```
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
```

```
2.19.0
```

```
# 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()
```

```
number of classes: 10
```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 32, 32, 3)	0
conv2d (Conv2D)	(None, 32, 32, 32)	896
batch_normalization (BatchNormalization)	(None, 32, 32, 32)	128
conv2d_1 (Conv2D)	(None, 32, 32, 32)	9,248
batch_normalization_1 (BatchNormalization)	(None, 32, 32, 32)	128
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_2 (Conv2D)	(None, 16, 16, 64)	18,496
batch_normalization_2 (BatchNormalization)	(None, 16, 16, 64)	256
conv2d_3 (Conv2D)	(None, 16, 16, 64)	36,928
batch_normalization_3 (BatchNormalization)	(None, 16, 16, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 64)	0
conv2d_4 (Conv2D)	(None, 8, 8, 128)	73,856
batch_normalization_4 (BatchNormalization)	(None, 8, 8, 128)	512
conv2d_5 (Conv2D)	(None, 8, 8, 128)	147,584
batch_normalization_5 (BatchNormalization)	(None, 8, 8, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 128)	0
flatten (Flatten)	(None, 2048)	0
dropout (Dropout)	(None, 2048)	0
dense (Dense)	(None, 1024)	2,098,176
dropout_1 (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 10)	10,250

```
# 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=5)
```

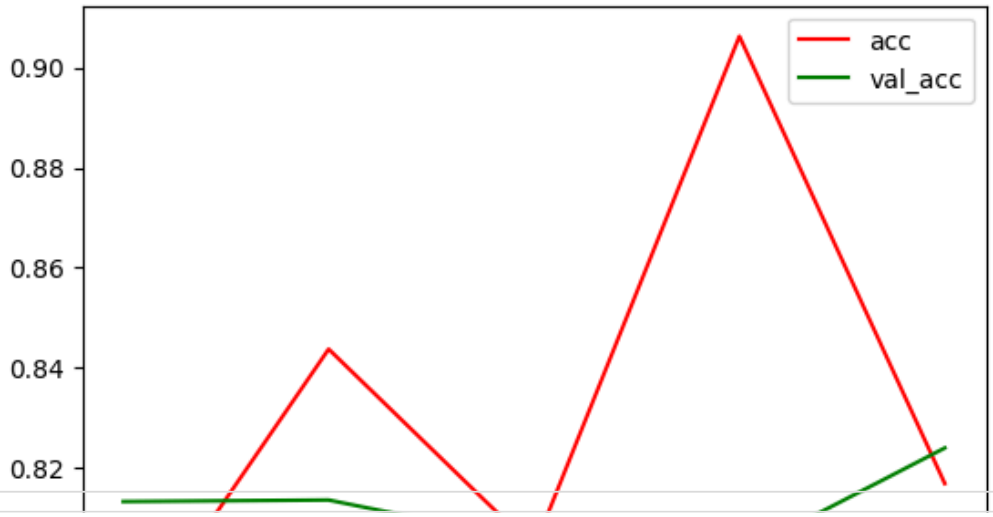
```
Epoch 1/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 411s 263ms/step - accuracy: 0.7802 - loss: 0.6436 - val_accur
Epoch 2/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 409s 261ms/step - accuracy: 0.8156 - loss: 0.5345 - val_accur
Epoch 3/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 410s 262ms/step - accuracy: 0.8424 - loss: 0.4505 - val_accur
Epoch 4/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 443s 263ms/step - accuracy: 0.8695 - loss: 0.3730 - val_accur
Epoch 5/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 442s 263ms/step - accuracy: 0.8910 - loss: 0.3133 - val_accur
```

```
# 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=5)
```

```
Epoch 1/5
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.
  self._warn_if_super_not_called()
1562/1562 ━━━━━━━━━━━━━━━━━━━━ 454s 291ms/step - accuracy: 0.7795 - loss: 0.6734 - val_accur
Epoch 2/5
   1/1562 ━━━━━━━━━━━━━━━━━━━━ 5:50 225ms/step - accuracy: 0.8438 - loss: 0.4647/usr/local/l
  self._interrupted_warning()
1562/1562 ━━━━━━━━━━━━━━━━━━━━ 19s 12ms/step - accuracy: 0.8438 - loss: 0.4647 - val_accurac
Epoch 3/5
1562/1562 ━━━━━━━━━━━━━━━━━━━━ 431s 276ms/step - accuracy: 0.8067 - loss: 0.5765 - val_accur
Epoch 4/5
1562/1562 ━━━━━━━━━━━━━━━━━━━━ 19s 12ms/step - accuracy: 0.9062 - loss: 0.3415 - val_accurac
Epoch 5/5
1562/1562 ━━━━━━━━━━━━━━━━━━━━ 428s 274ms/step - accuracy: 0.8170 - loss: 0.5398 - val_accur
```

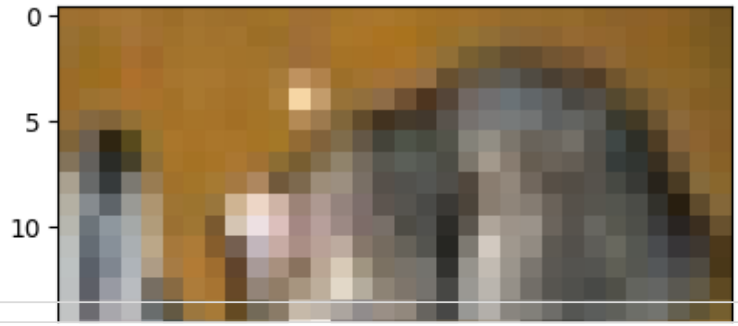
```
# Plot accuracy per iteration
plt.plot(r.history['accuracy'], label='acc', color='red')
plt.plot(r.history['val_accuracy'], label='val_acc', color='green')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x789f01ccf890>
```




```
# 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))
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 253ms/step
Original label is cat and predicted label is frog
```




```
# save the model
model.save('geeksforgeeks.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.s
```