```
import tensorflow as tf

# Display the version
print(tf.__version__)

# other imports
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Input, Conv2D, Dense, Flatten, Dropout
from tensorflow.keras.layers import GlobalMaxPooling2D, MaxPooling2D
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.models import Model
```

```
2.14.0
```

```
# Load in the data
cifar10 = tf.keras.datasets.cifar10

# Distribute it to train and test set
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
print(x_train.shape, y_train.shape, x_test.shape, y_test.shape)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 4s 0us/step
(50000, 32, 32, 3) (50000, 1) (10000, 32, 32, 3) (10000, 1)
```

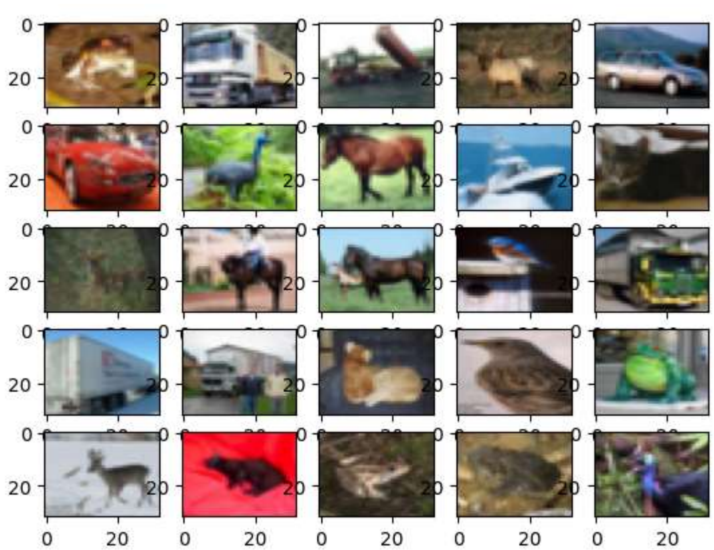
```
# Reduce pixel values
x_train, x_test = x_train / 255.0, x_test / 255.0

# flatten the label values
y_train, y_test = y_train.flatten(), y_test.flatten()
```

```
# visualize data by plotting images
fig, ax = plt.subplots(5, 5)
k = 0

for i in range(5):
    for j in range(5):
        ax[i][j].imshow(x_train[k], aspect='auto')
        k += 1

plt.show()
```



```
# number of classes
K = len(set(y_train))
```

```
# calculate total number of classes
# for output layer
print("number of classes:", K)

# Build the model using the functional API
# input layer
i = Input(shape=x_train[0].shape)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(i)
x = BatchNormalization()(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)

x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)

x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)

x = Flatten()(x)
x = Dropout(0.2)(x)

# Hidden layer
x = Dense(1024, activation='relu')(x)
x = Dropout(0.2)(x)

# last hidden layer i.e.. output layer
x = Dense(K, activation='softmax')(x)

model = Model(i, x)

# model description
model.summary()
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 32, 32, 3)]       0

 conv2d (Conv2D)             (None, 32, 32, 32)        896

 batch_normalization (Batch  (None, 32, 32, 32)        128
 Normalization)

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        9248

 batch_normalization_1 (Bat  (None, 32, 32, 32)        128
 chNormalization)

 max_pooling2d (MaxPooling2  (None, 16, 16, 32)        0
 D)

 conv2d_2 (Conv2D)           (None, 16, 16, 64)        18496

 batch_normalization_2 (Bat  (None, 16, 16, 64)        256
 chNormalization)

 conv2d_3 (Conv2D)           (None, 16, 16, 64)        36928

 batch_normalization_3 (Bat  (None, 16, 16, 64)        256
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 8, 8, 64)          0
 g2D)

 conv2d_4 (Conv2D)           (None, 8, 8, 128)         73856

 batch_normalization_4 (Bat  (None, 8, 8, 128)         512
 chNormalization)

 conv2d_5 (Conv2D)           (None, 8, 8, 128)         147584
```

```
 max_pooling2d_2 (MaxPoolin  (None, 4, 4, 128)         0
 g2D)

 flatten (Flatten)           (None, 2048)              0

 dropout (Dropout)           (None, 2048)              0

 dense (Dense)               (None, 1024)              2098176

 dropout_1 (Dropout)         (None, 1024)              0

 dense_1 (Dense)             (None, 10)                10250

=================================================================
Total params: 2397226 (9.14 MB)
Trainable params: 2396330 (9.14 MB)
Non-trainable params: 896 (3.50 KB)
```

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

```
# Fit
r = model.fit(
x_train, y_train, validation_data=(x_test, y_test), epochs=50)
```

```
Epoch 1/50
1563/1563 [==============================] - 495s 315ms/step - loss: 1.3326 - accuracy: 0.5421 - val_loss: 1.1686 - val_accuracy: 0.6140
Epoch 2/50
1563/1563 [==============================] - 491s 314ms/step - loss: 0.8561 - accuracy: 0.7017 - val_loss: 1.1013 - val_accuracy: 0.6201
Epoch 3/50
1563/1563 [==============================] - 485s 310ms/step - loss: 0.7016 - accuracy: 0.7589 - val_loss: 0.8520 - val_accuracy: 0.7170
Epoch 4/50
1563/1563 [==============================] - 528s 338ms/step - loss: 0.5956 - accuracy: 0.7939 - val_loss: 0.6855 - val_accuracy: 0.7738
Epoch 5/50
1563/1563 [==============================] - 498s 318ms/step - loss: 0.5007 - accuracy: 0.8267 - val_loss: 0.7451 - val_accuracy: 0.7593
Epoch 6/50
 802/1563 [==============>...............] - ETA: 3:41 - loss: 0.4118 - accuracy: 0.8576
```

```
# Fit with data augmentation
# Note: if you run this AFTER calling
# the previous model.fit()
# it will CONTINUE training where it left off
batch_size = 32
data_generator = tf.keras.preprocessing.image.ImageDataGenerator(
width_shift_range=0.1, height_shift_range=0.1, horizontal_flip=True)

train_generator = data_generator.flow(x_train, y_train, batch_size)
steps_per_epoch = x_train.shape[0] // batch_size

r = model.fit(train_generator, validation_data=(x_test, y_test),
            steps_per_epoch=steps_per_epoch, epochs=50)
```

```
# Plot accuracy per iteration
plt.plot(r.history['accuracy'], label='acc', color='red')
plt.plot(r.history['val_accuracy'], label='val_acc', color='green')
plt.legend()
```

```
# label mapping

labels = '''airplane automobile bird cat deerdog frog horseship truck'''.split()

# select the image from our test dataset
image_number = 0

# display the image
plt.imshow(x_test[image_number])

# load the image in an array
n = np.array(x_test[image_number])
```

```
# reshape it
p = n.reshape(1, 32, 32, 3)

# pass in the network for prediction and
# save the predicted label
predicted_label = labels[model.predict(p).argmax()]

# load the original label
original_label = labels[y_test[image_number]]

# display the result
print("Original label is {} and predicted label is {}".format(
  original_label, predicted_label))


# save the model
model.save('TensorFlow-08_CIFAR-10 Image Classification in TensorFlow.h5')
```