Fruits Classification with Tensorflow and Keras

Nguyen Thi Hoai Thuong

March 2020

1 Introduction

In this paper we will detailed step to prepare dataset by myself about fruits - 4 labels (orange, mango, apple and banana bunch), use CNN architecture to train model using Tensorflow and Keras. We can compare two API for result.
Let's start

2 Deep learning

Deep learning is a subfield of Machine Learning that use multiple layers that contain nonlinear processing units. Each level learns to transform its input data into a slightly more abstract and composite representation.
In this area of image classification we use Deep learning for successful result.

2.1 Convolution Neural Network

Convolution Neural Networks (CNNs) is one of the important part of Deep learning. It consists of: convolution layers, pooling layers, ReLu layers, fully connector layer,...

In a typical CNN architecture, each convolution layer is followed by a Rectified Linear Unit (ReLU) layer, then a Pooling layer then one or more convolution layer and finally one or more fully connected layer. A characteristic that sets apart the CNN from a regular neural network is taking into account the structure.

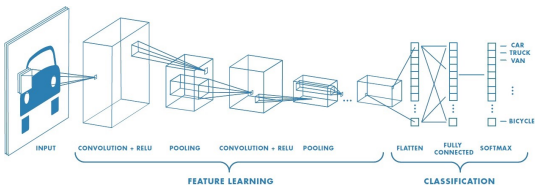

Figure 1: Convolution Neuron Networks (CNN) [1]

2.1.1 Convolution layer

Convolution is the first layer to extract features from an input image. Convolution preserves the relationship between pixels by learning image features using small squares of input data. It is a mathematical operation that takes two inputs such as image matrix and a filter or kernel.

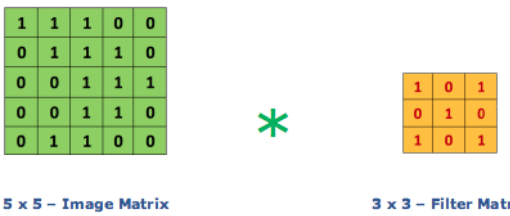

Figure 2: Image matrix multiples with kernel to get extract features [1]

Activation Function
Following linear layers, whether convolutional or fully connected,it is common practice to apply nonlinear

activation functions (see Figure 3 for some examples). One practical aspect of activation functions is that consecutive linear operations can be replaced by a single one, and thus depth does not contribute to the expressiveness of the model unless we use nonlinear activations between the linear layers.

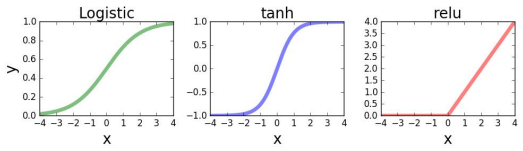

Figure 3: Activation functions [1]

2.1.2 Pooling layer

Pooling layers section would reduce the number of parameters when the images are too large. Spatial pooling also called sub sampling or down sampling which reduces the dimensionality of each map but retains important information. Spatial pooling can be of different types:

- Max Pooling
- Average Pooling
- Sum Pooling

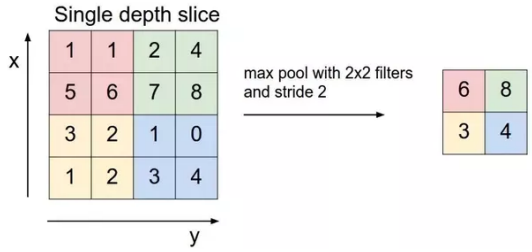

Figure 4: Example Max Pooling

2.1.3 Fully Connector Layer

Fully Connector Layer (FC), we flattened our matrix into vector and feed it into a fully connected layer like a neural network.

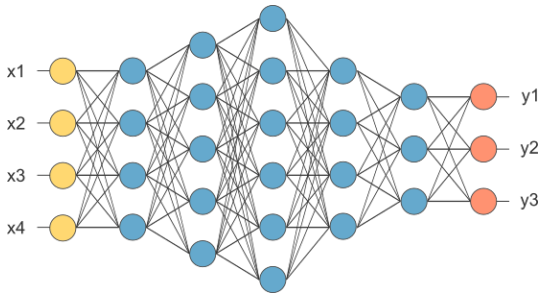

Figure 5: After Max Pooling flatten by FC

2.1.4 Loss Layer

For object classification, if the label is 2, we will use loss='binary crossentropy'. If it is more than 2, use loss = 'sparse categorical crossentropy'

