```
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
```

```
2.8.2
```

```
# 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
170500096/170498071 [==============================] - 2s 0us/step
170508288/170498071 [==============================] - 2s 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: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
```

```
=================================================================
 input_1 (InputLayer)        [(None, 32, 32, 3)]       0

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

 batch_normalization (BatchN  (None, 32, 32, 32)       128
 ormalization)

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

 batch_normalization_1 (Batc  (None, 32, 32, 32)       128
 hNormalization)

 max_pooling2d (MaxPooling2D  (None, 16, 16, 32)       0
 )

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

 batch_normalization_2 (Batc  (None, 16, 16, 64)       256
 hNormalization)

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

 batch_normalization_3 (Batc  (None, 16, 16, 64)       256
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 64)         0
 2D)

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

 batch_normalization_4 (Batc  (None, 8, 8, 128)        512
 hNormalization)

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

 batch_normalization_5 (Batc  (None, 8, 8, 128)        512
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 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: 2,397,226
```

```
# 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=20)
```

```
Epoch 1/20
1563/1563 [==============================] - 483s 308ms/step - loss: 1.3193 - accura
Epoch 2/20
1563/1563 [==============================] - 474s 303ms/step - loss: 0.8521 - accura
Epoch 3/20
1563/1563 [==============================] - 466s 298ms/step - loss: 0.6968 - accura
Epoch 4/20
1563/1563 [==============================] - 467s 299ms/step - loss: 0.5822 - accura
Epoch 5/20
1563/1563 [==============================] - 469s 300ms/step - loss: 0.4998 - accura
Epoch 6/20
1563/1563 [==============================] - 466s 298ms/step - loss: 0.4243 - accura
Epoch 7/20
1563/1563 [==============================] - 460s 295ms/step - loss: 0.3649 - accura
Epoch 8/20
1563/1563 [==============================] - 461s 295ms/step - loss: 0.3028 - accura
Epoch 9/20
1563/1563 [==============================] - 463s 296ms/step - loss: 0.2562 - accura
Epoch 10/20
1563/1563 [==============================] - 460s 295ms/step - loss: 0.2316 - accura
Epoch 11/20
1563/1563 [==============================] - 460s 295ms/step - loss: 0.1993 - accura
Epoch 12/20
1563/1563 [==============================] - 460s 294ms/step - loss: 0.1760 - accura
Epoch 13/20
1563/1563 [==============================] - 464s 297ms/step - loss: 0.1593 - accura
Epoch 14/20
1563/1563 [==============================] - 470s 301ms/step - loss: 0.1399 - accura
Epoch 15/20
1563/1563 [==============================] - 467s 299ms/step - loss: 0.1379 - accura
Epoch 16/20
1563/1563 [==============================] - 469s 300ms/step - loss: 0.1296 - accura
Epoch 17/20
1563/1563 [==============================] - 472s 302ms/step - loss: 0.1177 - accura
Epoch 18/20
1563/1563 [==============================] - 468s 300ms/step - loss: 0.1073 - accura
Epoch 19/20
1563/1563 [==============================] - 465s 297ms/step - loss: 0.1070 - accura
Epoch 20/20
1563/1563 [==============================] - 465s 298ms/step - loss: 0.0981 - accura
```

```
# Fit with data augmentation
# # if you run this after calling
# the previous model.fit()
# it will continue training where it left off


# ImageDataGenerator helps to bring about augmentation according to given changes

batch_size = 32
data_generator = tf.keras.preprocessing.image.ImageDataGenerator(
  width_shift_range=0.1, height_shift_range=0.1, horizontal_flip=True)
```

