     print(x_test.shape, y_test.shape)

     (60000, 28, 28) (60000,)
     (10000, 28, 28) (10000,)
```

Show number of the data

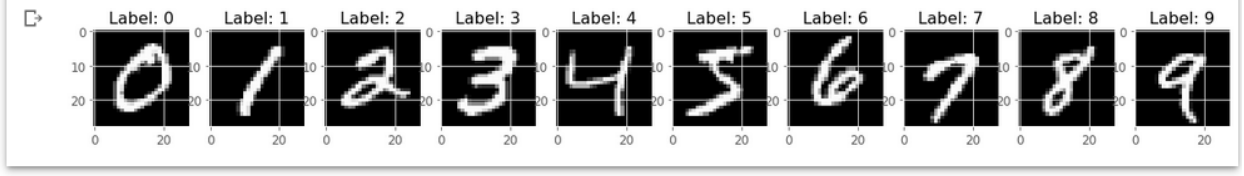
```
print(x_train.shape, y_train.shape)
print(x_test.shape, y_test.shape)
```

```
(60000, 28, 28) (60000,)
(10000, 28, 28) (10000,)
```

Show sample digits in image form

```
num_classes = 10
f, ax = plt.subplots(1, num_classes, figsize=(20,20))

for i in range(0, num_classes):
  sample = x_train[y_train == i][0] #pick the first image from each sample
  ax[i].imshow(sample, cmap='gray')
  ax[i].set_title("Label: {}".format(i), fontsize=16)
```



Show individual data in digit form

```
[78] for i in range(10):
       print(y_train[i])
```

```
5
0
4
1
9
2
1
3
1
4
```

Change train data and test data to categorical format

```
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

Show the category

```
for i in range(10):
  print(y_train[i])
```

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

Reshape the data to one dimensional array

```
[80] # Reshape Data
     x_train = x_train.reshape(x_train.shape[0], -1) #return the same number of data in one dimension
     x_test = x_test.reshape(x_test.shape[0], -1)
     print(x_train.shape)
```

```
(60000, 784)
```

Build the model and show its structure

```
#Create Model - Fully Connected Neural Network
model = Sequential()
model.add(Dense(units=128, input_shape=(784,), activation='relu'))
model.add(Dense(units=128, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(units=10, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_11 (Dense)            (None, 128)               100480

 dense_12 (Dense)            (None, 128)               16512

 dropout_3 (Dropout)         (None, 128)               0

 dense_13 (Dense)            (None, 10)                1290

=================================================================
Total params: 118,282
Trainable params: 118,282
Non-trainable params: 0
```

Train the model

```
#train th emodel
batch_size = 512
epochs=3
train=model.fit(x=x_train, y=y_train, batch_size=batch_size, epochs=epochs)
```

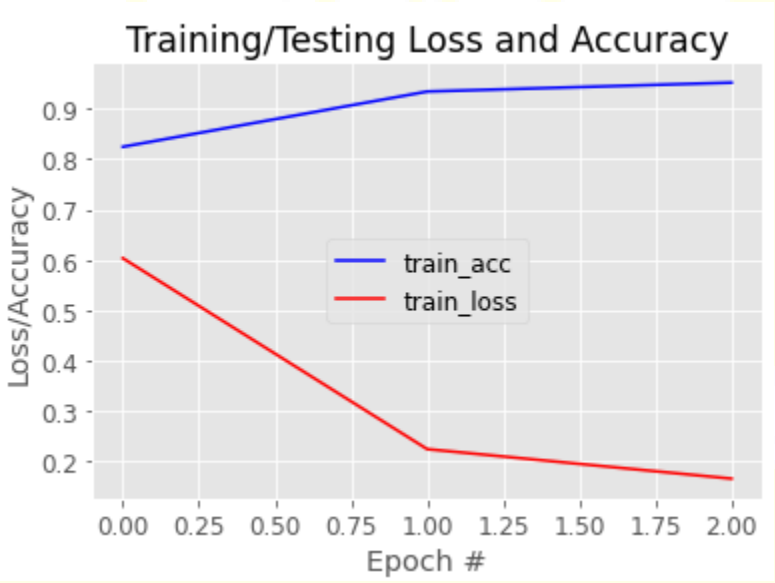
```
Epoch 1/3
118/118 [==============================] - 1s 3ms/step - loss: 0.6037 - accuracy: 0.8247
Epoch 2/3
118/118 [==============================] - 0s 4ms/step - loss: 0.2248 - accuracy: 0.9344
Epoch 3/3
118/118 [==============================] - 0s 4ms/step - loss: 0.1663 - accuracy: 0.9520
```