3 Tensorflow Library

TensorFlow™ is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The flexible architecture allows you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API.
TensorFlow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well.
It's very helpful for me to train core.
First, we should define variables for all layer.
This specifies the weights for either fully connected or convolutional layers of the network.

```
def weight_variable(shape):
    initial = tf.random.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)
```

This defines the bias elements in either a fully connected or a convolutional layer. These are all initialized with the constant value of .1 .

```
def bias_variable(shape):
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)
```

A convolution layer define like this:

```
def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1],          padding='SAME')
```

The input is x-image input. Here, this filter is parameterized by W , the learned weights of our network representing the convolution filter.
Performs the max pooling operation on the input. The ksize and strides parameters can be tuples or lists of tuples of 4 elements. Ksize represents the size of the window for each dimension of the input tensor and strides represents the stride of the sliding window for each dimension of the input tensor. The padding parameter can be ''VALID'' or ''SAME'', here we use SAME.

```
def max_pool_2x2(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                          strides=[1, 2, 2, 1], padding='SAME')
```

This is the actual layer we will use. Linear convolution as defined in conv2d , with a bias, followed by the ReLU nonlinearity.

```
def conv_layer(input, shape):
    W = weight_variable(shape)
    b = bias_variable([shape[3]])
    return tf.nn.relu(conv2d(input, W) + b)
```

This sets the max pool to half the size across the height/width dimensions, and in total a quarter the size of the feature map.

```
def max_pool_2x2(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                          strides=[1, 2, 2, 1], padding='SAME')
```

This is the actual layer we will use. Linear convolution as defined in conv2d , with a bias, followed by the ReLU nonlinearity.

```
def conv_layer(input, shape):
    W = weight_variable(shape)
    b = bias_variable([shape[3]])
    return tf.nn.relu(conv2d(input, W) + b)
```

A standard full layer with a bias. Notice that here we didn't add the ReLU. This allows us to use the same layer for the final output, where we don't need the non-linear part.

```
def full_layer(input, size):
    in_size = int(input.get_shape()[1])
    W = weight_variable([in_size, size])
    b = bias_variable([size])
    return tf.matmul(input, W) + b
```

4 Keras Library

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.
We designed a project is coded by Tensorflow and this we used Keras library
Same Tensorflow, we define layers: conv2d, max pooling, fully connector.
Conv2d

```
keras.layers.Conv2D(filters, kernel_size)
```

Max pooling

```
keras.layers.MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid')
```

Activation Function We used activation function 'relu' and 'soft max' for convolution and full connected layer.

```
keras.layers.Conv2D(32,(3,3), activation='relu',input_shape=(50, 50, 3)),
```

Loss

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

5 Create our dataset and preprocessing image

5.1 Create dataset

Dataset is list of images pictured object need to be trained. We prepare dataset by capture video of the object and get frame from video list into folder name of this object.
On my dataset, we have 4 object data: orange, apple, mango and banana bunch. It's about 4000 images. This dataset we will public on github dataset.

5.2 Process Dataset

Using OpenCV library to processing image, read and resize all dataset. Here we resized all image to 50x50x3 (3 is channels Red, Green and Blue)

```
img_array = cv2.imread(os.path.join(path,img),cv2.COLOR_BGR2RGB)  #convert to array
new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
 #resize to normalize data size
```

Define a function will create a list of image and lalels (labels after encoding (Tensorflow))

```
def get_data(data):
    x= []
    y= []
    for features,label in data:
        x.append(features) #images
        y.append(label)  #labels
    y = encode(y)
    return x,y
```

Define function to encode labels to onehot type for Tensorflow

```
def encode(data):
    encoded = tf.keras.utils.to_categorical(data)
    return encoded
```

Beside, we should split data to train and test. Here, we use 0.3 data for testing and remain for traning.

```
(x_train, x_test, y_train, y_test) = train_test_split(x,y,test_size=0.25, random_state
```

Save data after create in fie *.pickle. It's wil help you for getting data easily in the future without read and load all data.

```
import pickle
#save data : images, labels one hot
pickle_out = open("X.pickle","wb")
pickle.dump(x_train, pickle_out)
pickle_out.close()

pickle_out = open("y.pickle","wb")
pickle.dump(y_train, pickle_out)
pickle_out.close()
```

