PMI Lab
Pattern Recognition and Machine Intelligence

Hankuk University of Foreign Studies

Implementing DCNN using Tensorflow

Prof. Jae Young Choi

Pattern Recognition and Machine Intelligence Lab. (PMI)

Hankuk University of Foreign Studies

2019

How to install Tensorflow

- **Open Terminal**
  $ sudo apt-get update

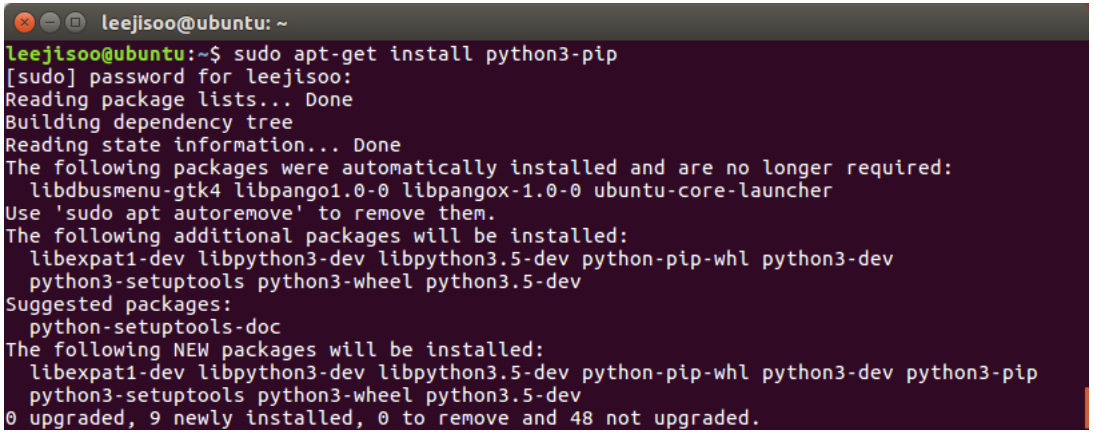
```
leejisoo@ubuntu: ~
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

leejisoo@ubuntu:~$ sudo apt-get update
[sudo] password for leejisoo:
Hit:1 http://us.archive.ubuntu.com/ubuntu xenial InRelease
Hit:2 http://us.archive.ubuntu.com/ubuntu xenial-updates InRelease
Hit:3 http://us.archive.ubuntu.com/ubuntu xenial-backports InRelease
Hit:4 http://security.ubuntu.com/ubuntu xenial-security InRelease
*** Error in `appstreamcli': double free or corruption (fasttop): 0x0000000001f1e570 ***
======= Backtrace: =========
/lib/x86_64-linux-gnu/libc.so.6(+0x77725)[0x7fa24bb97725]
/lib/x86_64-linux-gnu/libc.so.6(+0x7ff4a)[0x7fa24bb9ff4a]
/lib/x86_64-linux-gnu/libc.so.6(cfree+0x4c)[0x7fa24bba3abc]
/usr/lib/x86_64-linux-gnu/libappstream.so.3(as_component_complete+0x439)[0x7fa24bf1bd19]
/usr/lib/x86_64-linux-gnu/libappstream.so.3(as_data_pool_update+0x44a)[0x7fa24bf1cf0a]
/usr/lib/x86_64-linux-gnu/libappstream.so.3(as_cache_builder_refresh+0x1c2)[0x7fa24bf1227
```

$ sudo apt-get upgrade python3

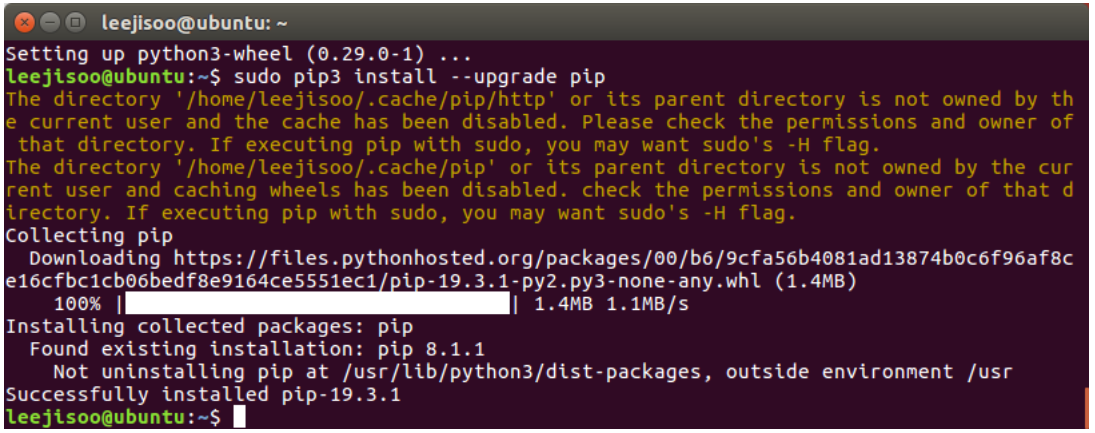
```
leejisoo@ubuntu: ~
leejisoo@ubuntu:~$ sudo apt-get upgrade python3
Reading package lists... Done
Building dependency tree
Reading state information... Done
python3 is already the newest version (3.5.1-3).
Calculating upgrade... Done
The following packages were automatically installed and are no longer required:
  libdbusmenu-gtk4 libpango1.0-0 libpangox-1.0-0 ubuntu-core-launcher
Use 'sudo apt autoremove' to remove them.
The following packages have been kept back:
  cups-filters cups-filters-core-drivers gir1.2-javascriptcoregtk-4.0
  gir1.2-webkit2-4.0 gnome-software gnome-software-common libdrm-amdgpu1
  libdrm2 libegl1-mesa libgbm1 libgl1-mesa-dri libgl1-mesa-glx libglapi-mesa
  libinput10 libjavascriptcoregtk-4.0-18 libmirclient9 libmm-glib0
  liboxideqt-qmlplugin liboxideqtcore0 liboxideqtquick0 libqmi-proxy
  libwayland-egl1-mesa libwebkit2gtk-4.0-37 libwebkit2gtk-4.0-37-gtk2
  libxatracker2 linux-generic linux-headers-generic linux-image-generic
```

