Pattarapon Buathong 62070504012 AE

Homework 3

```
[1] import tensorflow as tf
    import matplotlib.pyplot as plt
    import numpy as np

[2] cifar = tf.keras.datasets.cifar10
    (x_train, y_train), (x_test, y_test) = cifar.load_data()

    Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
    170498071/170498071 [==============================] - 14s 0us/step

[3] # Convert 2D to 1D array
    y_train = y_train.reshape(-1,)
    y_test = y_test.reshape(-1,)

[4] # Normalization
    x_train, x_test = x_train / 255.0, x_test / 255.0
```

[1] I imported necessary packages.

[2] Download CIFAR10 dataset and assign it to cifar variable. It already has train and test data separated, so assign those to the x and y variables.

[3] Initially, y is 2D array, I do reshape(-1,) to convert it to a 1D array.

[4] Divide the x by 255.0 for a normalization.

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(filters=32, kernel_size=(3,3), padding='same', activation='relu', input_shape=(32, 32, 3)),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters=32, kernel_size=(3,3), padding='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPool2D((2, 2)),
    tf.keras.layers.Dropout(0.2),

    tf.keras.layers.Conv2D(filters=64, kernel_size=(3,3), padding='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters=64, kernel_size=(3,3), padding='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPool2D((2, 2)),
    tf.keras.layers.Dropout(0.3),

    tf.keras.layers.Conv2D(filters=128, kernel_size=(3,3), padding='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters=128, kernel_size=(3,3), padding='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPool2D((2, 2)),
    tf.keras.layers.Dropout(0.4),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Softmax()
])

loss = tf.keras.losses.SparseCategoricalCrossentropy()

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

tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir='log')
```

This is the Keras CNN model. The input shape is (32, 32, 3). Since I will train in batch, so I have to add batch normalizations. Batch normalization has to be after Conv2D but before MaxPool2D. I train Conv2D → BatchNormalization → Conv2D → Batch Normalization → MaxPool2D → Dropout, and increase the filters size from time to time. I have to deal with the overfitting on training data problem, so I add Dropout layer to my neural network. Then, flatten the layer, add one normal hidden layer, do the batch normalization again, then add the Softmax layer lastly. The loss function is Sparse categorical crossentropy. Optimizer will be AdamOptimizer. Use accuracy metrics for measuring training accuracy.

```
model.summary()

batch_normalization_65 (Bat  (None, 32, 32, 32)       128
chNormalization)

conv2d_75 (Conv2D)          (None, 32, 32, 32)       9248

batch_normalization_66 (Bat  (None, 32, 32, 32)       128
chNormalization)

max_pooling2d_39 (MaxPoolin  (None, 16, 16, 32)       0
g2D)

dropout_31 (Dropout)        (None, 16, 16, 32)       0

conv2d_76 (Conv2D)          (None, 16, 16, 64)       18496

batch_normalization_67 (Bat  (None, 16, 16, 64)       256
chNormalization)

conv2d_77 (Conv2D)          (None, 16, 16, 64)       36928

batch_normalization_68 (Bat  (None, 16, 16, 64)       256
chNormalization)

max_pooling2d_40 (MaxPoolin  (None, 8, 8, 64)         0
g2D)

dropout_32 (Dropout)        (None, 8, 8, 64)         0

conv2d_78 (Conv2D)          (None, 8, 8, 128)        73856

batch_normalization_69 (Bat  (None, 8, 8, 128)        512
chNormalization)

conv2d_79 (Conv2D)          (None, 8, 8, 128)        147584

batch_normalization_70 (Bat  (None, 8, 8, 128)        512
chNormalization)
```

```
max_pooling2d_41 (MaxPoolin  (None, 4, 4, 128)        0
g2D)

dropout_33 (Dropout)        (None, 4, 4, 128)        0

flatten_17 (Flatten)        (None, 2048)             0

dense_17 (Dense)            (None, 128)              262272

batch_normalization_71 (Bat  (None, 128)              512
chNormalization)

softmax_17 (Softmax)        (None, 128)              0

=================================================================
Total params: 551,584
Trainable params: 550,432
Non-trainable params: 1,152
_________________________________________________________________
```

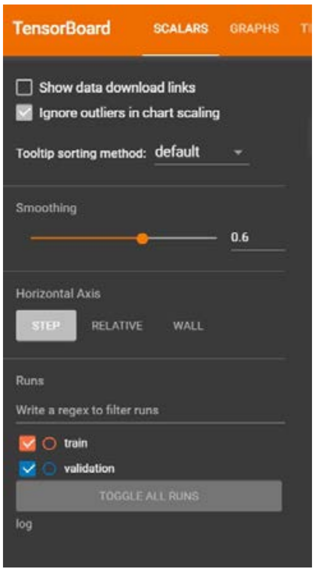
These figures show the model summary.