6 Structure of the neural networks

6.1 Tensorflow CNNs

First, when using Tensorflow, we need array input, so convert x (list of images) and y (labels) to arrays using Numpy library.

```
x_train = np.array(x_train)
x_test = np.array(x_test)
y_train = np.array(y_train)
y_test = np.array(y_test)
```

We use function below to create a graph where feed data in.

```
x = tf.placeholder(tf.float32, shape=[None,50,50,3], name = "Input")
y_ = tf.placeholder(tf.float32, shape=[None,4],name = "Output")
x_image = tf.reshape(x,[-1,50,50,3])
```

Using layers are defined connects together, we have an CNNs

Table 1: Structure of network layers used Tensorflow

Layer Type	Dimension	Out put
Convolutional	5x5x3	64
Max Pooling	2x2, Stride 2x2	-
Convolutional	5x5x64	256
Max Pooling	2x2, Stride 2x2	-
Fully Connector	5x5x256	1024
Softmax	1024	4

```
keep_prob = tf.placeholder(tf.float32)
conv1 = conv_layer(x_image, shape=[5, 5, 3, 32])
conv1_pool = max_pool_2x2(conv1)

conv2 = conv_layer(conv1_pool, shape=[5, 5, 32, 64])
conv2_pool = max_pool_2x2(conv2)

conv3 = conv_layer(conv2_pool, shape=[5, 5, 64, 128])
conv3_pool = max_pool_2x2(conv3)

conv3_flat = tf.reshape(conv3_pool, [-1, 7* 7 * 128])
conv3_drop = tf.nn.dropout(conv3_flat, keep_prob=keep_prob)

full_1 = tf.nn.relu(full_layer(conv3_drop, 512))
full1_drop = tf.nn.dropout(full_1, keep_prob=keep_prob)
y_conv = full_layer(full1_drop, 4)
```

6.2 Keras CNNs

We have CNN below:

```
model = keras.Sequential(
    [
    keras.layers.Conv2D(16,(5,5), activation='relu',input_shape=(50, 50, 3)),
    keras.layers.MaxPooling2D((2,2)),
    keras.layers.Conv2D(32, (5, 5), activation='relu'),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Conv2D(64,(5,5), activation='relu'),
    keras.layers.MaxPooling2D((2,2)),
    keras.layers.Flatten(),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(4, activation='softmax')
    ])
```

7 Results

After create model with Tensorflow and Keras we have results of training accuracy and test accuracy in table 2 below. We public our on github [2].
The accuracy we get in different structure network and numbers of step.
We train model in GPU device Gerfoce GTX 1080 Ti, using Tensorflow GPU version 1.12, Keras Processing 1.1.0 and Opencv 4.2.0. We run model in python IDE PyCharm.

Table 2: Result of structure

Number	Layer Structure	Accuracy Train	Accuracy Test
1. Keras Session 5 epoch	Conv1 5x5x32 Max Pooling 2x2 Conv1 5x5x64 Max Pooling 2x2 Conv1 5x5x128 Max Pooling 2x2 Full Connector 256	100%	100%
2. Keras 6 epoch	Conv1 5x5x32 Max Pooling 2x2 Conv1 5x5x64 Max Pooling 2x2 Conv1 5x5x128 Max Pooling 2x2 Full Connector 256	100%	99.90%
3. Keras 5 epoch	Conv1 5x5x16 Max Pooling 2x2 Conv1 5x5x32 Max Pooling 2x2 Conv1 5x5x64 Max Pooling 2x2 Full Connector 128	99.97%	100%
4. Tensorflow Step: 800	Conv1 5x5x32 Max Pooling 2x2 Conv1 5x5x64 Max Pooling 2x2 Conv1 5x5x128 Max Pooling 2x2 Full Connector 512	100%	99.09%
5. Tensorflow Step: 900	Conv1 5x5x32 Max Pooling 2x2 Conv1 5x5x64 Max Pooling 2x2 Conv1 5x5x128 Max Pooling 2x2 Full Connector 512	100%	97.73%
6. Tensorflow Step: 1000	Conv1 5x5x32 Max Pooling 2x2 Conv1 5x5x64 Max Pooling 2x2 Conv1 5x5x128 Max Pooling 2x2 Full Connector 512	100%	99.72%

References

[1]https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148
[2]https://github.com/ivsr/Computer-Visison/tree/master/fruit_classification