How to install Tensorflow

- **Open Terminal**
  $ sudo apt-get install python3-pip

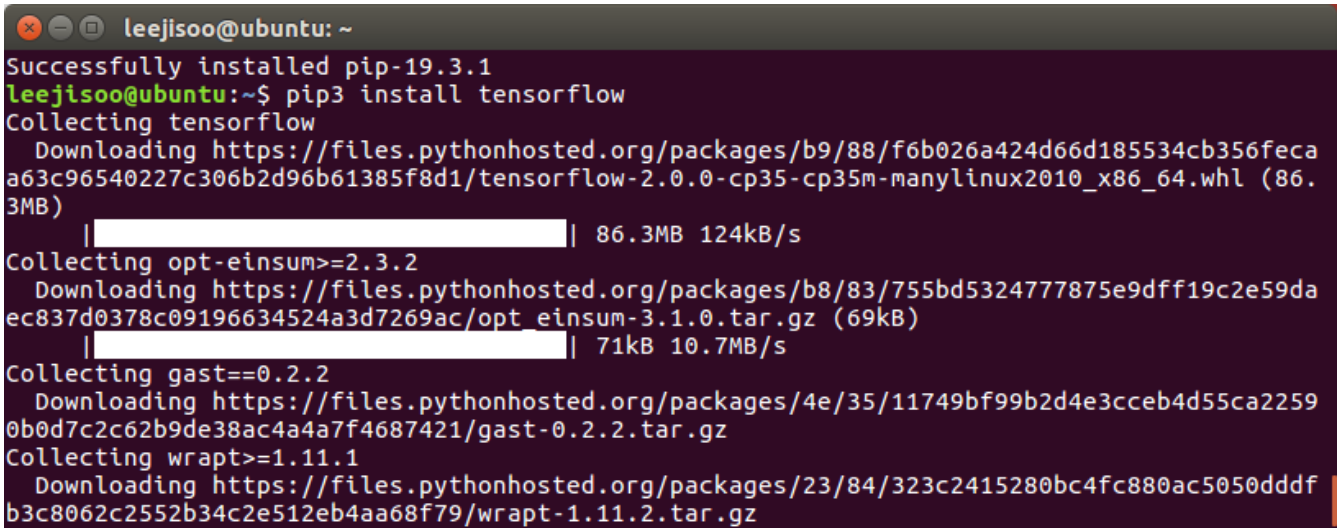


$ sudo pip3 install –upgrade pip



How to install Tensorflow

- **Open Terminal**
  $ pip3 install tensorflow



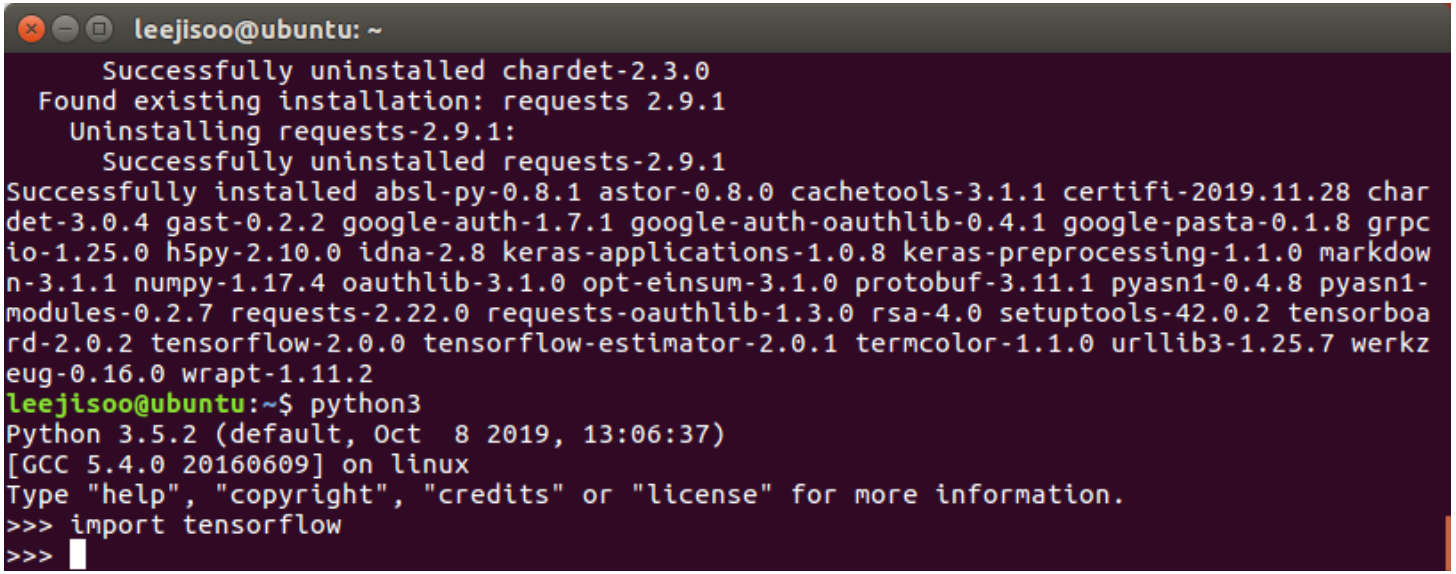
If you get the following error:
ERROR: Could not install packages due to an EnvironmentError: [Errno 13] Permission denied: '/usr/local/lib/python3.5/dist-packages/numpy'
Consider using the `--user` option or check the permissions.

