```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import numpy as np
```

```
(X_train, y_train), (X_test,y_test) = datasets.cifar10.load_data()

X_train.shape  # to check the  shape of x train 50,000 sample with 32 by 32 image with RGB
```

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

```
X_test.shape
```

```
(10000, 32, 32, 3)
```

```
y_train.shape
```

```
(50000, 1)
```

```
y_train[:5]   #y_train is a 2D array, for our classification having 1D array is good enoug
```

```
array([[6],
       [9],
       [9],
       [4],
       [1]], dtype=uint8)
```

```
y_train = y_train.reshape(-1,)
y_train[:5]
```

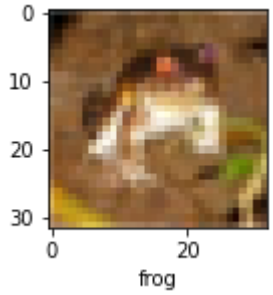
```
array([6, 9, 9, 4, 1], dtype=uint8)
```

```
y_test = y_test.reshape(-1,)
```

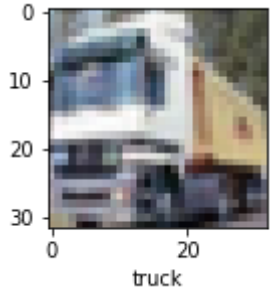
```
classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"
```

```
def plot_sample(X, y, index):
    plt.figure(figsize = (10,2))
    plt.imshow(X[index])
    plt.xlabel(classes[y[index]])
```

```
plot_sample(X_train, y_train, 0)
```




```
plot_sample(X_train, y_train, 1)
```

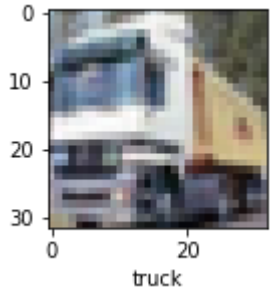



```
# Normalize the images to a number from 0 to 1. Image has 3 channels (R,G,B) and each valu
# Hence to normalize in 0-->1 range, we need to divide it by 255


# Normalizing the training data


X_train = X_train / 255.0
X_test = X_test / 255.0


plot_sample(X_train, y_train, 1)
```




```
# Build simple (ANN) artificial neural network for image classification


ann = models.Sequential([
        layers.Flatten(input_shape=(32,32,3)),
        layers.Dense(3000, activation='relu'),
        layers.Dense(1000, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
```

```
ann.compile(optimizer='SGD',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

ann.fit(X_train, y_train, epochs=5)
```

```
Epoch 1/5
1563/1563 [==============================] - 123s 78ms/step - loss: 1.8103 - accuracy
Epoch 2/5
1563/1563 [==============================] - 121s 77ms/step - loss: 1.6231 - accuracy
Epoch 3/5
1563/1563 [==============================] - 118s 76ms/step - loss: 1.5403 - accuracy
Epoch 4/5
1563/1563 [==============================] - 121s 78ms/step - loss: 1.4806 - accuracy
Epoch 5/5
1563/1563 [==============================] - 119s 76ms/step - loss: 1.4328 - accuracy
<keras.callbacks.History at 0x7ffa458a68d0>
```

```
#You can see that at the end of 5 epochs, accuracy is at around 49%


#from sklearn.metrics import confusion_matrix , classification_report
#import numpy as np
#y_pred = ann.predict(X_test)
#y_pred_classes = [np.argmax(element) for element in y_pred]

#print("Classification Report: \n", classification_report(y_test, y_pred_classes))


# Now let us build a convolutional neural network to train our images


cnn = models.Sequential([
    layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(32, 32,
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])


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


cnn.fit(X_train, y_train, epochs=10)
```

```
Epoch 1/10
1563/1563 [==============================] - 72s 46ms/step - loss: 1.4672 - accuracy
Epoch 2/10
1563/1563 [==============================] - 71s 46ms/step - loss: 1.1116 - accuracy
```

```
Epoch 3/10
1563/1563 [==============================] - 71s 45ms/step - loss: 0.9794 - accuracy
Epoch 4/10
1563/1563 [==============================] - 72s 46ms/step - loss: 0.9038 - accuracy
Epoch 5/10
1563/1563 [==============================] - 70s 45ms/step - loss: 0.8393 - accuracy
Epoch 6/10
1563/1563 [==============================] - 70s 45ms/step - loss: 0.7879 - accuracy
Epoch 7/10
1563/1563 [==============================] - 71s 46ms/step - loss: 0.7430 - accuracy
Epoch 8/10
1563/1563 [==============================] - 70s 45ms/step - loss: 0.6991 - accuracy
Epoch 9/10
1563/1563 [==============================] - 72s 46ms/step - loss: 0.6664 - accuracy
Epoch 10/10
1563/1563 [==============================] - 70s 45ms/step - loss: 0.6285 - accuracy
<keras.callbacks.History at 0x7ffa458221d0>
```

```
# With CNN, at the end 5 epochs, accuracy was at around 70% which is a significant improve
# CNN's are best for image classification and gives superb accuracy.
# Also computation is much less compared to simple ANN as maxpooling reduces the image dim
```

```
cnn.evaluate(X_test,y_test)
```

```
313/313 [==============================] - 5s 15ms/step - loss: 0.9475 - accuracy: 0
[0.9474993944168091, 0.6859999895095825]
```

```
y_pred = cnn.predict(X_test)
y_pred[:5]
```

```
313/313 [==============================] - 5s 15ms/step
array([[1.5868401e-03, 1.8028191e-03, 5.9748096e-03, 7.0118445e-01,
        2.1088107e-04, 1.5785888e-01, 9.3305893e-03, 4.7079670e-06,
        1.2123139e-01, 8.1466825e-04],
       [2.4813578e-02, 2.4031416e-02, 5.0115148e-05, 2.0153209e-06,
        2.3528816e-08, 2.2390243e-08, 4.7162453e-07, 5.4393973e-07,
        9.5101124e-01, 9.0612855e-05],
       [1.3396062e-01, 1.2232437e-01, 3.7134791e-04, 6.5783923e-03,
        5.0119538e-04, 5.6403758e-05, 2.3397022e-04, 5.8700405e-03,
        7.2316617e-01, 6.9373976e-03],
       [6.6001970e-01, 1.4544354e-03, 7.2645452e-03, 2.5712149e-04,
        8.8411319e-04, 4.7058384e-06, 7.2884301e-05, 2.1169943e-05,
        3.2991278e-01, 1.0842344e-04],
       [5.5461595e-07, 6.0095413e-06, 1.2652254e-02, 2.8258780e-02,
        2.8226715e-01, 3.7825224e-04, 6.7638308e-01, 6.6074762e-08,
        4.9438822e-05, 4.4286821e-06]], dtype=float32)
```

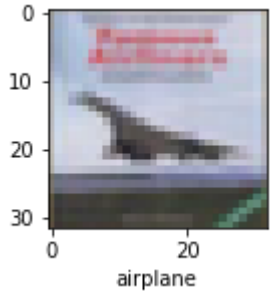
```
y_classes = [np.argmax(element) for element in y_pred]
y_classes[:6]
```

```
[3, 8, 8, 0, 6, 6]
```

```
y_test[:5]
```

```
array([3, 8, 8, 0, 6], dtype=uint8)
```

```
plot_sample(X_test, y_test,3)
```




```
classes[y_classes[3]]
```

```
'airplane'
```

```
classes[y_classes[100]]
```

```
'deer'
```