Analyze accuracy and loss using graph

```
# plot the training loss and accuracy
plt.style.use("ggplot")
plt.figure()
N = 3
plt.rcParams['font.size'] = '12'

plt.plot(np.arange(0, N), train.history["accuracy"], 'b', label="train_acc")
# plt.plot(np.arange(0, N), train.history["val_accuracy"], 'g', label="val_acc")
plt.plot(np.arange(0, N), train.history["loss"], 'r',label="train_loss")
# plt.plot(np.arange(0, N), train.history["val_loss"], 'orange', label="val_loss")

plt.title("Training/Testing Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="center")
plt.savefig('result.png')
```



Find testing accuracy and testing loss

```
#Evaluate
test_loss, test_acc = model.evaluate(x_test, y_test)
print("Test Loss: {}, Test Accuracy: {}".format(test_loss, test_acc))
```

```
313/313 [==============================] - 1s 2ms/step - loss: 16.8844 - accuracy: 0.9605
Test Loss: 16.88435173034668, Test Accuracy: 0.9605000019073486
```

Save the designed model

```
#SAVE THE MODEL
model.save('mnist_test.h5')
```

Fined predication values

```
# prediction values
y_pred = model.predict(x_test)
y_pred_classes = np.argmax(y_pred, axis=1)
print(y_pred)

print('Predicted classes ',y_pred_classes)
```

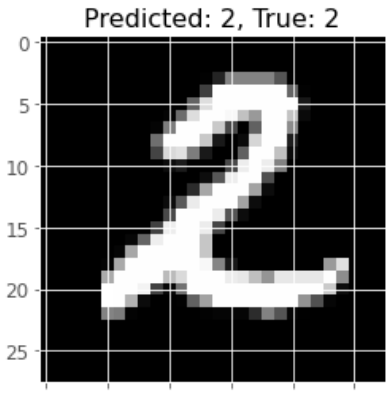
```
313/313 [==============================] - 1s 2ms/step
[[0. 0. 0. ... 1. 0. 0.]
 [0. 0. 1. ... 0. 0. 0.]
 [0. 1. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
Predicted classes  [7 2 1 ... 4 5 6]
```

Show sample predication

```
# Single Example
random_idx = np.random.choice(len(x_test))
x_sample = x_test[random_idx]
y_true = np.argmax(y_test, axis=1)
y_sample_true = y_true[random_idx]
y_sample_pred_class = y_pred_classes[random_idx]

plt.title("Predicted: {}, True: {}".format(y_sample_pred_class, y_sample_true), fontsize=16)
plt.imshow(x_sample.reshape(28, 28), cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f6b9644cd30>
```