Enter the following command:
$ sudo pip3 install tensorflow

How to install Tensorflow

- **Open Terminal**
  $ python3

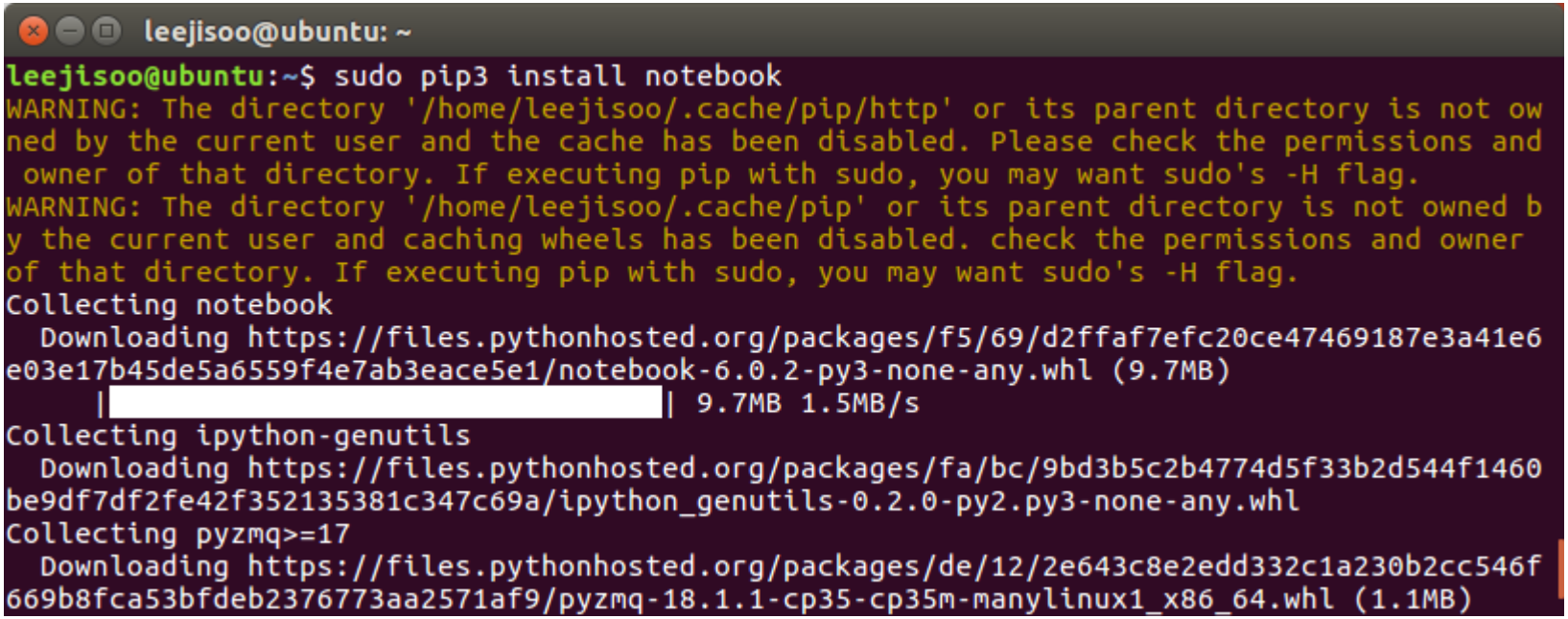
>>> import tensorflow



Notice: If you get an error when importing tensorflow,
it is not installed correctly.

How to install Jupyter notebook

- **Open Terminal**
  $ sudo pip3 install notebook



How to use Jupyter notebook

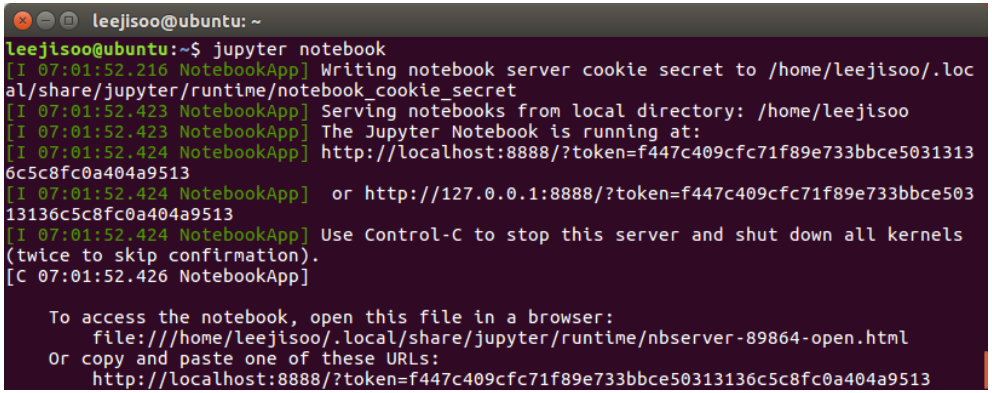
- **Jupyter notebook**

  - Congratulations! All installation and setup is complete.

  - Now, let's write and execute the code easily with Jupyter notebook.

  - Enter the following command in the cmd.

  $ Jupyter notebook



This command is a "run" command, not a jupyter installation.

How to use Jupyter notebook

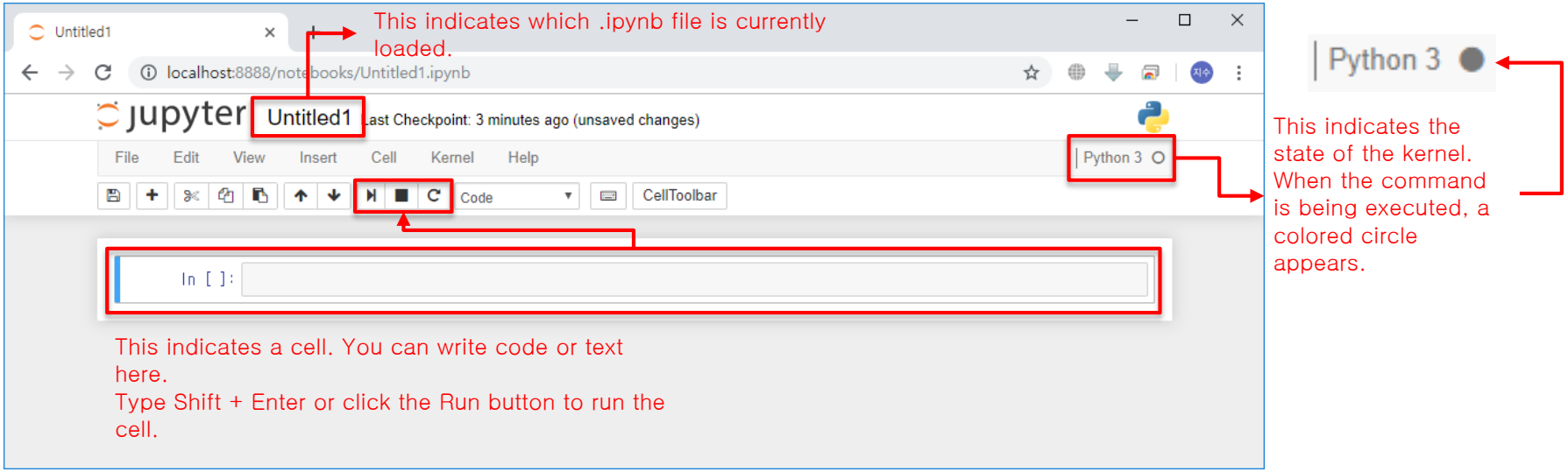
- **Jupyter notebook**

  - jupyter notebook is a very useful web application. You can write, modify, and execute code in the form of a Web.

  - Enter the desired path and create a new .ipynb file via [new] - [python 3].