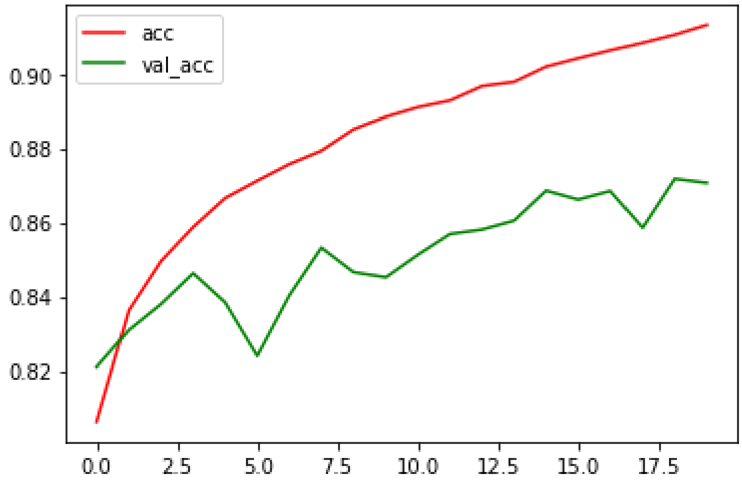
```
 # .flow loads the image dataset in memory and generates batches of augmented data

train_generator = data_generator.flow(x_train, y_train, batch_size)
steps_per_epoch = x_train.shape[0] // batch_size
 # steps_per_epoch how many batches of samples to use in one epoch

# Fitting data with augmented data

r = model.fit(train_generator, validation_data=(x_test, y_test),
              steps_per_epoch=steps_per_epoch, epochs=20)
```

```
Epoch 1/20
1562/1562 [==============================] - 487s 311ms/step - loss: 0.6017 - accurac
Epoch 2/20
1562/1562 [==============================] - 486s 311ms/step - loss: 0.4841 - accurac
Epoch 3/20
1562/1562 [==============================] - 485s 310ms/step - loss: 0.4459 - accurac
Epoch 4/20
1562/1562 [==============================] - 486s 311ms/step - loss: 0.4176 - accurac
Epoch 5/20
1562/1562 [==============================] - 487s 312ms/step - loss: 0.3932 - accurac
Epoch 6/20
1562/1562 [==============================] - 485s 311ms/step - loss: 0.3820 - accurac
Epoch 7/20
1562/1562 [==============================] - 484s 310ms/step - loss: 0.3650 - accurac
Epoch 8/20
1562/1562 [==============================] - 487s 312ms/step - loss: 0.3554 - accurac
Epoch 9/20
1562/1562 [==============================] - 486s 311ms/step - loss: 0.3421 - accurac
Epoch 10/20
1562/1562 [==============================] - 485s 310ms/step - loss: 0.3250 - accurac
Epoch 11/20
1562/1562 [==============================] - 479s 307ms/step - loss: 0.3191 - accurac
Epoch 12/20
1562/1562 [==============================] - 488s 312ms/step - loss: 0.3146 - accurac
Epoch 13/20
1562/1562 [==============================] - 487s 312ms/step - loss: 0.3015 - accurac
Epoch 14/20
1562/1562 [==============================] - 485s 311ms/step - loss: 0.2951 - accurac
Epoch 15/20
1562/1562 [==============================] - 483s 309ms/step - loss: 0.2873 - accurac
Epoch 16/20
1562/1562 [==============================] - 488s 312ms/step - loss: 0.2806 - accurac
Epoch 17/20
1562/1562 [==============================] - 483s 309ms/step - loss: 0.2722 - accurac
Epoch 18/20
1562/1562 [==============================] - 485s 310ms/step - loss: 0.2686 - accurac
Epoch 19/20
1562/1562 [==============================] - 483s 309ms/step - loss: 0.2597 - accurac
Epoch 20/20
1562/1562 [==============================] - 485s 310ms/step - loss: 0.2595 - accurac
```

```
# 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 0x7f5e312306d0>
```



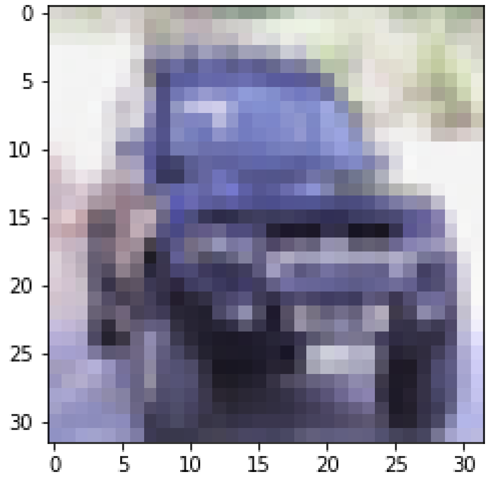

```
loss, accuracy = model.evaluate(train_generator, verbose=1)
loss_v, accuracy_v = model.evaluate(x_test, y_test, verbose=1)
print("Validation: accuracy = %f  ;  loss_v = %f" % (accuracy_v, loss_v))
print("Train: accuracy = %f  ;  loss = %f" % (accuracy, loss))
```

```
1563/1563 [==============================] - 131s 84ms/step - loss: 0.1776 - accuracy
313/313 [==============================] - 22s 70ms/step - loss: 0.4476 - accuracy: 0
Validation: accuracy = 0.870700  ;  loss_v = 0.447611
Train: accuracy = 0.938740  ;  loss = 0.177566
```

```
# label mapping

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

# select the image from our test dataset
image_number = 9

# 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))
```

```
Original label is automobile and predicted label is automobile
```




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

✓ 0s completed at 2:50 PM