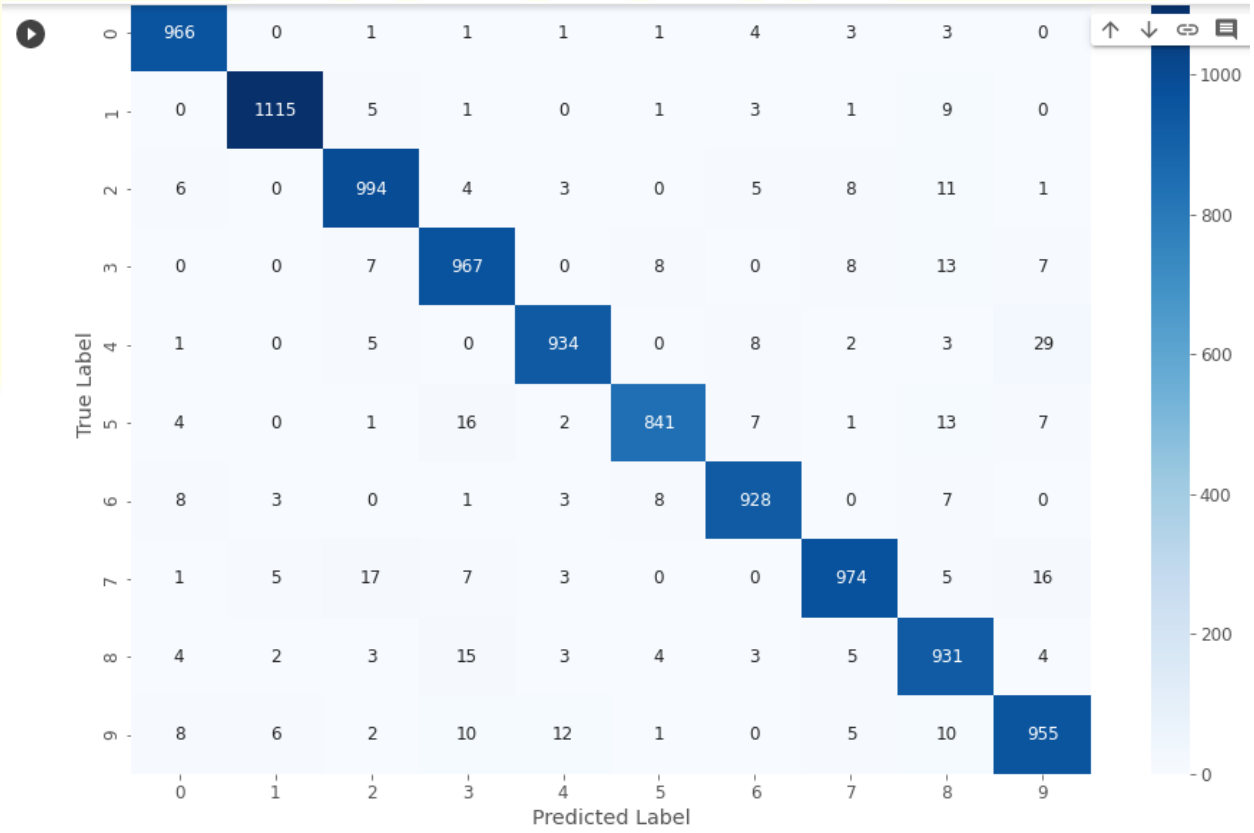
Evaluate the model using confusion matrix

```
#Confusion Matrix. Medel evaluation

confusion_mtx = confusion_matrix(y_true, y_pred_classes)

# Plot
fig, ax = plt.subplots(figsize=(15,10))
ax = sns.heatmap(confusion_mtx, annot=True, fmt='d', ax=ax, cmap="Blues")
ax.set_xlabel('Predicted Label')
ax.set_ylabel('True Label')
ax.set_title('Confusion Matrix');
```

The following figure shows the result true value and predicted value of each category. The Y-axis represents the predicted value. The X-axis represents the true (truth) value.



Show classification report. Calculating precision, recall and F1-score for each category.

```
#classification report
from sklearn.metrics import classification_report
# Print the precision and recall, among other metrics
print(classification_report(y_true, y_pred_classes, digits=3))
```

```
              precision    recall  f1-score   support

           0      0.968     0.986     0.977       980
           1      0.986     0.982     0.984      1135
           2      0.960     0.963     0.962      1032
           3      0.946     0.957     0.952      1010
           4      0.972     0.951     0.961       982
           5      0.973     0.943     0.958       892
           6      0.969     0.969     0.969       958
           7      0.967     0.947     0.957      1028
           8      0.926     0.956     0.941       974
           9      0.937     0.946     0.942      1009

    accuracy                          0.961     10000
   macro avg      0.961     0.960     0.960     10000
weighted avg      0.961     0.961     0.961     10000
```

Building Big CNN (Design VGG16 Model)