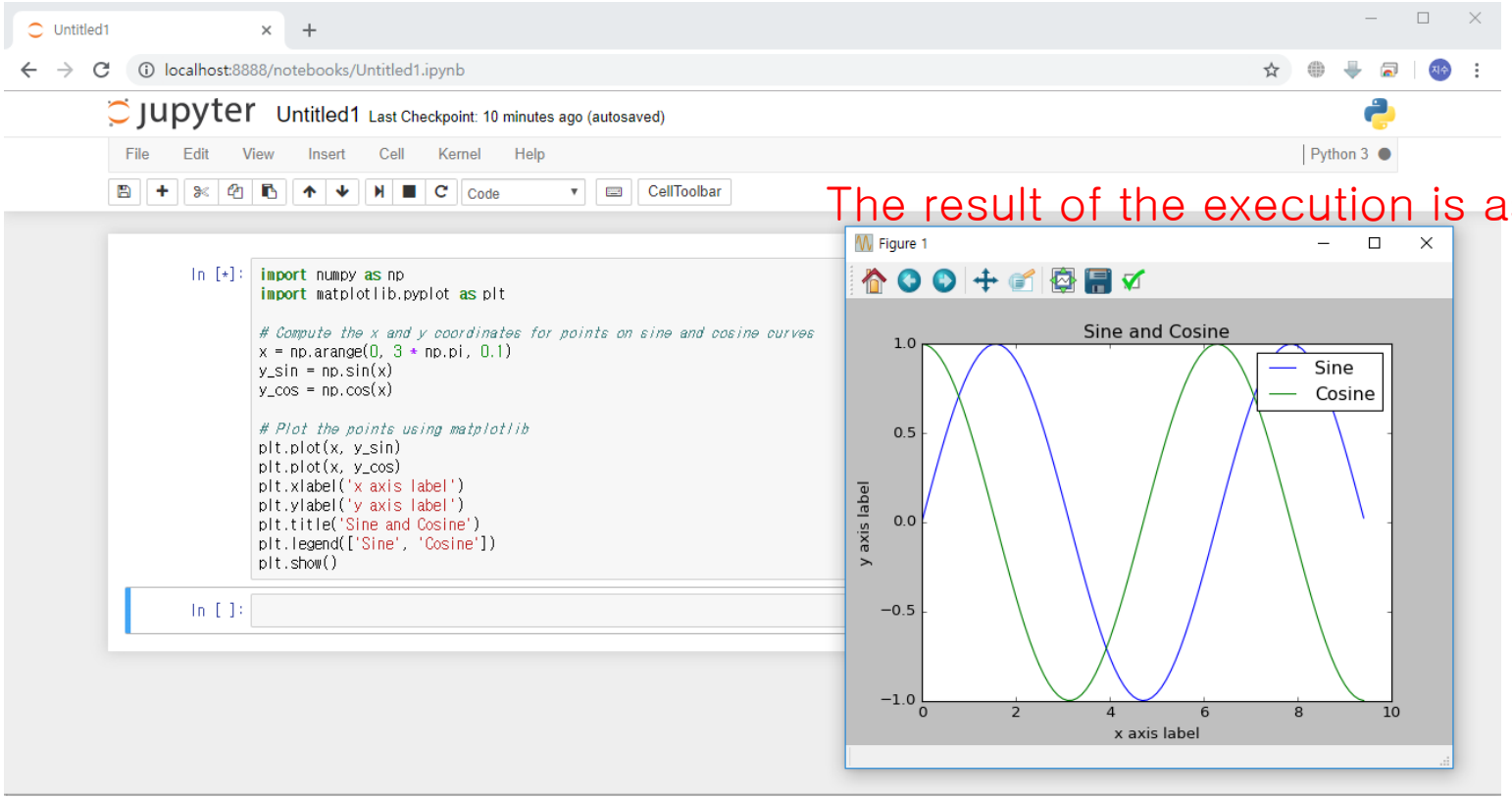
How to use Jupyter notebook

- **Jupyter notebook**



How to use Jupyter notebook

- **Jupyter notebook**

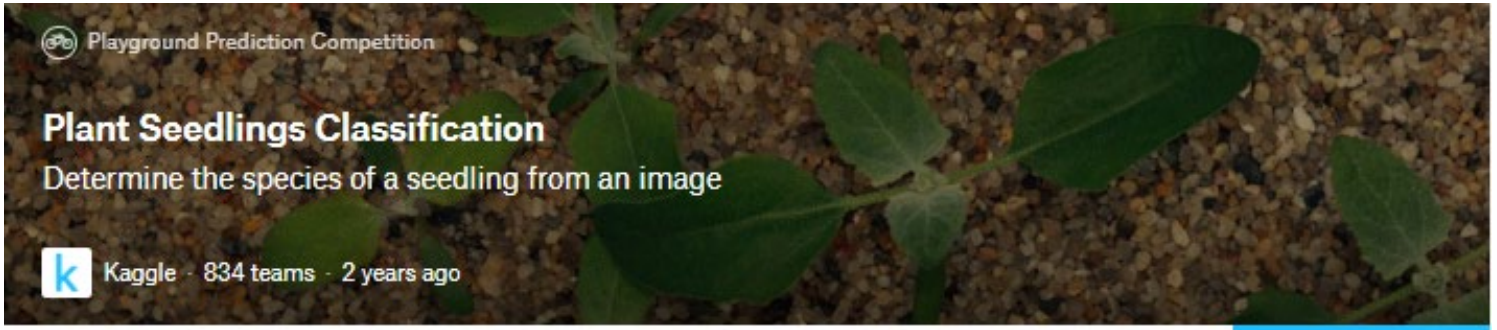


The result of the execution is as follows:

Experiment Environment

- Python v3.52 (Latest version also available)
- TensorFlow v1.8.0 (Latest version also available)
- Jupyter notebook

Data Introduction



Overview Data Notebooks Discussion Leaderboard Rules Team My Submissions Late Submission

Overview

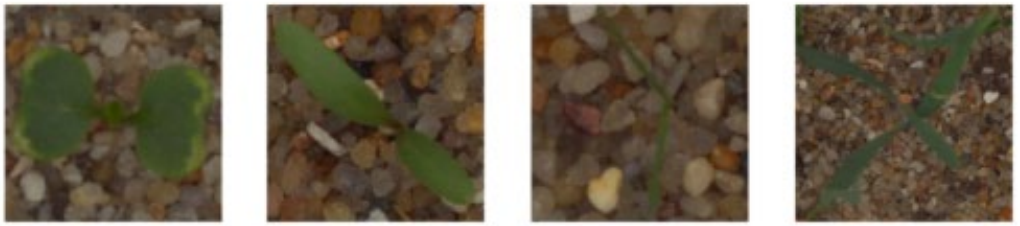
Description

Evaluation

Can you differentiate a weed from a crop seedling?