```
[59] model.fit(x_train, y_train, batch_size=64, epochs=100, callbacks=[tensorboard_callback], validation_data=(x_test, y_test))
     782/782 [==============================] - 8s 10ms/step - loss: 0.1445 - accuracy: 0.9501 - val_loss: 0.4680 - val_accuracy: 0.8662
     Epoch 73/100
     782/782 [==============================] - 8s 10ms/step - loss: 0.1453 - accuracy: 0.9503 - val_loss: 0.4391 - val_accuracy: 0.8755
     Epoch 74/100
     782/782 [==============================] - 8s 10ms/step - loss: 0.1442 - accuracy: 0.9511 - val_loss: 0.4468 - val_accuracy: 0.8711
     Epoch 75/100
     782/782 [==============================] - 8s 10ms/step - loss: 0.1358 - accuracy: 0.9545 - val_loss: 0.4496 - val_accuracy: 0.8707
     Epoch 76/100
     782/782 [==============================] - 8s 10ms/step - loss: 0.1408 - accuracy: 0.9523 - val_loss: 0.4698 - val_accuracy: 0.8676
     Epoch 77/100
     782/782 [==============================] - 8s 10ms/step - loss: 0.1349 - accuracy: 0.9530 - val_loss: 0.4755 - val_accuracy: 0.8672
     Epoch 78/100
     782/782 [==============================] - 8s 10ms/step - loss: 0.1351 - accuracy: 0.9545 - val_loss: 0.4468 - val_accuracy: 0.8750
     Epoch 79/100
     782/782 [==============================] - 8s 10ms/step - loss: 0.1322 - accuracy: 0.9537 - val_loss: 0.4672 - val_accuracy: 0.8729
     Epoch 80/100
     782/782 [==============================] - 8s 10ms/step - loss: 0.1361 - accuracy: 0.9533 - val_loss: 0.4710 - val_accuracy: 0.8725
     Epoch 81/100
     782/782 [==============================] - 8s 10ms/step - loss: 0.1341 - accuracy: 0.9534 - val_loss: 0.4841 - val_accuracy: 0.8656
     Epoch 82/100
     782/782 [==============================] - 8s 10ms/step - loss: 0.1342 - accuracy: 0.9542 - val_loss: 0.4821 - val_accuracy: 0.8674
     Epoch 83/100
     782/782 [==============================] - 8s 10ms/step - loss: 0.1307 - accuracy: 0.9556 - val_loss: 0.4905 - val_accuracy: 0.8660
     Epoch 84/100
     782/782 [==============================] - 8s 10ms/step - loss: 0.1305 - accuracy: 0.9554 - val_loss: 0.4726 - val_accuracy: 0.8683
     Epoch 85/100
     782/782 [==============================] - 8s 10ms/step - loss: 0.1270 - accuracy: 0.9566 - val_loss: 0.4663 - val_accuracy: 0.8714
     Epoch 86/100
     782/782 [==============================] - 8s 10ms/step - loss: 0.1301 - accuracy: 0.9542 - val_loss: 0.4691 - val_accuracy: 0.8709
     Epoch 87/100
     782/782 [==============================] - 8s 10ms/step - loss: 0.1275 - accuracy: 0.9570 - val_loss: 0.4646 - val_accuracy: 0.8714
     Epoch 88/100
     782/782 [==============================] - 8s 10ms/step - loss: 0.1257 - accuracy: 0.9563 - val_loss: 0.4762 - val_accuracy: 0.8698
     Epoch 89/100
```

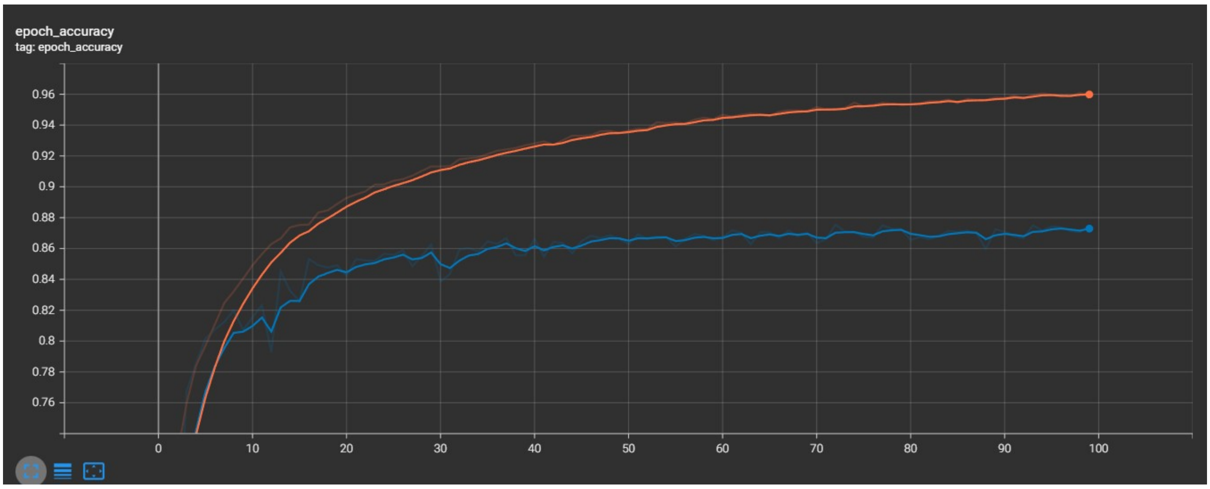
Then, fit the model. My batch size is 64, with 100 epochs. The validation data parameter here uses the variable x_test and y_test that we assigned it beforehand. Also use Google Colab's GPU because if not, its going to take too long time. After the training is completed, we will check the results.