Do not forget mounting your google drive to google Colab environment and import necessary library and packages

```
# get data from training and test folder. ImageDataGenerator will automatically label all the data inside cat
#folder as cat and vis-à-vis for dog folder
trdata = ImageDataGenerator()
traindata = trdata.flow_from_directory(directory="training",target_size=
(32,32))
tsdata = ImageDataGenerator()
testdata = tsdata.flow_from_directory(directory="test", target_size=
(32,32))
```

```
Found 8005 images belonging to 2 classes.
Found 2023 images belonging to 2 classes.
```

Build CNN

```
#builde the CNN
model = Sequential()
#2 x convolution layer of 64 channel of 3x3 kernal and same padding
model.add(Conv2D(input_shape=(32,32,3),filters=64,kernel_size=(3,3),padding="same", activation="relu"))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same",activation="relu"))
#1 x maxpool layer of 2x2 pool size and stride 2x2
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))

#2 x convolution layer of 128 channel of 3x3 kernal and same padding
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same",activation="relu"))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same",activation="relu"))
#1 x maxpool layer of 2x2 pool size and stride 2x2
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))

#3 x convolution layer of 256 channel of 3x3 kernal and same padding
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same",activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same",activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same",activation="relu"))
#1 x maxpool layer of 2x2 pool size and stride 2x2
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
```

```
#3 x convolution layer of 512 channel of 3x3 kernal and same padding
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",activation="relu"))
#1 x maxpool layer of 2x2 pool size and stride 2x2
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
```

```
#3 x convolution layer of 512 channel of 3x3 kernal and same padding
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",activation="relu"))
#1 x maxpool layer of 2x2 pool size and stride 2x2
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
```

```
#Classification layer. includes flatten layer and dens layer
model.add(Flatten())
#2x Dense layer of 4096 units
model.add(Dense(units=4096,activation="relu"))
model.add(Dense(units=4096,activation="relu"))

model.add(BatchNormalization()) # addition of BN layer
#1 x Dense Softmax layer of 2 units
model.add(Dense(units=2, activation="softmax"))
```

```
#compile the model
from keras.optimizers import Adam
opt = Adam(lr=0.001)
model.compile(optimizer=opt,
loss=keras.losses.categorical_crossentropy, metrics=['accuracy'])
```

```
#model summary. show model parameter
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_33 (Conv2D)          (None, 32, 32, 64)        1792

 conv2d_34 (Conv2D)          (None, 32, 32, 64)        36928

 max_pooling2d_13 (MaxPoolin  (None, 16, 16, 64)       0
 g2D)

 conv2d_35 (Conv2D)          (None, 16, 16, 128)       73856

 conv2d_36 (Conv2D)          (None, 16, 16, 128)       147584

 max_pooling2d_14 (MaxPoolin  (None, 8, 8, 128)        0
 g2D)

 conv2d_37 (Conv2D)          (None, 8, 8, 256)         295168

 conv2d_38 (Conv2D)          (None, 8, 8, 256)         590080

 conv2d_39 (Conv2D)          (None, 8, 8, 256)         590080
```

Train the model

```
#train the model
with tf.device('/gpu:0'):
  from keras.callbacks import ModelCheckpoint, EarlyStopping
  checkpoint = ModelCheckpoint("vgg16_1.h5", monitor='val_acc',
  verbose=1, save_best_only=True, save_weights_only=False, mode='auto',period=1)
  early = EarlyStopping(monitor='val_acc', min_delta=0, patience=20,verbose=1, mode='auto')
  hist = model.fit_generator(steps_per_epoch=100,generator=traindata,validation_data= testdata,
                             validation_steps=10,epochs=10,callbacks=[checkpoint,early])
```

```
WARNING:tensorflow:`period` argument is deprecated. Please use `save_freq` to specify the freque
Epoch 1/10
<ipython-input-23-b44550748d2d>:7: UserWarning: `Model.fit_generator` is deprecated and will be
  hist = model.fit_generator(steps_per_epoch=100,generator=traindata,validation_data= testdata,
100/100 [==============================] - ETA: 0s - loss: 5.7386 - accuracy: 0.5103 WARNING:ter
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Availa
100/100 [==============================] - 2073s 21s/step - loss: 5.7386 - accuracy: 0.5103 - va
Epoch 2/10
 43/100 [===========>..................] - ETA: 10:11 - loss: 0.6931 - accuracy: 0.5167
```

```
#plot the graph to analyse
import matplotlib.pyplot as plt
plt.plot(hist.history["acc"])
plt.plot(hist.history['val_acc'])
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title("model accuracy")
plt.ylabel("Accuracy")
plt.xlabel("Epoch")
plt.legend(["Accuracy","Validation Accuracy","loss","Validation Loss"])
plt.show()
```

```
#predict
from keras.preprocessing import image
img = image.load_img("image.jpeg",target_size=(32,32))
img = np.asarray(img)
plt.imshow(img)
img = np.expand_dims(img, axis=0)
from keras.models import load_model
saved_model = load_model("vgg16_1.h5")
output = saved_model.predict(img)
if output[0][0] > output[0][1]:
print("cat")
else:
print('dog')
```