The ability to do so effectively can mean better crop yields and better stewardship of the environment.

The Aarhus University Signal Processing group, in collaboration with University of Southern Denmark, has recently released a dataset containing images of approximately 960 unique plants belonging to 12 species at several growth stages.



We're hosting this dataset as a Kaggle competition in order to give it wider exposure, to give the community an opportunity to experiment with different image recognition techniques, as well to provide a place to cross-pollenate ideas.

https://www.kaggle.com/c/plant-seedlings-classification/overview/description

Data Introduction

- **Plant Seedlings DB**
  - Total number of classes : 12
  - Total number of images : 4,750 (train : 3,847, validation : 447, test : 456)
  - Original image size : *99~2670* **x** *99~2840* **x** *3* → Resized image size : *64* **x** *64* **x** *3*
  - ***Detailed class information*** **and** ***number of images***
    - Black-grass : 263
    - Charlock : 390
    - Cleavers : 287
    - Common Chickweed : 611
    - Common wheat : 221
    - Fat Hen : 475
    - Loose Silky-bent : 654
    - Maize : 221
    - Scentless Mayweed : 516
    - Shepherds Purse : 231
    - Small-flowered Cranesbill : 496
    - Sugar beet : 385


Maize


Common wheat


Sugar beet


Scentless Mayweed

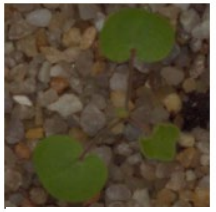

Chickweed


Shepherds Purse


Cleavers


Charlock


Fat Hen


Cranesbill


Black-grass


Loose Silky-bent

Data preprocessing

```
1  import sys
2  import os
3  import cv2
4  import time
5  import glob
6  import random
7  import numpy as np
8  import tensorflow as tf
9  import matplotlib.pyplot as plt
10
11 %matplotlib inline
12
13 train_folder = '/home/joohyung/Documents/plant-seedlings/train'
14 val_folder = '/home/joohyung/Documents/plant-seedlings/validation'
15 test_folder = '/home/joohyung/Documents/plant-seedlings/test'
16
17 def _minmax_scaler(data):
18     numerator = data - np.min(data, 0)
19     denominator = np.max(data, 0) - np.min(data, 0)
20
21     return numerator / (denominator + 1e-7)
22
23
```

Enter your full path

$$MinMaxNorm = \frac{x - Min}{Max - Min}$$

- **Minmax normalization** is a **normalization** strategy which linearly transforms x to y= (x-**min**)/(**max**-**min**), where **min** and **max** are the minimum and **maximum** values in X, where X is the set of observed values of x. When x=**max**, then y=1. This means, the minimum value in X is mapped to 0 and the **maximum** value in X is mapped to 1.

Data preprocessing

```
24 def _one_hot(idx):
25     idx = tf.one_hot(np.array(idx), 12).eval(session=tf.Session())
26 
27     return idx
28 
29 def _shuffle(images, labels):
30     tmp = [[x,y] for x, y in zip(images, labels)]
31     random.shuffle(tmp)
32     shuffle_images = [n[0] for n in tmp]
33     shuffle_labels = [n[1] for n in tmp]
34 
35     return shuffle_images, shuffle_labels
```

- The reason for using **Function _one_hot(idx)** is because of the cost function(*softmax-cross-entropy loss*).

- When importing data, the same kind of class is accumulated because they are loaded in order of folder name. This will cause the same class to exist in the mini batch and will not train well. For this reason, **the Function _shuffle(images, labels) should be used to mix different classes of mini batches.**

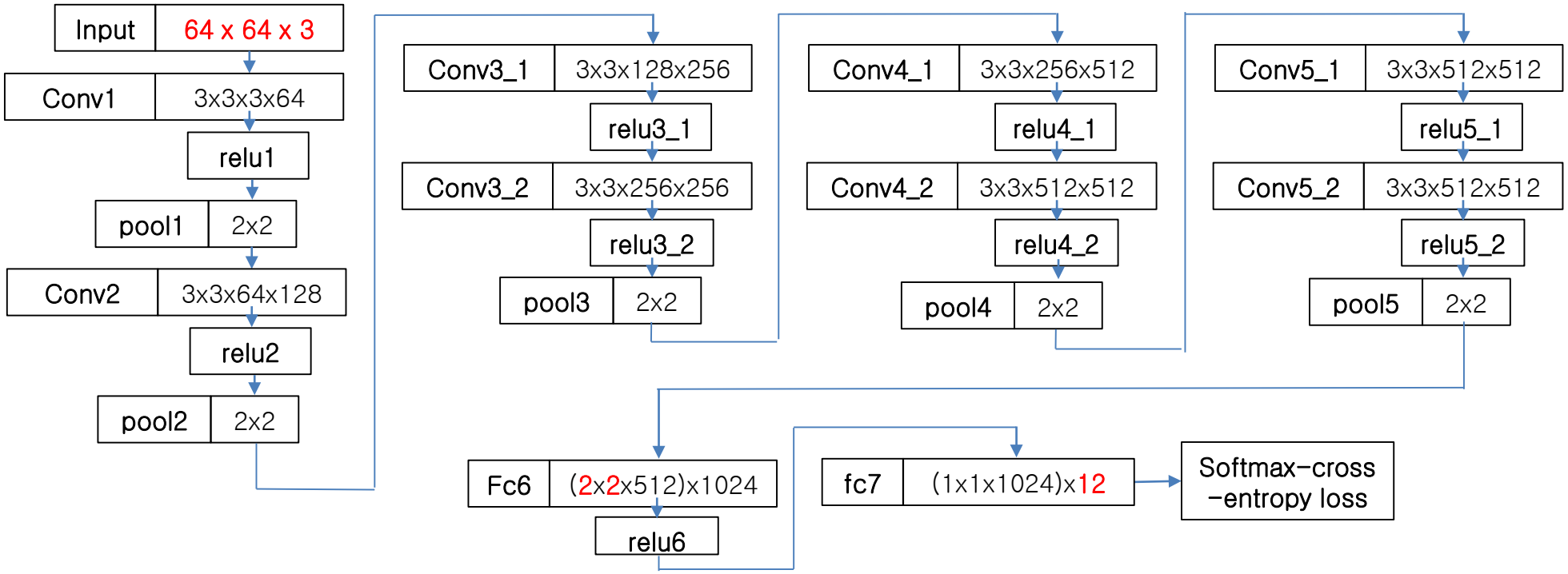
Data preprocessing

```
36
37  def _data_load(folder_path):
38      images = []
39      labels = []
40      folder_list = os.listdir(folder_path)
41      for i, v in enumerate(folder_list):
42          image_path = glob.glob(os.path.join(folder_path,v,'*.png'))
43          for j in image_path:
44              images.append(cv2.imread(j))
45              labels.append(i)
46
47      return _minmax_scaler(images), _one_hot(labels)
48
49
50  train_images, train_labels = _data_load(train_folder)
51  val_images, val_labels = _data_load(val_folder)
52  test_images, test_labels = _data_load(test_folder)
53
54  print("train_images shape: ", np.array(train_images).shape, "// type: ", type(np.array(train_images)))
55  print("train_labels shape: ", np.array(train_labels).shape, "// type: ", type(np.array(train_labels)))
56  print("validation_images shape: ", np.array(val_images).shape, "// type: ", type(np.array(val_images)))
57  print("validation_labels shape: ", np.array(val_labels).shape, "// type: ", type(np.array(val_labels)))
58  print("test_images shape: ", np.array(test_images).shape, "// type: ", type(np.array(test_images)))
59  print("test_labels shape: ", np.array(test_labels).shape, "// type: ", type(np.array(test_labels)))
```

