# playground

January 18, 2019

In [1]:
```
import keras
from keras.models import load_model
from keras.datasets import cifar10
from keras import backend as K
import numpy as np

# Load the dataset that could be used to test for model performance
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
num_classes = 10
y_test = keras.utils.to_categorical(y_test, num_classes)


# Interrupt the kernel after evaluate the model
```

```
C:\Users\wy_cl\AppData\Local\Continuum\anaconda3\lib\site-packages\h5py\__init__.py:36: FutureW
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

The model's accuracy is as below. It is not necessay to run this section because this would just serve as a baseline. Have to restart kernel after running the evaluation to clear GPU memeory usage.

In [4]:
```
# Load the keras model files that could be used to test the functions in later section
model_path = "test_model_files/cifar10_model.h5"
model = load_model(model_path)
model.evaluate(x=x_test, y=y_test)
```

```
10000/10000 [==============================] - 1s 96us/step
```

Out[4]:
```
[4.972338047027588, 0.686]
```

### 0.0.1 load model weights from a keras model

In this function, I am able to take a keras model and read the each layer's weight matrix and configuration. Then I save the each layer's weight matrix in one list so that I could modify them later.

```
In [2]: def load_model_weights(model):
            # Create two variables to store the layer information and weight matrixes.
            layers_name = []
            weights = []

            # Loop through the model layer by layer
            for layer in model.layers:
                # extract the layer information by using layer.get_config()
                layers_name.append(layer.get_config())

                # extract the weight matrix for this layer using layer.get_weights()
                weights.append(layer.get_weights())
            return weights, layers_name
```

### 0.0.2 save the modified model weights into the keras model

```
In [3]: def save_model_weights(model, new_weights):
            # Loop through the list containing each layer's new weight
            for i in range(len(new_weights)):
                # set the weight for each layer using set_weight()
                model.layers[i].set_weights(new_weights[i])
            return model
```

### 0.0.3 binarization

```
In [31]: # map the weights to -1 or 1
         def post_train_binarization(weight_data):
             # Loop through each layer
             for i in range(len(weight_data)):
                 # if the layer has weight to be modified:
                 if weight_data[i]:
                     # loop through all weights in the layer
                     for j in range(len(weight_data[i])):
                         # if the number is greater than 0, map the number to 1, and vice vers
                         weight_data[i][j] = np.where(weight_data[i][j] > 0, 1, -1).astype(np.
             return weight_data
```

### 0.0.4 quantization

```
In [ ]: def post_train_quantization(weight_data):
            # map the 8 bits of precision to 4 bits
            # [7, -8]
```

### 0.0.5 Test for loading and saving weights from files

```
In [6]: weights, model_struct = load_model_weights(model)
```

```
In [6]: model_path = "test_model_files/cifar10_model.h5"
        model = load_model(model_path)
        weights, model_struct = load_model_weights(model)

In [7]: model.summary()
```

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d_1 (Conv2D) | (None, 32, 32, 32) | 896 |
| activation_1 (Activation) | (None, 32, 32, 32) | 0 |
| conv2d_2 (Conv2D) | (None, 32, 32, 32) | 9248 |
| activation_2 (Activation) | (None, 32, 32, 32) | 0 |
| conv2d_3 (Conv2D) | (None, 32, 32, 32) | 9248 |
| activation_3 (Activation) | (None, 32, 32, 32) | 0 |
| conv2d_4 (Conv2D) | (None, 32, 32, 48) | 13872 |
| activation_4 (Activation) | (None, 32, 32, 48) | 0 |
| conv2d_5 (Conv2D) | (None, 32, 32, 48) | 20784 |
| activation_5 (Activation) | (None, 32, 32, 48) | 0 |
| max_pooling2d_1 (MaxPooling2 | (None, 16, 16, 48) | 0 |
| dropout_1 (Dropout) | (None, 16, 16, 48) | 0 |
| conv2d_6 (Conv2D) | (None, 16, 16, 80) | 34640 |
| activation_6 (Activation) | (None, 16, 16, 80) | 0 |
| conv2d_7 (Conv2D) | (None, 16, 16, 80) | 57680 |
| activation_7 (Activation) | (None, 16, 16, 80) | 0 |
| conv2d_8 (Conv2D) | (None, 16, 16, 80) | 57680 |
| activation_8 (Activation) | (None, 16, 16, 80) | 0 |
| conv2d_9 (Conv2D) | (None, 16, 16, 80) | 57680 |
| activation_9 (Activation) | (None, 16, 16, 80) | 0 |

```
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 16, 16, 80)        57680
_________________________________________________________________
activation_10 (Activation)   (None, 16, 16, 80)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 80)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 8, 8, 80)          0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 8, 8, 128)         92288
_________________________________________________________________
activation_11 (Activation)   (None, 8, 8, 128)         0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 8, 8, 128)         147584
_________________________________________________________________
activation_12 (Activation)   (None, 8, 8, 128)         0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 8, 8, 128)         147584
_________________________________________________________________
activation_13 (Activation)   (None, 8, 8, 128)         0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 8, 8, 128)         147584
_________________________________________________________________
activation_14 (Activation)   (None, 8, 8, 128)         0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 8, 8, 128)         147584
_________________________________________________________________
activation_15 (Activation)   (None, 8, 8, 128)         0
_________________________________________________________________
global_max_pooling2d_1 (Glob (None, 128)               0
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 500)               64500
_________________________________________________________________
activation_16 (Activation)   (None, 500)               0
_________________________________________________________________
dropout_4 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                5010
_________________________________________________________________
activation_17 (Activation)   (None, 10)                0
=================================================================
Total params: 1,071,542
Trainable params: 1,071,542
Non-trainable params: 0
_________________________________________________________________
```

```
In [8]: weights[0][0].shape

Out[8]: (3, 3, 3, 32)

In [9]: weights[0][1].shape

Out[9]: (32,)
```

From the above, using the model.summary() I can see each layer's information and the size. To test if the load_model_weights() is working, I check the shape of first layer's weight matrix. The first layer weights contains two weight matrix, with shape of (3, 3, 3, 32) and (32,), which means there are 896 parameters in the first layer and the number agrees to the first row of table from model.summarty()

### 0.0.6 Test for modifing weights and load to model

```
In [32]: # Binarize the weights and load the modified weights to model
         new_weights = post_train_binarization(weights)
         binarized_model = save_model_weights(model, new_weights)

In [33]: model.evaluate(x=x_test, y=y_test)

10000/10000 [==============================] - 1s 89us/step


Out[33]: [14.364446496582032, 0.1088]
```

The binarization and save weights functions works. As the result shows, the accuracy decreases significantly. Here the data type is 8 bit integer, I am still working on how to convert the weights to signed 4 bit integer.

```
In [39]: # Showing a piece of weights from the new weight
         new_weights[0][1]

Out[39]: array([-1, -1, -1, -1,  1, -1, -1,  1,  1, -1,  1, -1,  1,  1, -1, -1,  1,
                -1, -1, -1,  1,  1, -1, -1, -1,  1, -1,  1, -1,  1,  1, -1],
               dtype=int8)
```