```
model.evaluate(x_train, y_train)

1563/1563 [==============================] - 6s 4ms/step - loss: 0.0066 - accuracy: 0.9995
[0.0065909819677472115, 0.9995200037956238]

[61] model.evaluate(x_test, y_test)

313/313 [==============================] - 1s 4ms/step - loss: 0.4731 - accuracy: 0.8749
[0.47305503487586975, 0.8748999834060669]
```

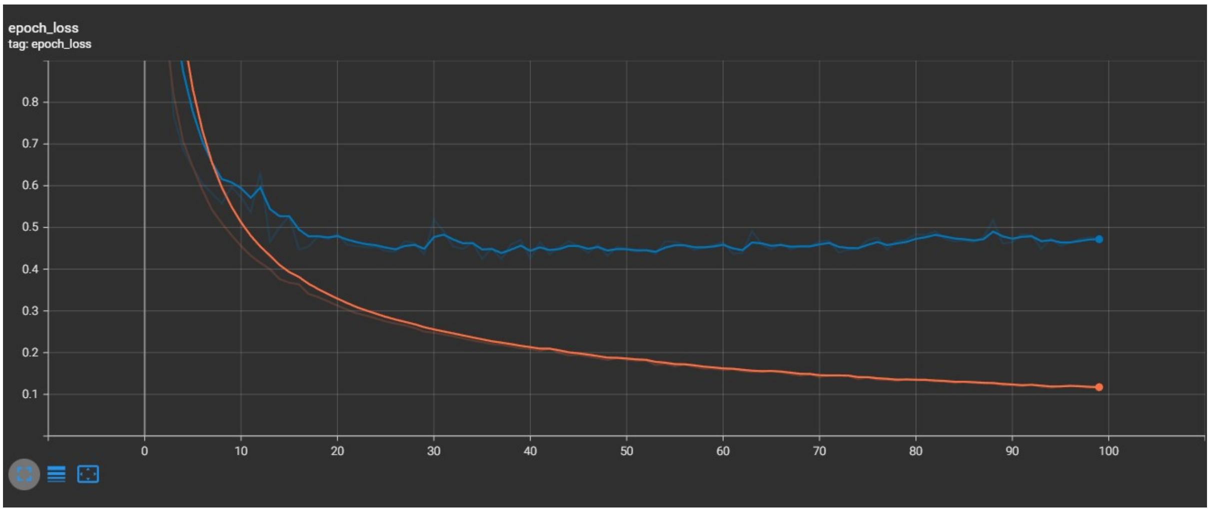
Here you can see that the training loss is 0.0066, the training accuracy is 0.9995, the testing loss is 0.4731, and the testing accuracy is 0.8749. I tried changing parameters for a several times, and this one was my best result. Now, let's see the Tensorboard graph results.



Orange line means training, blue line means validation (testing)



Here is the accuracy in each epoch of training (orange) and testing (blue) data.



Here is the loss in each epoch of training (orange) and testing (blue) data.

Here is the visualization of CNN architecture.