<output>

```
train_images shape:  (3847, 64, 64, 3) // type:  <class 'numpy.ndarray'>
train_labels shape:  (3847, 12) // type:  <class 'numpy.ndarray'>
validation_images shape:  (474, 64, 64, 3) // type:  <class 'numpy.ndarray'>
validation_labels shape:  (474, 12) // type:  <class 'numpy.ndarray'>
test_images shape:  (429, 64, 64, 3) // type:  <class 'numpy.ndarray'>
test_labels shape:  (429, 12) // type:  <class 'numpy.ndarray'>
```

Network Structure



- **Number of layer by type**
  - conv. Layer : 8
  - pooling layer : 5
  - fully-connected layer : 2
- **Cost(loss) function**
  - Softmax-cross-entropy loss
- **Optimaizer**
  - Adam or SGD+momentum

Network design 1

```
learning_rate = 0.0001
batch_size = 64
n_epochs = 100

# Create placeholders
X = tf.placeholder(tf.float32,shape=[None,64,64,3], name="image")
Y = tf.placeholder(tf.int32,[None, 12], name="label")
#Y = tf.one_hot(Y, 12).eval(session=tf.Session())
#Y1 = tf.one_hot(Y, 12)
#Y2 = tf.reshape(Y1, [-1, 12])

Kernel1 = tf.get_variable("Kernel1",shape=[3,3,3,64],initializer=tf.contrib.layers.xavier_initializer())
Bias1 = tf.Variable(tf.truncated_normal(shape=[64],stddev=0.1))
Conv1 = tf.nn.conv2d(X, Kernel1, strides=[1,1,1,1], padding='SAME')+Bias1
Activation1 = tf.nn.relu(Conv1)

Pool1 = tf.nn.max_pool(Activation1, ksize=[1,2,2,1],strides=[1,2,2,1],padding='SAME')
```

- **Parameter setting**
  - Learning rate = 0.0001
  - Batch size = 64 (If your computer's performance is good, you can increase its batch size.)
  - epochs = 100

- **tf.placeholder** : A placeholder is simply a variable that we will assign data to at a later date. It allows us to create our operations and build our computation graph, without needing the data.

Network design 2

```
19 Kernel2 = tf.get_variable("Kernel2",shape=[3,3,64,128],initializer=tf.contrib.layers.xavier_initializer())
20 Bias2 = tf.Variable(tf.truncated_normal(shape=[128],stddev=0.1))
21 Conv2 = tf.nn.conv2d(Pool1, Kernel2, strides=[1,1,1,1], padding='SAME')+Bias2
22 Activation2 = tf.nn.relu(Conv2)
23
24 Pool2 = tf.nn.max_pool(Activation2, ksize=[1,2,2,1],strides=[1,2,2,1],padding='SAME')
25
26 Kernel3_1 = tf.get_variable("Kernel3_1",shape=[3,3,128,256],initializer=tf.contrib.layers.xavier_initializer())
27 Bias3_1 = tf.Variable(tf.truncated_normal(shape=[256],stddev=0.1))
28 Conv3_1 = tf.nn.conv2d(Pool2, Kernel3_1, strides=[1,1,1,1], padding='SAME')+Bias3_1
29 Activation3_1 = tf.nn.relu(Conv3_1)
30
31 Kernel3_2 = tf.get_variable("Kernel3_2",shape=[3,3,256,256],initializer=tf.contrib.layers.xavier_initializer())
32 Bias3_2 = tf.Variable(tf.truncated_normal(shape=[256],stddev=0.1))
33 Conv3_2 = tf.nn.conv2d(Activation3_1, Kernel3_2, strides=[1,1,1,1], padding='SAME')+Bias3_2
34 Activation3_2 = tf.nn.relu(Conv3_2)
35
36 Pool3 = tf.nn.max_pool(Activation3_2, ksize=[1,2,2,1],strides=[1,2,2,1],padding='SAME')
37
38 Kernel4_1 = tf.get_variable("Kernel4_1",shape=[3,3,256,512],initializer=tf.contrib.layers.xavier_initializer())
39 Bias4_1 = tf.Variable(tf.truncated_normal(shape=[512],stddev=0.1))
40 Conv4_1 = tf.nn.conv2d(Pool3, Kernel4_1, strides=[1,1,1,1], padding='SAME')+Bias4_1
41 Activation4_1 = tf.nn.relu(Conv4_1)
42
43 Kernel4_2 = tf.get_variable("Kernel4_2",shape=[3,3,512,512],initializer=tf.contrib.layers.xavier_initializer())
44 Bias4_2 = tf.Variable(tf.truncated_normal(shape=[512],stddev=0.1))
45 Conv4_2 = tf.nn.conv2d(Activation4_1, Kernel4_2, strides=[1,1,1,1], padding='SAME')+Bias4_2
46 Activation4_2 = tf.nn.relu(Conv4_2)
47
48 Pool4 = tf.nn.max_pool(Activation4_2, ksize=[1,2,2,1],strides=[1,2,2,1],padding='SAME')
49
```