Transfer Learning

Import the data

```
img_height, img_width = 128, 128
batch_size = 32
# train=ImageDataGenerator(rescale=1/255)
# val=ImageDataGenerator(rescale=1/255)
# test=ImageDataGenerator(rescale=1/255)
```

```
train_ds = tf.keras.utils.image_dataset_from_directory('training',image_size =
                                                       (img_height, img_width),batch_size = batch_size)
val_ds = tf.keras.utils.image_dataset_from_directory('validat',image_size =
                                                     (img_height, img_width),batch_size = batch_size)
test_ds = tf.keras.utils.image_dataset_from_directory('test',image_size =
                                                      (img_height, img_width),batch_size = batch_size)
```

```
Found 7603 files belonging to 2 classes.
Found 402 files belonging to 2 classes.
Found 2023 files belonging to 2 classes.
```

```
class_names = ["cats", "dogs"]
plt.figure(figsize=(10,10))
for images, labels in train_ds.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```

Download pretrained model

```
from keras import models
from keras import layers
#Load the MobileNet model
IMAGE_SIZE=[128,128]
Mobilenet = MobileNet(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
# Freeze the layers except the last 4 layers
for layer in Mobilenet.layers[:-2]:
  layer.trainable = False
# Create the model
with tf.device('/gpu:0'):
  mobilenet = tf.keras.models.Sequential([tf.keras.layers.Rescaling(1./255)])
  mobilenet.add(Mobilenet)
  # Add new layers
  mobilenet.add(layers.Flatten())
  mobilenet.add(layers.Dense(1024, activation='relu'))
  mobilenet.add(layers.Dense(1024, activation='relu'))
  mobilenet.add(layers.Dense(1024, activation='relu')) #addition of layer
  mobilenet.add(BatchNormalization())                  # addition of BN layer
  mobilenet.add(layers.Dense(2))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet/mobilenet_1_0_128_t
17225924/17225924 [==============================] - 0s 0us/step
```

Compile the model

```
lr = 0.01
epoch = 5
opt = Adam(learning_rate = lr, decay = 1e-6)
```

```
mobilenet.compile(optimizer=opt, loss=tf.losses.SparseCategoricalCrossentropy(from_logits = True),
                  metrics=['accuracy'])
```

Train the model

```
with tf.device("/gpu:0"):
  history=mobilenet.fit(train_ds,validation_data = val_ds,epochs = epoch)
```

```
Epoch 1/5
238/238 [==============================] - 662s 2s/step - loss: 0.1558 - accuracy: 0.9448 - val_loss: 0.0563
Epoch 2/5
238/238 [==============================] - 159s 667ms/step - loss: 0.0702 - accuracy: 0.9746 - val_loss: 0.07
Epoch 3/5
238/238 [==============================] - 156s 655ms/step - loss: 0.0458 - accuracy: 0.9828 - val_loss: 0.09
Epoch 4/5
238/238 [==============================] - 157s 655ms/step - loss: 0.0326 - accuracy: 0.9875 - val_loss: 0.08
Epoch 5/5
238/238 [==============================] - 157s 660ms/step - loss: 0.0335 - accuracy: 0.9886 - val_loss: 0.19
```

Save the model

