```
! wget https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
```

```
--2024-04-12 08:21:23--  https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
Resolving www.cs.toronto.edu (www.cs.toronto.edu)... 128.100.3.30
Connecting to www.cs.toronto.edu (www.cs.toronto.edu)|128.100.3.30|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 170498071 (163M) [application/x-gzip]
Saving to: ‘cifar-10-python.tar.gz’

cifar-10-python.tar 100%[===================>] 162.60M  72.1MB/s    in 2.3s

2024-04-12 08:21:26 (72.1 MB/s) - ‘cifar-10-python.tar.gz’ saved [170498071/170498071]
```

```
!tar -xvf  /content/cifar-10-python.tar.gz
```

```
cifar-10-batches-py/
cifar-10-batches-py/data_batch_4
cifar-10-batches-py/readme.html
cifar-10-batches-py/test_batch
cifar-10-batches-py/data_batch_3
cifar-10-batches-py/batches.meta
cifar-10-batches-py/data_batch_2
cifar-10-batches-py/data_batch_5
cifar-10-batches-py/data_batch_1
```

```
!pip install keras
!pip show keras
```

```
Requirement already satisfied: keras in /usr/local/lib/python3.10/dist-packages (2.15.0)
Name: keras
Version: 2.15.0
Summary: Deep learning for humans.
Home-page: https://keras.io/
Author: Keras team
Author-email: keras-users@googlegroups.com
License: Apache 2.0
Location: /usr/local/lib/python3.10/dist-packages
Requires:
Required-by: tensorflow
```

```
import numpy as np
import matplotlib.pyplot as plt
import os

from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import classification_report, confusion_matrix

import tensorflow as tf
import tensorflow.keras as k
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten, Dropout, BatchNormali
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.preprocessing.image import ImageDataGenerator

from six.moves import cPickle as pickle


def load_CIFAR_batch(filename):
    with open(filename, 'rb') as f:
        datadict = pickle.load(f, encoding='latin1')
        X = datadict['data']
        Y = datadict['labels']
        X = np.array(X)
        Y = np.array(Y)
        return X, Y



def load_CIFAR10(path):
    xs = []
    ys = []

    for number in range(1,6):
        file = os.path.join(path, 'data_batch_%d' %(number))
        X, Y = load_CIFAR_batch(file)
        xs.append(X)
        ys.append(Y)

    Xtrain = np.concatenate(xs)
    Ytrain = np.concatenate(ys)
    Xtest, Ytest = load_CIFAR_batch(os.path.join(path, 'test_batch'))
    return Xtrain, Ytrain, Xtest, Ytest


def get_CIFAR10_data():
    X_train, y_train, X_test, y_test = load_CIFAR10('/content/cifar-10-batches-py')
    x_train = X_train.astype('float32')
    x_test = X_test.astype('float32')
    x_train /= 255
    x_test /= 255
    return x_train, y_train, x_test, y_test
```

```
x_train, y_train, x_test, y_test = get_CIFAR10_data()

print('Train data shape  : ', x_train.shape)
print('Train labels shape: ', y_train.shape)
print('Test data shape   : ', x_test.shape)
print('Test labels shape : ', y_test.shape)
```

```
Train data shape  :  (50000, 3072)
Train labels shape:  (50000,)
Test data shape   :  (10000, 3072)
Test labels shape :  (10000,)
```

```
print('The training data contains %d images and %d labels' %(x_train.shape[0], y_train.shape
print('The testing  data contains %d images and %d labels' %(x_test.shape[0], y_test.shape[0
```

```
The training data contains 50000 images and 50000 labels
The testing  data contains 10000 images and 10000 labels
```

```
def rotate(imgs):
    for i in range(imgs.shape[0]):
        imgs[i] = np.rot90(imgs[i], k = -1)
    return imgs


def convert_into_images(data):
    data_shaped = data.reshape(data.shape[0], 3, 32, 32)
    data_swaped = np.swapaxes(data_shaped, 1, 3)
    data_rot = rotate(data_swaped)
    return data_rot
```

```
!pip install convert_images
!pip show convert_images
```

```
Collecting convert_images
  Downloading convert_images-0.1.0-py3-none-any.whl (3.5 kB)
Requirement already satisfied: Pillow>7 in /usr/local/lib/python3.10/dist-packages (from
Collecting loguru<0.7.0,>=0.6.0 (from convert_images)
  Downloading loguru-0.6.0-py3-none-any.whl (58 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.3/58.3 kB 1.7 MB/s eta 0:00:00
Installing collected packages: loguru, convert_images
Successfully installed convert_images-0.1.0 loguru-0.6.0
Name: convert-images
Version: 0.1.0
Summary: Simple CLI to convert images to JPEG and PNG format
Home-page:
Author: Mohammad Alyetama
Author-email: malyetama@pm.me
License: MIT
```

```
      Location: /usr/local/lib/python3.10/dist-packages
      Requires: loguru, Pillow
      Required-by:
```

```
x_train = convert_into_images(x_train)
x_test = convert_into_images(x_test)
```

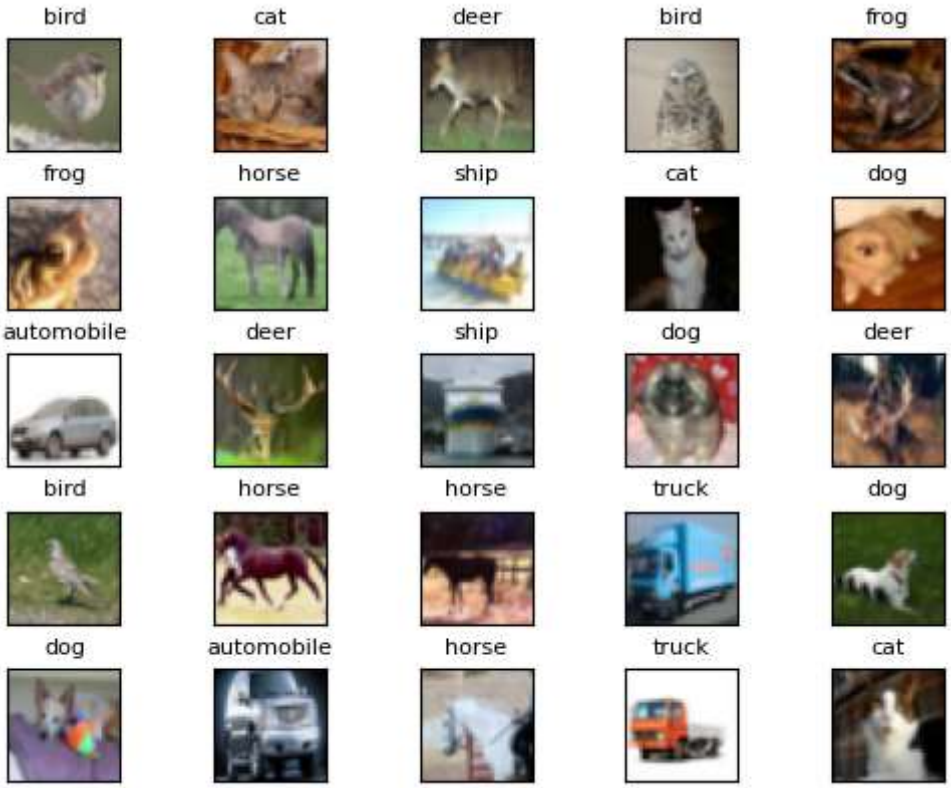
```
print('the training data has %d images of the shape : (%d, %d, %d)' % (x_train.shape[0], x_t
print('the testing  data has %d images of the shape : (%d, %d, %d)' % (x_test.shape[0], x_te
```

```
     the training data has 50000 images of the shape : (32, 32, 3)
     the testing  data has 10000 images of the shape : (32, 32, 3)
```

```
labels = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', '
```

```
def plot_sample_images(xdata, ydata):
    f, ax = plt.subplots(5, 5)
    for row in range(5):
        for col in range(5):
            idx = np.random.randint(0, xdata.shape[0])
            ax[row,col].imshow(xdata[idx])
            ax[row,col].set_title(labels[ydata[idx]], fontsize = 8)
            ax[row,col].get_xaxis().set_visible(False)
            ax[row,col].get_yaxis().set_visible(False)
    plt.subplots_adjust(hspace=0.4)
    plt.show()
```

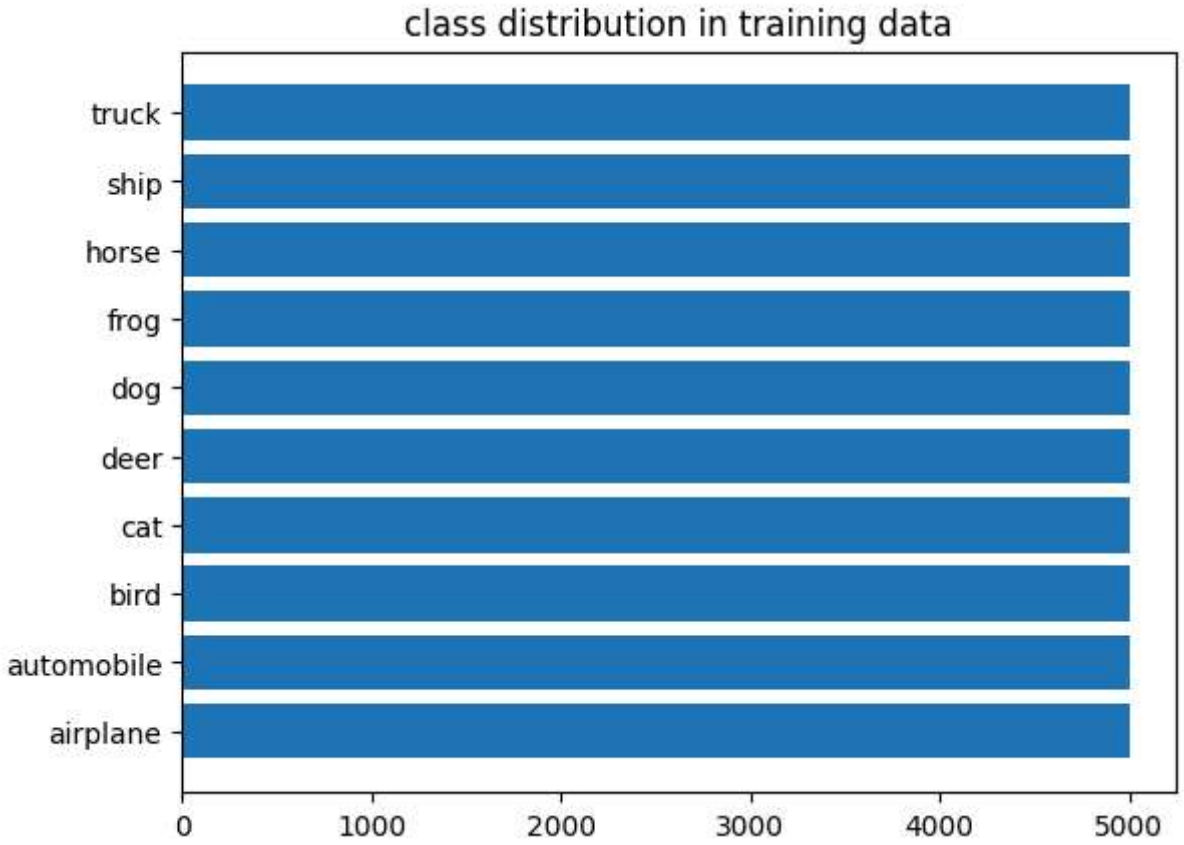
```
plot_sample_images(x_train, y_train)
```




```
plot_sample_images(x_test, y_test)
```

```
classes, counts = np.unique(y_train, return_counts=True)
plt.barh(labels, counts)
plt.title('class distribution in training data')
```

```
Text(0.5, 1.0, 'class distribution in training data')
```




```
classes, counts = np.unique(y_test, return_counts=True)
plt.barh(labels, counts)
plt.title('Class distribution in testing data')
```

```
Text(0.5, 1.0, 'Class distribution in testing data')
```


```
categorical_y_train = to_categorical(y_train, 10)
categorical_y_test = to_categorical(y_test, 10)
```

```
model = Sequential()

model.add(Conv2D(filters=32, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu',
model.add(BatchNormalization())
model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(10, activation='softmax'))


METRICS = ['accuracy',
    tf.keras.metrics.Precision(name='precision'),
    tf.keras.metrics.Recall(name='recall')
]


model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=METRICS)


model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 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)

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

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)

dropout_1 (Dropout)         (None, 8, 8, 64)          0

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

batch_normalization_5 (Bat  (None, 8, 8, 128)         512
chNormalization)

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

dropout_2 (Dropout)         (None, 4, 4, 128)         0

flatten (Flatten)           (None, 2048)              0

dense (Dense)               (None, 128)               262272

dropout_3 (Dropout)         (None, 128)               0

dense_1 (Dense)             (None, 64)                8256

dropout_4 (Dropout)         (None, 64)                0
```

```
data_generator = ImageDataGenerator(width_shift_range=0.1, height_shift_range=0.1, horizonta
train_generator = data_generator.flow(x_train, categorical_y_train, 32)
steps_per_epoch = x_train.shape[0] // 32
```

```
his = model.fit(train_generator, epochs=50, steps_per_epoch=steps_per_epoch, validation_data
```

```
Epoch 1/50
1562/1562 [==============================] - 488s 310ms/step - loss: 1.7447 - accuracy:
Epoch 2/50
 229/1562 [===>..........................] - ETA: 6:09 - loss: 1.4352 - accuracy: 0.4856
```

```
plt.figure(figsize=(8, 12))

plt.subplot(4, 2, 1)
plt.plot(his.history['loss'], label='Loss')
plt.plot(his.history['val_loss'], label='val_Loss')
plt.title('Loss Function Evolution')
plt.legend()
```