Network design 3

```
50 Kernel5_1 = tf.get_variable("Kernel5_1",shape=[3,3,512,512],initializer=tf.contrib.layers.xavier_initializer())
51 Bias5_1 = tf.Variable(tf.truncated_normal(shape=[512],stddev=0.1))
52 Conv5_1 = tf.nn.conv2d(Pool4, Kernel5_1, strides=[1,1,1,1], padding='SAME')+Bias5_1
53 Activation5_1 = tf.nn.relu(Conv5_1)
54
55 Kernel5_2 = tf.get_variable("Kernel5_2",shape=[3,3,512,512],initializer=tf.contrib.layers.xavier_initializer())
56 Bias5_2 = tf.Variable(tf.truncated_normal(shape=[512],stddev=0.1))
57 Conv5_2 = tf.nn.conv2d(Activation5_1, Kernel5_2, strides=[1,1,1,1], padding='SAME')+Bias5_2
58 Activation5_2 = tf.nn.relu(Conv5_2)
59
60 Pool5 = tf.nn.max_pool(Activation5_2, ksize=[1,2,2,1],strides=[1,2,2,1],padding='SAME')
61
62 W1 = tf.get_variable("W1",shape=[2*2*512, 512],initializer=tf.contrib.layers.xavier_initializer())
63 B1 = tf.Variable(tf.truncated_normal(shape=[512]))
64 Pool5_flat = tf.reshape(Pool5,[-1,2*2*512])
65 fc6 = tf.matmul(Pool5_flat,W1)+B1
66 Activation6 = tf.nn.relu(fc6)
67
68 W2 = tf.get_variable("W2",shape=[512, 12],initializer=tf.contrib.layers.xavier_initializer())
69 B2 = tf.Variable(tf.truncated_normal(shape=[12]))
70 OutputLayer = tf.matmul(Activation6 ,W2)+B2
71
```

Network design 4

Use softmax-cross-entropy loss

```
72 loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=OutputLayer, labels=Y))
73 optimizer = tf. train.MomentumOptimizer(learning_rate=learning_rate,momentum=0.9).minimize(loss)
74 #optimizer = tf. train.AdamOptimizer(learning_rate=learning_rate).minimize(loss)
75
76 correct_prediction = tf.equal(tf.argmax(OutputLayer,1),tf.argmax(Y,1))
77 accuracy = tf.reduce_mean(tf.cast(correct_prediction,tf.float32))
```

<SGD+momentum>
If you want to use SGD, change the learning rate to 0.01

OR

```
72 loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=OutputLayer, labels=Y))
73 #optimizer = tf. train.MomentumOptimizer(learning_rate=learning_rate,momentum=0.9).minimize(loss)
74 optimizer = tf. train.AdamOptimizer(learning_rate=learning_rate).minimize(loss)
75
76 correct_prediction = tf.equal(tf.argmax(OutputLayer,1),tf.argmax(Y,1))
77 accuracy = tf.reduce_mean(tf.cast(correct_prediction,tf.float32))
```

<Adam> *recommend*
If you want to use Adam, change the learning rate to 0.00001

- **Adam optimizer** is more stable than **SGD+momentum optimizer** in training.

momentum source code : https://github.com/tensorflow/tensorflow/blob/r2.0/tensorflow/python/training/momentum.py#L29-L133
Adam source code : https://github.com/tensorflow/tensorflow/blob/r2.0/tensorflow/python/training/adam.py#L32-L235

Main code - train

```
loss_train_history = []
acc_train_history = []
loss_val_history = []
acc_val_history = []

init = tf.global_variables_initializer()

with tf.Session() as sess:
    sess.run(init)
    n_batches = int(len(train_images)/batch_size)

    for i in range(n_epochs):
        total_correct_train_preds = 0
        x_train, y_train = shuffle(train_images, train_labels)
        for k in range(n_batches):
            X_train_batch, Y_train_batch = x_train[k*batch_size:k*batch_size+batch_size], \
                                           y_train[k*batch_size:k*batch_size+batch_size]
            _, loss_train, OutputLayer_train_batch = sess.run([optimizer, loss, accuracy], \
                                                              feed_dict={X: X_train_batch, Y: Y_train_batch})

            total_correct_train_preds += OutputLayer_train_batch
        train_accuracy = total_correct_train_preds/n_batches

```

Find the total number of batches

Shuffle the train data.

Extract batch about image and label.

If 'optimizer' is written, it is training.
Otherwise, it is validation or testing.

- Performing a computation in TensorFlow is slightly different from doing the same computation in plain python. One first needs to define the **structure ("graph")** of the computation, then start a TensorFlow **environment ("session")** for the graph, and finally execute the graph in the context of the session. We define the input parameters**(“feed_dict”)** and their associated data types.

Main code – validation and test

```
25          # Check validation accuracy
26          n_v_batches = int(len(val_images)/batch_size)
27          total_correct_val_preds = 0
28          for j in range(n_v_batches):
29              X_val_batch, Y_val_batch = val_images[j*batch_size:j*batch_size+batch_size], \
30                                         val_labels[j*batch_size:j*batch_size+batch_size]
31              loss_val, OutputLayer_val_batch = sess.run([loss, accuracy], feed_dict={X: X_val_batch, \
32                                                                                      Y: Y_val_batch})
33              total_correct_val_preds += OutputLayer_val_batch
34          validation_accuracy = total_correct_val_preds/n_v_batches
35
36          print('epoch:%d // train loss:%.3f // train accuracy:%.3f // val loss:%.3f // val accuracy:%.3f' \
37                % (i, loss_train, train_accuracy, loss_val, validation_accuracy))
38          loss_train_history.append(loss_train)
39          acc_train_history.append(train_accuracy)
40          loss_val_history.append(loss_val)
41          acc_val_history.append(validation_accuracy)
42
43
44
45
46      # Test the model
47      n_batches = int(len(test_images)/batch_size)
48      total_correct_test_preds = 0
49      for i in range(n_batches):
50          X_test_batch, Y_test_batch = test_images[i*batch_size:i*batch_size+batch_size], \
51                                       test_labels[i*batch_size:i*batch_size+batch_size]
52          OutputLayer_test_batch = sess.run(OutputLayer, feed_dict={X: X_test_batch, Y:Y_test_batch})
53          preds = tf.nn.softmax(OutputLayer_test_batch)
54          correct_preds = tf.equal(tf.argmax(preds, 1), tf.argmax(Y_test_batch, 1))
55          accuracy = tf.reduce_sum(tf.cast(correct_preds, tf.float32))
56          total_correct_test_preds += sess.run(accuracy)
57
58      print ("Test accuracy is {0}".format(total_correct_test_preds/len(test_images)))
```