```
mobilenet.save('pretrained.h5')
```

Calculate mean accuracy and loss

```
mobilenet.save('pretrained.h5')
```

```
np.mean(history.history['accuracy'])
```

```
0.9756412029266357
```

```
np.mean(history.history['val_accuracy'])
```

```
0.9666666746139526
```

```
np.mean(history.history['loss'])
```

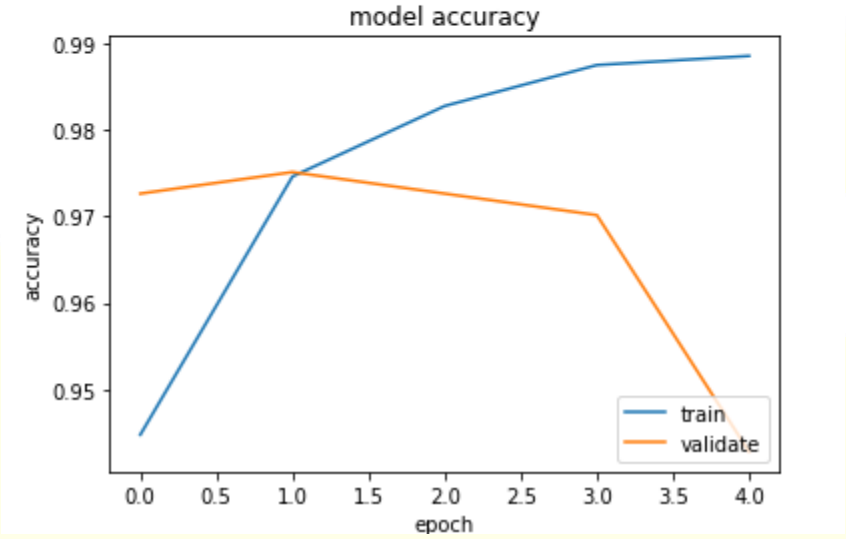
```
0.06758365780115128
```

```
np.mean(history.history['val_loss'])
```

```
0.10110663399100303
```

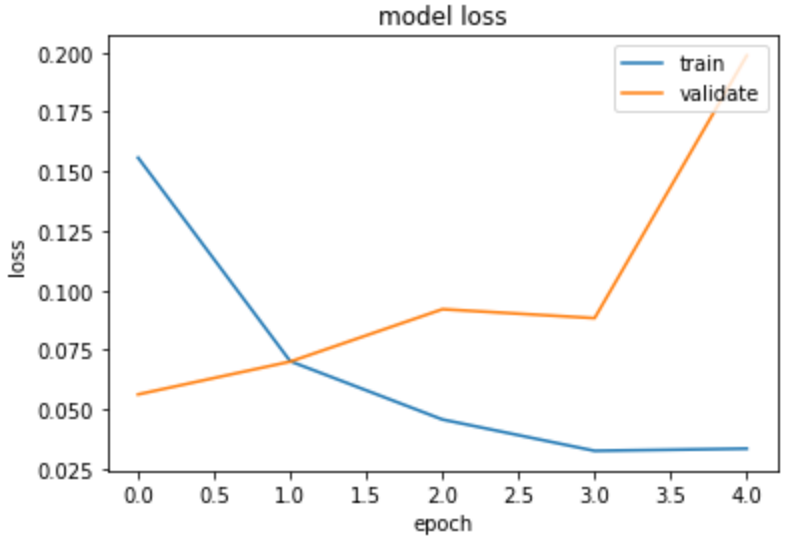
Draw accuracy Vs Epoch graph

```
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validate'], loc='lower right')
plt.show()
```



Draw loss Vs Epoch graph

```
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validate'], loc='upper right')
plt.show()
```



Evaluate the model

```
mobilenet.evaluate(test_ds)
```

```
64/64 [==============================] - 203s 3s/step - loss: 0.1246 - accuracy: 0.9634
[0.12459906935691833, 0.9634206891059875]
```

```
predictions = mobilenet.predict(test_ds)
```

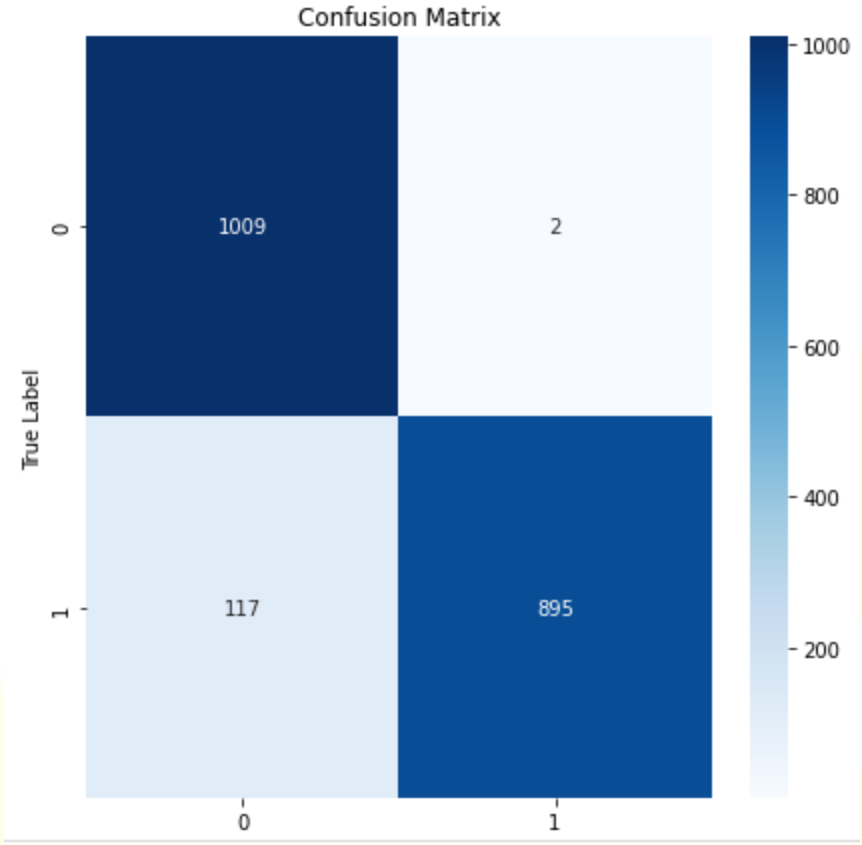
Confusion matrix

```
y_pred = []
y_true = []
# iterate over the dataset
for image_batch, label_batch in test_ds:   # use dataset.unbatch() with repeat
   # append true labels
   y_true.append(label_batch)
   # compute predictions
   preds = mobilenet.predict(image_batch)
   # append predicted labels
   y_pred.append(np.argmax(preds, axis = - 1))
# convert the true and predicted labels into tensors
true_labels = tf.concat([item for item in y_true], axis = 0)
predicted_labels = tf.concat([item for item in y_pred], axis = 0)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(true_labels, predicted_labels)
print(cm)
```

```
[[1009    2]
 [ 117  895]]
```

Draw the confusion matrix

```
# Plot
fig, ax = plt.subplots(figsize=(15,10))
ax = sns.heatmap(cm, annot=True, fmt='d', ax=ax, cmap="Blues")
ax.set_xlabel('Predicted Label')
ax.set_ylabel('True Label')
ax.set_title('Confusion Matrix');
```



Classification report

```
from sklearn.metrics import classification_report
target_names = ['cats','dogs']
print(classification_report(true_labels, predicted_labels))
```

```
              precision    recall  f1-score   support

           0       0.90      1.00      0.94      1011
           1       1.00      0.88      0.94      1012

    accuracy                           0.94      2023
   macro avg       0.95      0.94      0.94      2023
weighted avg       0.95      0.94      0.94      2023
```

Make prediction

```
import numpy

plt.figure(figsize=(10,10))
for images, labels in test_ds.take(1):
  classifications = mobilenet(images)
  # print(classifications)

  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    index = numpy.argmax(classifications[i])
    plt.title("Pred: " + class_names[index] + " | Real: " + class_names[labels[i]])
```

Convert the model to TFLite

This helps to deploy the model on mobile application

```
converter = tf.lite.TFLiteConverter.from_keras_model(mobilenet)
tflite_model = converter.convert()

with open("model.tflite", 'wb') as f:
  f.write(tflite_model)
```