If 'optimizer' is used it is training, otherwise it is validation or testing

Save the loss and accuracy value to draw a graph

Main code execution result

<Output>

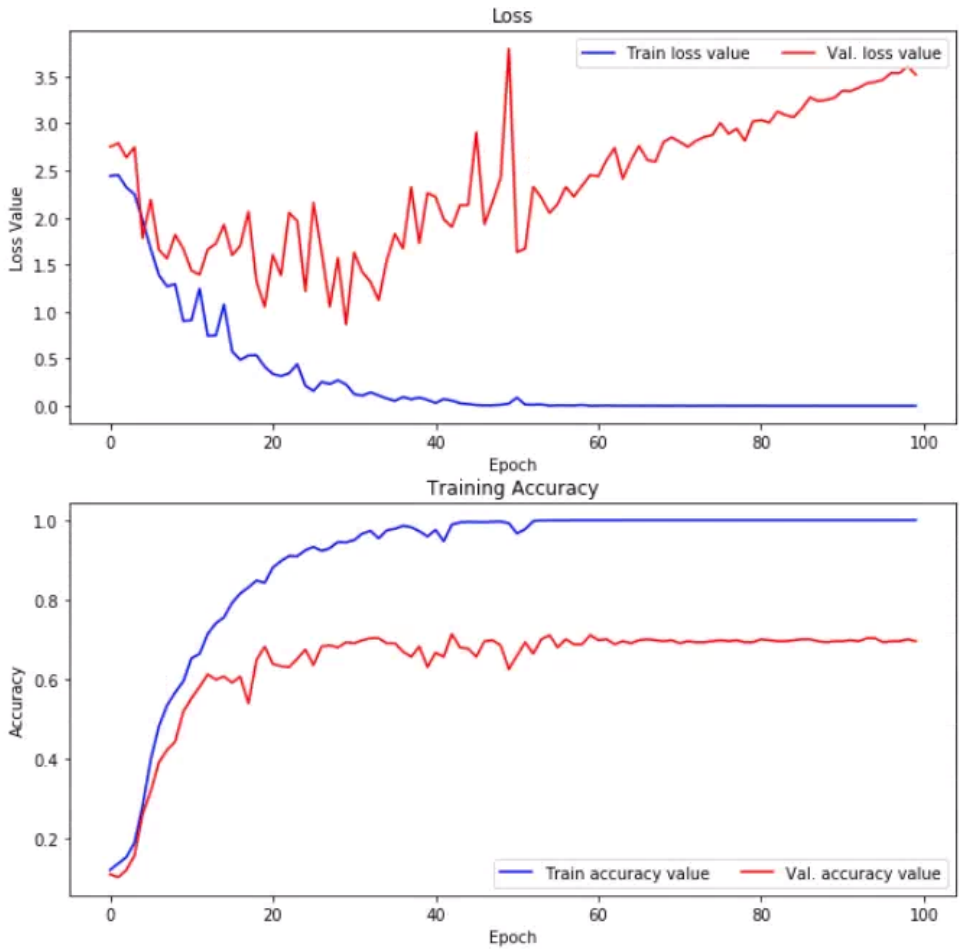
```
epoch:0 // train loss:2.363 // train accuracy:0.131 // val loss:2.767 // val accuracy:0.109
epoch:1 // train loss:2.424 // train accuracy:0.145 // val loss:2.560 // val accuracy:0.154
epoch:2 // train loss:1.978 // train accuracy:0.201 // val loss:2.538 // val accuracy:0.161
epoch:3 // train loss:1.653 // train accuracy:0.368 // val loss:2.163 // val accuracy:0.268
epoch:4 // train loss:1.339 // train accuracy:0.454 // val loss:2.124 // val accuracy:0.383
epoch:5 // train loss:1.262 // train accuracy:0.525 // val loss:1.817 // val accuracy:0.445
epoch:6 // train loss:1.185 // train accuracy:0.576 // val loss:2.063 // val accuracy:0.414
epoch:7 // train loss:1.177 // train accuracy:0.635 // val loss:1.526 // val accuracy:0.542
epoch:8 // train loss:0.736 // train accuracy:0.680 // val loss:1.495 // val accuracy:0.521
epoch:9 // train loss:0.951 // train accuracy:0.709 // val loss:1.484 // val accuracy:0.599
epoch:10 // train loss:0.701 // train accuracy:0.769 // val loss:1.371 // val accuracy:0.583
epoch:11 // train loss:0.683 // train accuracy:0.784 // val loss:1.150 // val accuracy:0.669
epoch:12 // train loss:0.552 // train accuracy:0.827 // val loss:1.200 // val accuracy:0.656
epoch:13 // train loss:0.470 // train accuracy:0.829 // val loss:1.106 // val accuracy:0.656
                                                    ⋮
epoch:89 // train loss:0.000 // train accuracy:1.000 // val loss:4.094 // val accuracy:0.729
epoch:90 // train loss:0.000 // train accuracy:1.000 // val loss:4.220 // val accuracy:0.737
epoch:91 // train loss:0.000 // train accuracy:1.000 // val loss:4.188 // val accuracy:0.732
epoch:92 // train loss:0.000 // train accuracy:1.000 // val loss:4.318 // val accuracy:0.740
epoch:93 // train loss:0.000 // train accuracy:1.000 // val loss:4.328 // val accuracy:0.742
epoch:94 // train loss:0.000 // train accuracy:1.000 // val loss:4.344 // val accuracy:0.724
epoch:95 // train loss:0.000 // train accuracy:1.000 // val loss:4.425 // val accuracy:0.729
epoch:96 // train loss:0.000 // train accuracy:1.000 // val loss:4.466 // val accuracy:0.724
epoch:97 // train loss:0.000 // train accuracy:1.000 // val loss:4.528 // val accuracy:0.727
epoch:98 // train loss:0.000 // train accuracy:1.000 // val loss:4.529 // val accuracy:0.727
epoch:99 // train loss:0.000 // train accuracy:1.000 // val loss:4.590 // val accuracy:0.734
Test accuracy is 0.7390350877192983
```

Train accuracy : 100%, validation accuracy : 73.4%, Test accuracy : 73.9%

Draw a graph for loss and accuracy

```
plt.subplot(2,1,1)
plt.plot(loss_train_history, 'b-', label='Train loss value')
plt.plot(loss_val_history, 'r-', label='Val. loss value')
plt.title('Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss Value')
plt.legend(ncol=2, loc='upper right')

plt.subplot(2,1,2)
plt.plot(acc_train_history, 'b-', label='Train accuracy value')
plt.plot(acc_val_history, 'r-', label='Val. accuracy value')
plt.title('Training Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(ncol=2, loc='lower right')

plt.gcf().set_size_inches(10, 10)
plt.show()
```

<Output>



If an error occurs, proceed as follows: