# Niboon Boonprakob 61340500038

# Object Recognition

The objective of this lab is very simple, to recognize objects in images. You will be working with a well-known dataset called CIFAR-10.

You can learn more about this dataset and download it here:

https://www.cs.toronto.edu/~kriz/cifar.html

In the webpage above, they also included a few publications based on CIFAR-10 data, which showed some amazing accuracies. The worst network on the page (a shallow convolutional neural network) can classify images with rouhgly 75% accuracy.

In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.
mount("/content/drive", force_remount=True).
```

# 1. Write a function to load data

The dataset webpage in the previous section also provide a simple way to load data from your harddrive using pickle. You may use their function for this exercise.

Construct two numpy arrays for train images and train labels from data_batch_1 to data_batch_5. Then, construct two numpy arrays for test images, and test labels from test batch file. The original image size is 32 x 32 x 3. You may flatten the arrays so the final arrays are of size 1 x 3072.

In [2]:
```
# Import Tensorflow 2.0
%tensorflow_version 2.x
import tensorflow as tf
import keras
import matplotlib.pyplot as plt
import numpy as np
import random
from tqdm import tqdm

# Check that we are using a GPU, if not switch runtimes
#   using Runtime > Change Runtime Type > GPU

assert len(tf.config.list_physical_devices('GPU')) > 0
```

In [3]:
```
def unpickle(file):
    import pickle
    with open(file, 'rb') as fo:
        dict = pickle.load(fo, encoding='bytes')
    return dict
```

In [4]:
```
pictures=[]
labels = []
for i in range(5):
```

```
  file = "/content/drive/MyDrive/Colab Notebooks/Lab 3/dataset/cifar-10-batches-py/d
  # print(file)
  dataset = unpickle(file)
  pictures.append(dataset[b"data"] )
  labels += dataset[b"labels"]
# print(np.array(pictures).shape)
pictures = np.concatenate(np.array(pictures) , axis = 0)
# print(path)
```

In [5]:
```
pictures
```

Out[5]:
```
array([[ 59,  43,  50, ..., 140,  84,  72],
       [154, 126, 105, ..., 139, 142, 144],
       [255, 253, 253, ...,  83,  83,  84],
       ...,
       [ 35,  40,  42, ...,  77,  66,  50],
       [189, 186, 185, ..., 169, 171, 171],
       [229, 236, 234, ..., 173, 162, 161]], dtype=uint8)
```

In [6]:
```
labels
```

Out[6]:
```
[6,
 9,
 9,
 4,
 1,
 1,
 2,
 7,
 8,
 3,
 4,
 7,
 7,
 2,
 9,
 9,
 9,
 3,
 2,
 6,
 4,
 3,
 6,
 6,
 2,
 6,
 3,
 5,
 4,
 0,
 0,
 9,
 1,
 3,
 4,
 0,
 3,
 7,
 3,
 3,
 5,
 2,
 2,
 7,
 1,
 1,
 1,
```

```
2,
2,
0,
9,
5,
7,
9,
2,
2,
5,
2,
4,
3,
1,
1,
8,
2,
1,
1,
4,
9,
7,
8,
5,
9,
6,
7,
3,
1,
9,
0,
3,
1,
3,
5,
4,
5,
7,
7,
4,
7,
9,
4,
2,
3,
8,
0,
1,
6,
1,
1,
4,
1,
8,
3,
9,
6,
6,
1,
8,
5,
2,
9,
9,
8,
1,
7,
7,
0,
```

```
0,
6,
9,
1,
2,
2,
9,
2,
6,
6,
1,
9,
5,
0,
4,
7,
6,
7,
1,
8,
1,
1,
2,
8,
1,
3,
3,
6,
2,
4,
9,
9,
5,
4,
3,
6,
7,
4,
6,
8,
5,
5,
4,
3,
1,
8,
4,
7,
6,
0,
9,
5,
1,
3,
8,
2,
7,
5,
3,
4,
1,
5,
7,
0,
4,
7,
5,
5,
1,
```

```
0,
9,
6,
9,
0,
8,
7,
8,
8,
2,
5,
2,
3,
5,
0,
6,
1,
9,
3,
6,
9,
1,
3,
9,
6,
6,
7,
1,
0,
9,
5,
8,
5,
2,
9,
0,
8,
8,
0,
6,
9,
1,
1,
6,
3,
7,
6,
6,
0,
6,
6,
1,
7,
1,
5,
8,
3,
6,
6,
8,
6,
8,
4,
6,
6,
1,
3,
8,
3,
```

```
4,
1,
7,
1,
3,
8,
5,
1,
1,
4,
0,
9,
3,
7,
4,
9,
9,
2,
4,
9,
9,
1,
0,
5,
9,
0,
8,
2,
1,
2,
0,
5,
6,
3,
2,
7,
8,
8,
6,
0,
7,
9,
4,
5,
6,
4,
2,
1,
1,
2,
1,
5,
9,
9,
0,
8,
4,
1,
1,
6,
3,
3,
9,
0,
7,
9,
7,
7,
9,
```

```
1,
5,
1,
6,
6,
8,
7,
1,
3,
0,
3,
3,
2,
4,
5,
7,
5,
9,
0,
3,
4,
0,
4,
4,
6,
0,
0,
6,
6,
0,
8,
1,
6,
2,
9,
2,
5,
9,
6,
7,
4,
1,
8,
7,
3,
6,
9,
3,
0,
4,
0,
5,
1,
0,
3,
4,
8,
5,
4,
7,
2,
3,
9,
7,
6,
7,
1,
4,
7,
```

```
0,
1,
7,
3,
1,
8,
4,
4,
2,
0,
2,
2,
0,
0,
9,
0,
9,
6,
8,
2,
7,
7,
4,
0,
3,
0,
8,
9,
4,
2,
7,
2,
5,
2,
5,
1,
9,
4,
8,
5,
1,
7,
4,
4,
0,
6,
9,
0,
7,
8,
8,
9,
9,
3,
3,
4,
0,
4,
5,
6,
6,
0,
1,
0,
8,
0,
4,
8,
8,
```

```
1,
5,
2,
6,
8,
1,
0,
0,
7,
7,
5,
9,
6,
2,
8,
3,
4,
7,
3,
9,
0,
1,
2,
4,
8,
1,
8,
6,
4,
4,
5,
7,
1,
3,
9,
8,
0,
1,
7,
5,
8,
2,
8,
0,
4,
1,
8,
9,
8,
2,
9,
9,
2,
7,
5,
7,
3,
8,
8,
4,
4,
2,
7,
1,
6,
4,
0,
4,
6,
```

```
9,
7,
6,
2,
5,
5,
1,
7,
2,
2,
2,
9,
5,
4,
2,
7,
8,
1,
3,
4,
3,
7,
6,
9,
8,
0,
6,
0,
2,
2,
2,
1,
8,
4,
0,
1,
8,
8,
1,
5,
7,
6,
4,
5,
8,
7,
1,
9,
1,
9,
8,
4,
7,
3,
8,
8,
2,
6,
6,
7,
1,
6,
8,
1,
9,
7,
8,
3,
0,
```

```
1,
0,
8,
8,
3,
0,
0,
1,
5,
0,
8,
8,
7,
9,
9,
0,
9,
4,
1,
3,
6,
6,
4,
4,
7,
5,
6,
0,
8,
0,
3,
2,
8,
4,
6,
9,
9,
7,
0,
3,
3,
6,
7,
4,
9,
1,
6,
2,
7,
2,
2,
0,
6,
7,
5,
7,
6,
8,
9,
0,
9,
4,
4,
7,
0,
9,
4,
9,
6,
```

```
9,
4,
5,
7,
9,
2,
4,
5,
1,
4,
3,
9,
6,
5,
6,
9,
3,
3,
5,
0,
7,
2,
1,
3,
6,
4,
0,
0,
2,
5,
0,
1,
0,
2,
3,
9,
8,
4,
9,
8,
0,
2,
6,
4,
4,
0,
1,
8,
8,
3,
6,
9,
6,
6,
7,
8,
2,
4,
5,
7,
6,
5,
3,
0,
5,
0,
5,
0,
8,
```

```
2,
6,
7,
3,
8,
2,
1,
7,
6,
7,
1,
0,
9,
5,
5,
0,
1,
7,
6,
9,
0,
4,
7,
7,
1,
5,
9,
4,
0,
8,
5,
9,
9,
6,
7,
1,
8,
3,
2,
3,
8,
2,
2,
4,
6,
0,
0,
5,
3,
8,
2,
3,
7,
2,
9,
3,
8,
7,
8,
2,
7,
9,
0,
2,
3,
2,
2,
2,
3,
```

```
3,
6,
2,
3,
2,
8,
0,
5,
5,
1,
4,
5,
6,
6,
2,
7,
0,
1,
7,
7,
8,
2,
9,
2,
2,
4,
2,
1,
1,
1,
6,
6,
6,
5,
1,
1,
7,
0,
4,
3,
3,
7,
1,
2,
3,
5,
5,
5,
6,
1,
4,
3,
7,
8,
8,
3,
6,
6,
2,
3,
0,
9,
4,
3,
8,
0,
0,
1,
1,
```

```
5,
4,
9,
3,
1,
8,
9,
3,
9,
9,
2,
9,
4,
8,
2,
9,
8,
8,
1,
5,
3,
6,
8,
7,
6,
9,
8,
0,
6,
4,
0,
0,
2,
5,
8,
2,
0,
2,
7,
6,
9,
7,
1,
5,
5,
6,
6,
3,
6,
2,
4,
7,
0,
5,
6,
4,
6,
5,
2,
4,
6,
1,
6,
0,
4,
0,
3,
1,
8,
```

```
5,
4,
4,
1,
7,
3,
9,
4,
7,
9,
7,
3,
7,
2,
8,
4,
6,
6,
1,
2,
9,
0,
4,
8,
7,
3,
9,
8,
7,
7,
0,
2,
4,
1,
1,
4,
1,
5,
4,
0,
5,
6,
2,
8,
5,
0,
2,
1,
3,
5,
7,
3,
5,
1,
3,
5,
...]
```

In [7]:
```
len(pictures) , len(labels)
```

Out[7]: (50000, 50000)

In [8]:
```
pictures.shape
```

Out[8]: (50000, 3072)

# 2. Classify Dogs v.s. Cats

Let's start simple by creating logistic regression model to classify images. We will select only two classes of images for this exercise.

1. From 50,000 train images and 10,000 test images, we want to reduce the data size. Write code to filter only dog images (label = 3) and cat images (label = 5).
2. Create a logistic regression model to classify cats and dogs. Report your accuracy.

In [9]:
```
labels.index(3)
```

Out[9]:
```
9
```

In [10]:
```
dog_labels = []
dog_img = []
cat_labels= []
cat_img = []

element_dog = 3
element_cat = 5

for i in range(len(labels)):
  if (labels[i] ==  element_dog):
    dog_labels.append(labels[i])
    dog_img.append(pictures[i])
  elif (labels[i] == element_cat):
    cat_labels.append(labels[i])
    cat_img.append(pictures[i])
```

In [11]:
```
np.array(dog_img).shape , np.array(cat_img).shape
```

Out[11]:
```
((5000, 3072), (5000, 3072))
```

In [12]:
```
img_all = [cat_img ,dog_img]
# np.array(img_all).shape
d_c_img = np.concatenate(np.array(img_all),axis = 0)
```

In [13]:
```
d_c_img.shape
```

Out[13]:
```
(10000, 3072)
```

In [14]:
```
labels_all = [cat_labels,dog_labels]
d_c_label = np.concatenate(np.array(labels_all),axis=0)
```

In [15]:
```
d_c_label.shape
```

Out[15]:
```
(10000,)
```

# import test_batch

In [16]:
```
test = "/content/drive/MyDrive/Colab Notebooks/Lab 3/dataset/cifar-10-batches-py/tes
y_test = unpickle(test)
y_test_data = y_test[b"data"]
y_test_labels = y_test[b"labels"]
```

In [17]:
```
len(y_test_data) , len(y_test_labels)
```

Out[17]: (10000, 10000)

In [18]:
```
# T : images , t:labels
T_y_test_dog = []
T_y_test_cat = []
t_y_test_dog = []
t_y_test_cat = []
element_dog = 3
element_cat = 5
for i in range(len(y_test_labels)):
  if (y_test_labels[i] == element_cat):
    t_y_test_cat.append(y_test_labels[i])
    T_y_test_cat.append(y_test_data[i])
  elif (y_test_labels[i] == element_dog):
    T_y_test_dog.append(y_test_data[i])
    t_y_test_dog.append(y_test_labels[i])

# y_test  dc_img_test : x_test , dc_labels_test : y_test
img_y_test = [T_y_test_dog , T_y_test_cat]
labels_y_test = [t_y_test_dog , t_y_test_cat]
dc_img_test = np.concatenate(np.array(img_y_test),axis = 0)
dc_labels_test = np.concatenate(np.array(labels_y_test),axis=0)
print(dc_img_test.shape , dc_labels_test.shape)
```

```
(2000, 3072) (2000,)
```

In [18]:

In [19]:
```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(random_state=42,max_iter=1000).fit(d_c_img, d_c_label)
y_pred = clf.predict(dc_img_test)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: Conver
genceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

In [20]:
```
from sklearn.metrics import accuracy_score
print(f"Accuracy: {accuracy_score(dc_labels_test , y_pred)}")
```

```
Accuracy: 0.561
```

# 3. The Real Challenge

The majority of your score for this lab will come from this real challenge. You are going to construct a neural network model to classify 10 classes of images from CIFAR-10 dataset. You will get half the credits for this one if you complete the assignment, and will get another half if you can exceed the target accuracy of 75%. (You may use any combination of sklearn, opencv, or tensorflow to do this exercise).

Design at least 3 variants of neural network models. Each model should have different architectures. (Do not vary just a few parameters, the architecture of the network must change in each model). In your notebook, explain your experiments in details and display the accuracy score for each experiment.

In [21]:
```
### load dataset ###

# X_train : pictures
# y_train : labels
# X_test : y_test_data
# y_test : y_test_labels
```

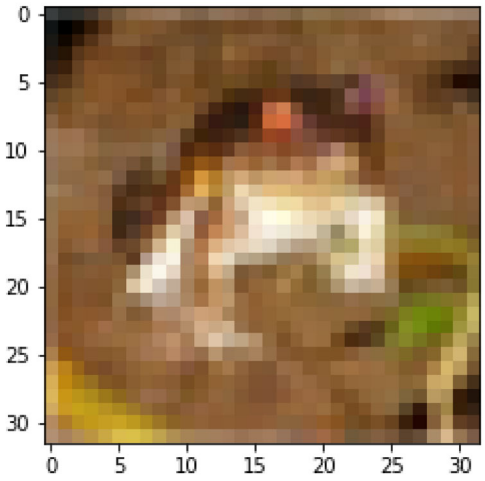
In [22]:
```
pictures.shape
```

Out[22]:
```
(50000, 3072)
```

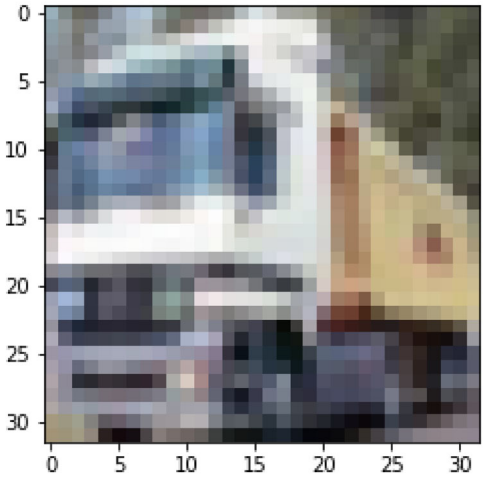
In [23]:
```
X_train = []
X_test = []
for i in range(len(pictures)):
  img = np.reshape(pictures[i], (3,32,32))
  X_train.append(np.transpose(img, (1, 2, 0)))
print(X_train[0].shape)

for i in range(len(y_test_data)):
  img = np.reshape(y_test_data[i], (3,32,32))
  X_test.append(np.transpose(img, (1, 2, 0)))
y_train = labels
y_test = y_test_labels
```

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

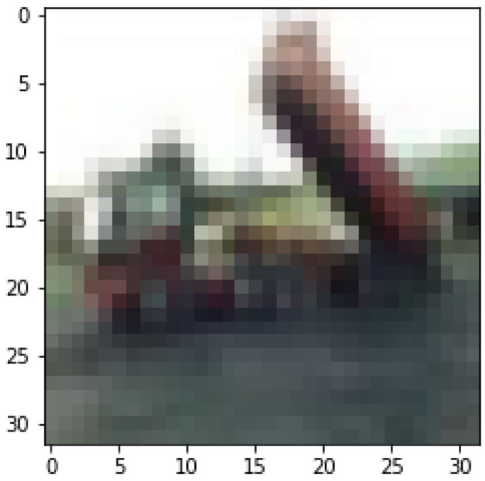
In [24]:
```
import matplotlib.pyplot as plt
for i in range(10):
  plt.imshow(X_train[i])
  plt.show()
  print(y_train[i])
```



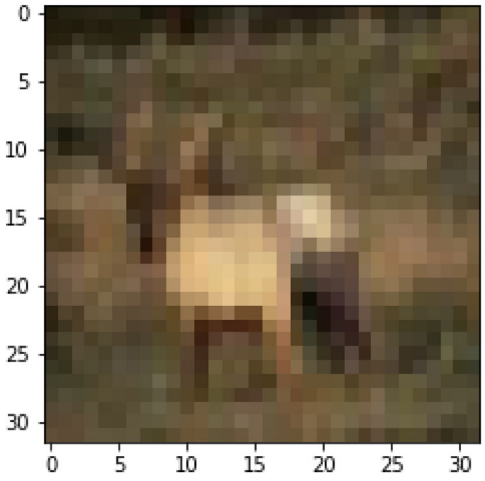

```
6
```



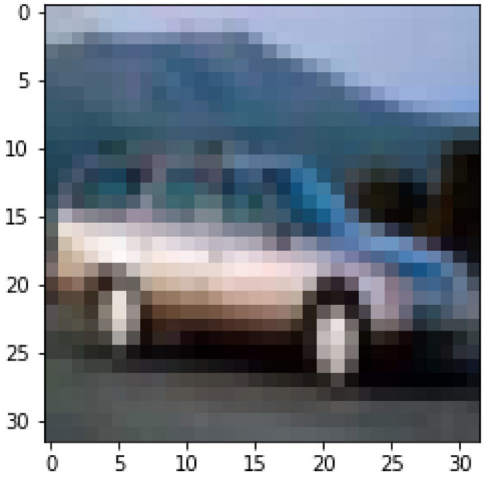
9



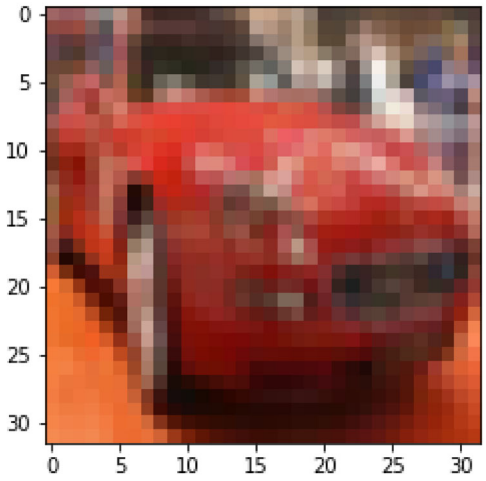
9



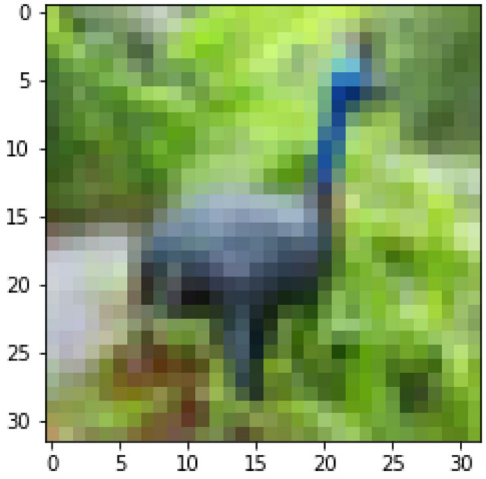
4



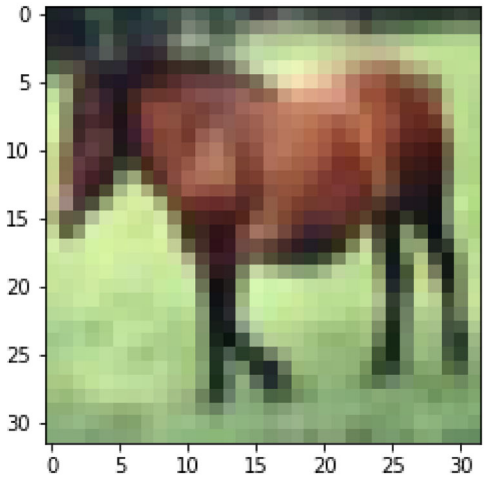
1



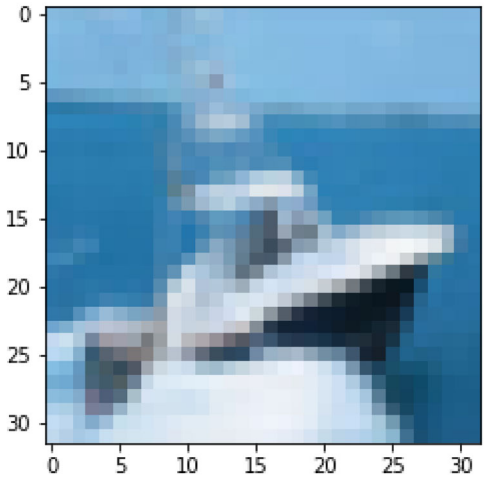
1



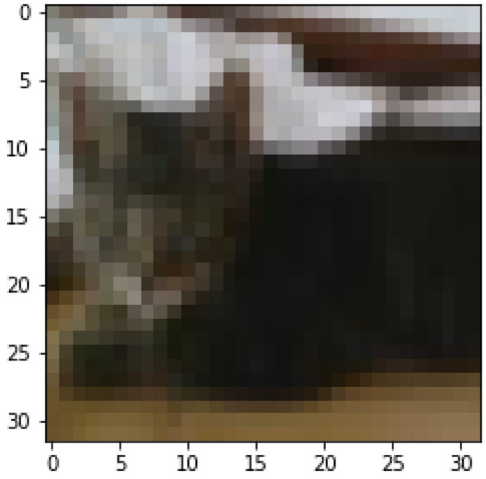
2



7

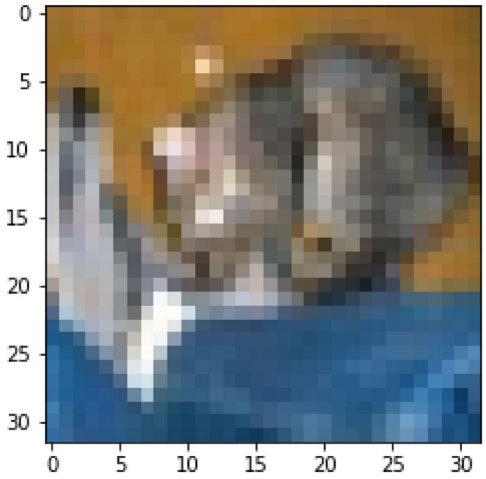


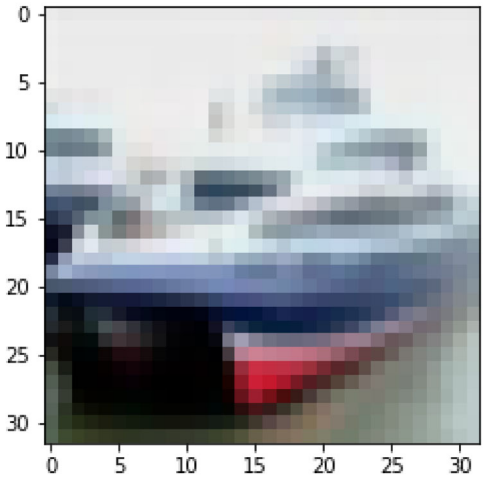
8

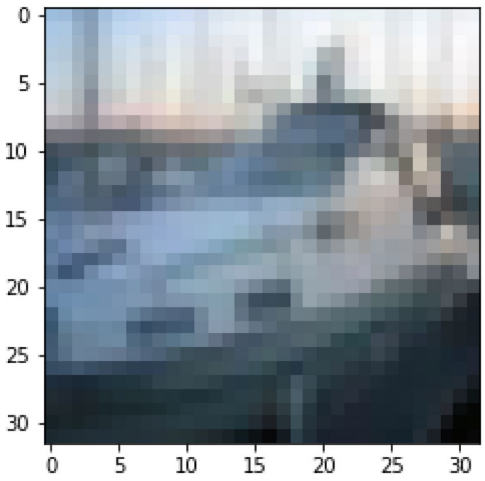


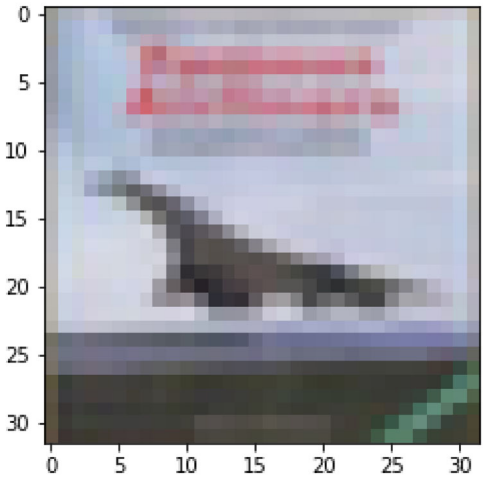
3

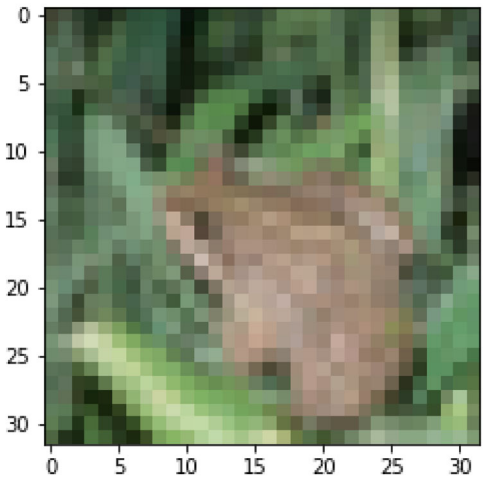
In [25]:
```
for i in range(10):
  plt.imshow(X_test[i])
  plt.show()
```

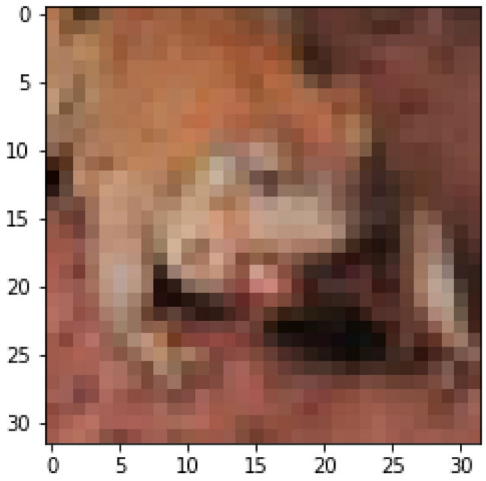


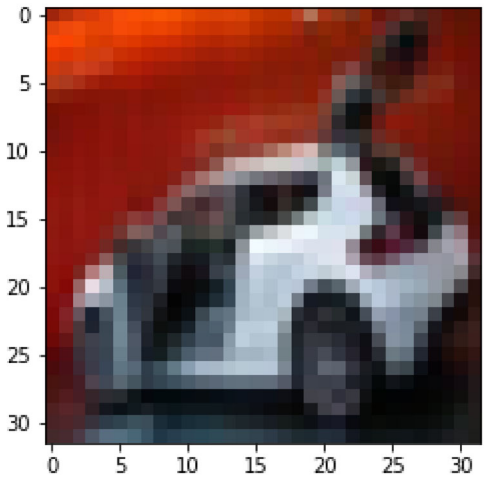

0
5
10
15
20
25
30
0 5 10 15 20 25 30

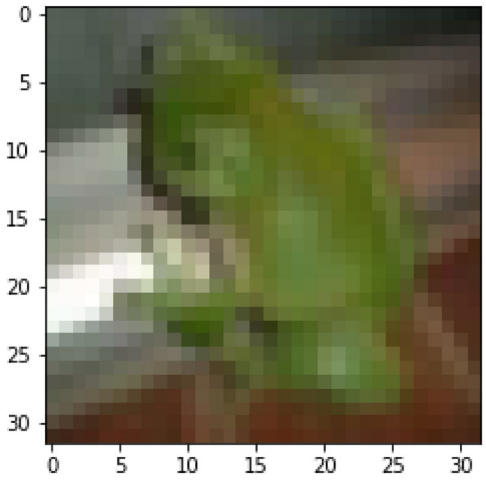

0
5
10
15
20
25
30
0 5 10 15 20 25 30

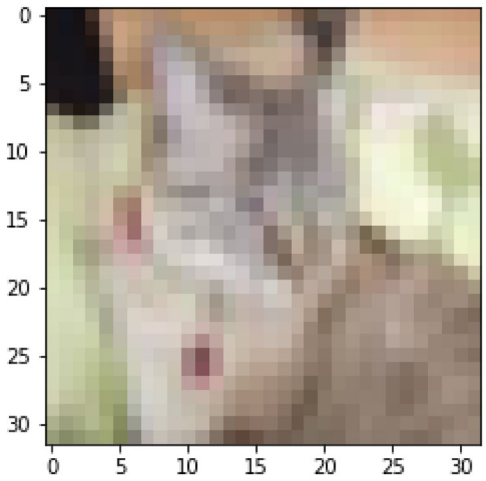

0
5
10
15
20
25
30
0 5 10 15 20 25 30

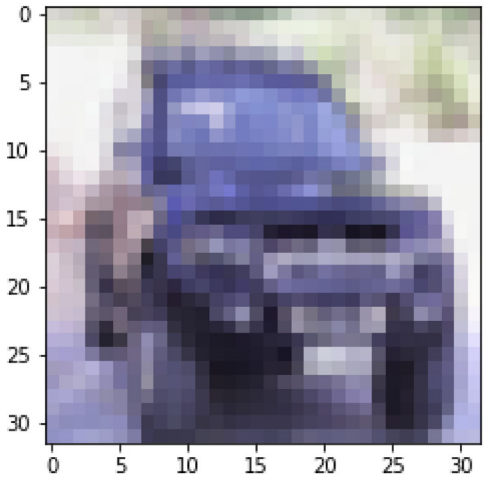

0
5
10
15
20
25
30
0 5 10 15 20 25 30


0
5
10
15
20
25
30
0 5 10 15 20 25 30


0
5
10
15
20
25
30
0 5 10 15 20 25 30


0
5
10
15
20
25
30
0 5 10 15 20 25 30


0
5
10
15
20
25
30
0 5 10 15 20 25 30



In [26]:
```
# 0->1

train_images = (np.expand_dims(X_train, axis=-1)/255.).astype(np.float32)
# train_labels = (y_train).astype(np.int64)
test_images = (np.expand_dims(X_test, axis=-1)/255.).astype(np.float32)
# test_labels = (y_test ).astype(np.int64)
```

In [27]:
```
train_images = train_images.reshape(train_images.shape[:-1])
print(train_images.shape)
```

```
(50000, 32, 32, 3)
```

In [28]:
```
test_images= test_images.reshape(test_images.shape[:-1])
print(test_images.shape)
```

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

In [29]:
```
def define_model():
        model = tf.keras.Sequential()
        model.add(tf.keras.layers.Convolution2D(32, (3, 3), activation='relu', kerne
        model.add(tf.keras.layers.Convolution2D(32, (3, 3), activation='relu', kerne
        model.add(tf.keras.layers.MaxPooling2D((2, 2)))
        model.add(tf.keras.layers.Flatten())
        model.add(tf.keras.layers.Dense(128, activation='relu', kernel_initializer='
        model.add(tf.keras.layers.Dense(10, activation='softmax'))
        # compile model
        opt = tf.keras.optimizers.SGD(lr=0.001, momentum=0.9)
        model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accu
        return model
model = define_model()
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/optimizer_v2/optimize
r_v2.py:375: UserWarning: The `lr` argument is deprecated, use `learning_rate` inste
ad.
  "The `lr` argument is deprecated, use `learning_rate` instead.")
```

In [30]:
```
from sklearn import preprocessing
lb = preprocessing.LabelBinarizer()
lb.fit(labels)
y_train = lb.transform(labels)
y_test = lb.transform(y_test_labels)
```

In [31]:
```
print(y_train)
```

```
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 1]
 [0 0 0 ... 0 0 1]
 ...
 [0 0 0 ... 0 0 1]
```

```
 [0 1 0 ... 0 0 0]
 [0 1 0 ... 0 0 0]]
```

In [32]:
```
print(y_test)
```

```
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 1 0]
 [0 0 0 ... 0 1 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 1 0 ... 0 0 0]
 [0 0 0 ... 1 0 0]]
```

In [33]:
```
print(train_images)
```

```
[[[[0.23137255 0.24313726 0.24705882]
   [0.16862746 0.18039216 0.1764706 ]
   [0.19607843 0.1882353  0.16862746]
   ...
   [0.61960787 0.5176471  0.42352942]
   [0.59607846 0.49019608 0.4       ]
   [0.5803922  0.4862745  0.40392157]]

  [[0.0627451  0.07843138 0.07843138]
   [0.         0.         0.        ]
   [0.07058824 0.03137255 0.        ]
   ...
   [0.48235294 0.34509805 0.21568628]
   [0.46666667 0.3254902  0.19607843]
   [0.47843137 0.34117648 0.22352941]]

  [[0.09803922 0.09411765 0.08235294]
   [0.0627451  0.02745098 0.        ]
   [0.19215687 0.10588235 0.03137255]
   ...
   [0.4627451  0.32941177 0.19607843]
   [0.47058824 0.32941177 0.19607843]
   [0.42745098 0.28627452 0.16470589]]

  ...

  [[0.8156863  0.6666667  0.3764706 ]
   [0.7882353  0.6        0.13333334]
   [0.7764706  0.6313726  0.10196079]
   ...
   [0.627451   0.52156866 0.27450982]
   [0.21960784 0.12156863 0.02745098]
   [0.20784314 0.13333334 0.07843138]]

  [[0.7058824  0.54509807 0.3764706 ]
   [0.6784314  0.48235294 0.16470589]
   [0.7294118  0.5647059  0.11764706]
   ...
   [0.72156864 0.5803922  0.36862746]
   [0.38039216 0.24313726 0.13333334]
   [0.3254902  0.20784314 0.13333334]]

  [[0.69411767 0.5647059  0.45490196]
   [0.65882355 0.5058824  0.36862746]
   [0.7019608  0.5568628  0.34117648]
   ...
   [0.84705883 0.72156864 0.54901963]
   [0.5921569  0.4627451  0.32941177]
   [0.48235294 0.36078432 0.28235295]]]


 [[[0.6039216  0.69411767 0.73333335]
   [0.49411765 0.5372549  0.53333336]
   [0.4117647  0.40784314 0.37254903]
```

```
  ...
  [0.35686275 0.37254903 0.2784314 ]
  [0.34117648 0.3529412  0.2784314 ]
  [0.30980393 0.31764707 0.27450982]]

 [[0.54901963 0.627451   0.6627451 ]
  [0.5686275  0.6        0.6039216 ]
  [0.49019608 0.49019608 0.4627451 ]
  ...
  [0.3764706  0.3882353  0.30588236]
  [0.3019608  0.3137255  0.24313726]
  [0.2784314  0.28627452 0.23921569]]

 [[0.54901963 0.60784316 0.6431373 ]
  [0.54509807 0.57254905 0.58431375]
  [0.4509804  0.4509804  0.4392157 ]
  ...
  [0.30980393 0.32156864 0.2509804 ]
  [0.26666668 0.27450982 0.21568628]
  [0.2627451  0.27058825 0.21568628]]

 ...

 [[0.6862745  0.654902   0.6509804 ]
  [0.6117647  0.6039216  0.627451  ]
  [0.6039216  0.627451   0.6666667 ]
  ...
  [0.16470589 0.13333334 0.14117648]
  [0.23921569 0.20784314 0.22352941]
  [0.3647059  0.3254902  0.35686275]]

 [[0.64705884 0.6039216  0.5019608 ]
  [0.6117647  0.59607846 0.50980395]
  [0.62352943 0.6313726  0.5568628 ]
  ...
  [0.40392157 0.3647059  0.3764706 ]
  [0.48235294 0.44705883 0.47058824]
  [0.5137255  0.4745098  0.5137255 ]]

 [[0.6392157  0.5803922  0.47058824]
  [0.61960787 0.5803922  0.47843137]
  [0.6392157  0.6117647  0.52156866]
  ...
  [0.56078434 0.52156866 0.54509807]
  [0.56078434 0.5254902  0.5568628 ]
  [0.56078434 0.52156866 0.5647059 ]]]


[[[1.         1.         1.        ]
  [0.99215686 0.99215686 0.99215686]
  [0.99215686 0.99215686 0.99215686]
  ...
  [0.99215686 0.99215686 0.99215686]
  [0.99215686 0.99215686 0.99215686]
  [0.99215686 0.99215686 0.99215686]]

 [[1.         1.         1.        ]
  [1.         1.         1.        ]
  [1.         1.         1.        ]
  ...
  [1.         1.         1.        ]
  [1.         1.         1.        ]
  [1.         1.         1.        ]]

 [[1.         1.         1.        ]
  [0.99607843 0.99607843 0.99607843]
  [0.99607843 0.99607843 0.99607843]
  ...
  [0.99607843 0.99607843 0.99607843]
```

```
 [0.99607843 0.99607843 0.99607843]
 [0.99607843 0.99607843 0.99607843]]

...

[[0.44313726 0.47058824 0.4392157 ]
 [0.43529412 0.4627451  0.43529412]
 [0.4117647  0.4392157  0.41568628]
 ...
 [0.28235295 0.31764707 0.3137255 ]
 [0.28235295 0.3137255  0.30980393]
 [0.28235295 0.3137255  0.30980393]]

[[0.43529412 0.4627451  0.43137255]
 [0.40784314 0.43529412 0.40784314]
 [0.3882353  0.41568628 0.38431373]
 ...
 [0.26666668 0.29411766 0.28627452]
 [0.27450982 0.29803923 0.29411766]
 [0.30588236 0.32941177 0.32156864]]

[[0.41568628 0.44313726 0.4117647 ]
 [0.3882353  0.41568628 0.38431373]
 [0.37254903 0.4        0.36862746]
 ...
 [0.30588236 0.33333334 0.3254902 ]
 [0.30980393 0.33333334 0.3254902 ]
 [0.3137255  0.3372549  0.32941177]]]


...


[[[0.13725491 0.69803923 0.92156863]
  [0.15686275 0.6901961  0.9372549 ]
  [0.16470589 0.6901961  0.94509804]
  ...
  [0.3882353  0.69411767 0.85882354]
  [0.30980393 0.5764706  0.77254903]
  [0.34901962 0.5803922  0.7411765 ]]

 [[0.22352941 0.7137255  0.91764706]
  [0.17254902 0.72156864 0.98039216]
  [0.19607843 0.7176471  0.9411765 ]
  ...
  [0.6117647  0.7137255  0.78431374]
  [0.5529412  0.69411767 0.80784315]
  [0.45490196 0.58431375 0.6862745 ]]

 [[0.38431373 0.77254903 0.92941177]
  [0.2509804  0.7411765  0.9882353 ]
  [0.27058825 0.7529412  0.9607843 ]
  ...
  [0.7372549  0.7647059  0.80784315]
  [0.46666667 0.5294118  0.5764706 ]
  [0.23921569 0.30980393 0.3529412 ]]

 ...

 [[0.28627452 0.30980393 0.3019608 ]
  [0.20784314 0.24705882 0.26666668]
  [0.21176471 0.26666668 0.3137255 ]
  ...
  [0.06666667 0.15686275 0.2509804 ]
  [0.08235294 0.14117648 0.2       ]
  [0.12941177 0.1882353  0.19215687]]

 [[0.23921569 0.26666668 0.29411766]
  [0.21568628 0.27450982 0.3372549 ]
```

```
  [0.22352941 0.30980393 0.40392157]
  ...
  [0.09411765 0.1882353  0.28235295]
  [0.06666667 0.13725491 0.20784314]
  [0.02745098 0.09019608 0.1254902 ]]

 [[0.17254902 0.21960784 0.28627452]
  [0.18039216 0.25882354 0.34509805]
  [0.19215687 0.3019608  0.4117647 ]
  ...
  [0.10588235 0.20392157 0.3019608 ]
  [0.08235294 0.16862746 0.25882354]
  [0.04705882 0.12156863 0.19607843]]]


[[[0.7411765  0.827451   0.9411765 ]
  [0.7294118  0.8156863  0.9254902 ]
  [0.7254902  0.8117647  0.92156863]
  ...
  [0.6862745  0.7647059  0.8784314 ]
  [0.6745098  0.7607843  0.87058824]
  [0.6627451  0.7607843  0.8627451 ]]

 [[0.7607843  0.8235294  0.9372549 ]
  [0.7490196  0.8117647  0.9254902 ]
  [0.74509805 0.80784315 0.92156863]
  ...
  [0.6784314  0.7529412  0.8627451 ]
  [0.67058825 0.7490196  0.85490197]
  [0.654902   0.74509805 0.84705883]]

 [[0.8156863  0.85882354 0.95686275]
  [0.8039216  0.84705883 0.9411765 ]
  [0.8        0.84313726 0.9372549 ]
  ...
  [0.6862745  0.7490196  0.8509804 ]
  [0.6745098  0.74509805 0.84705883]
  [0.6627451  0.7490196  0.84313726]]

 ...

 [[0.8117647  0.78039217 0.70980394]
  [0.79607844 0.7647059  0.6862745 ]
  [0.79607844 0.76862746 0.6784314 ]
  ...
  [0.5294118  0.5176471  0.49803922]
  [0.63529414 0.61960787 0.5882353 ]
  [0.65882355 0.6392157  0.5921569 ]]

 [[0.7764706  0.74509805 0.6666667 ]
  [0.7411765  0.70980394 0.62352943]
  [0.7058824  0.6745098  0.5764706 ]
  ...
  [0.69803923 0.67058825 0.627451  ]
  [0.6862745  0.6627451  0.6117647 ]
  [0.6862745  0.6627451  0.6039216 ]]

 [[0.7764706  0.7411765  0.6784314 ]
  [0.7411765  0.70980394 0.63529414]
  [0.69803923 0.6666667  0.58431375]
  ...
  [0.7647059  0.72156864 0.6627451 ]
  [0.76862746 0.7411765  0.67058825]
  [0.7647059  0.74509805 0.67058825]]]


[[[0.8980392  0.8980392  0.9372549 ]
  [0.9254902  0.92941177 0.96862745]
  [0.91764706 0.9254902  0.96862745]
```

```
  ...
  [0.8509804  0.85882354 0.9137255 ]
  [0.8666667  0.8745098  0.91764706]
  [0.87058824 0.8745098  0.9137255 ]]

 [[0.87058824 0.8666667  0.8980392 ]
  [0.9372549  0.9372549  0.9764706 ]
  [0.9137255  0.91764706 0.9647059 ]
  ...
  [0.8745098  0.8745098  0.9254902 ]
  [0.8901961  0.89411765 0.93333334]
  [0.8235294  0.827451   0.8627451 ]]

 [[0.8352941  0.80784315 0.827451  ]
  [0.91764706 0.9098039  0.9372549 ]
  [0.90588236 0.9137255  0.95686275]
  ...
  [0.8627451  0.8627451  0.9098039 ]
  [0.8627451  0.85882354 0.9098039 ]
  [0.7921569  0.79607844 0.84313726]]

 ...

 [[0.5882353  0.56078434 0.5294118 ]
  [0.54901963 0.5294118  0.49803922]
  [0.5176471  0.49803922 0.47058824]
  ...
  [0.8784314  0.87058824 0.85490197]
  [0.9019608  0.89411765 0.88235295]
  [0.94509804 0.94509804 0.93333334]]

 [[0.5372549  0.5176471  0.49411765]
  [0.50980395 0.49803922 0.47058824]
  [0.49019608 0.4745098  0.4509804 ]
  ...
  [0.70980394 0.7058824  0.69803923]
  [0.7921569  0.7882353  0.7764706 ]
  [0.83137256 0.827451   0.8117647 ]]

 [[0.47843137 0.46666667 0.44705883]
  [0.4627451  0.45490196 0.43137255]
  [0.47058824 0.45490196 0.43529412]
  ...
  [0.7019608  0.69411767 0.6784314 ]
  [0.6431373  0.6431373  0.63529414]
  [0.6392157  0.6392157  0.6313726 ]]]
```

In [34]:
```
test_images.shape
```

Out[34]:
```
(10000, 32, 32, 3)
```

In [35]:
```
y_test.shape
```

Out[35]:
```
(10000, 10)
```

# Variable

- X_train : train_images
- y_train : np.array(y_train)
- X_test : test_images
- y_test : y_test

In [36]:
```
# Define the batch size and the number of epochs to use during training
BATCH_SIZE = 100 #
```

```
EPOCHS = 40

'''TODO: Use model.fit to train the CNN model, with the same batch_size and number o
H = model.fit(train_images,np.array(y_train), validation_data=(test_images , y_test)
```

```
Epoch 1/40
500/500 [==============================] - 3s 5ms/step - loss: 1.8241 - accuracy: 0.
3541 - val_loss: 1.6231 - val_accuracy: 0.4260
Epoch 2/40
500/500 [==============================] - 2s 4ms/step - loss: 1.4876 - accuracy: 0.
4704 - val_loss: 1.4034 - val_accuracy: 0.4980
Epoch 3/40
500/500 [==============================] - 2s 4ms/step - loss: 1.3434 - accuracy: 0.
5236 - val_loss: 1.3056 - val_accuracy: 0.5352
Epoch 4/40
500/500 [==============================] - 2s 4ms/step - loss: 1.2596 - accuracy: 0.
5570 - val_loss: 1.2358 - val_accuracy: 0.5609
Epoch 5/40
500/500 [==============================] - 2s 4ms/step - loss: 1.1949 - accuracy: 0.
5819 - val_loss: 1.1951 - val_accuracy: 0.5792
Epoch 6/40
500/500 [==============================] - 2s 4ms/step - loss: 1.1379 - accuracy: 0.
6023 - val_loss: 1.1488 - val_accuracy: 0.5934
Epoch 7/40
500/500 [==============================] - 2s 4ms/step - loss: 1.0909 - accuracy: 0.
6173 - val_loss: 1.1368 - val_accuracy: 0.5979
Epoch 8/40
500/500 [==============================] - 2s 4ms/step - loss: 1.0409 - accuracy: 0.
6372 - val_loss: 1.1131 - val_accuracy: 0.6038
Epoch 9/40
500/500 [==============================] - 2s 4ms/step - loss: 1.0010 - accuracy: 0.
6509 - val_loss: 1.0758 - val_accuracy: 0.6210
Epoch 10/40
500/500 [==============================] - 2s 4ms/step - loss: 0.9599 - accuracy: 0.
6656 - val_loss: 1.0502 - val_accuracy: 0.6350
Epoch 11/40
500/500 [==============================] - 2s 4ms/step - loss: 0.9253 - accuracy: 0.
6779 - val_loss: 1.0476 - val_accuracy: 0.6320
Epoch 12/40
500/500 [==============================] - 2s 4ms/step - loss: 0.8821 - accuracy: 0.
6935 - val_loss: 1.0280 - val_accuracy: 0.6428
Epoch 13/40
500/500 [==============================] - 2s 4ms/step - loss: 0.8483 - accuracy: 0.
7064 - val_loss: 0.9994 - val_accuracy: 0.6517
Epoch 14/40
500/500 [==============================] - 2s 4ms/step - loss: 0.8103 - accuracy: 0.
7199 - val_loss: 1.0018 - val_accuracy: 0.6518
Epoch 15/40
500/500 [==============================] - 2s 4ms/step - loss: 0.7788 - accuracy: 0.
7321 - val_loss: 1.0101 - val_accuracy: 0.6517
Epoch 16/40
500/500 [==============================] - 2s 4ms/step - loss: 0.7508 - accuracy: 0.
7397 - val_loss: 0.9822 - val_accuracy: 0.6624
Epoch 17/40
500/500 [==============================] - 2s 4ms/step - loss: 0.7111 - accuracy: 0.
7553 - val_loss: 0.9730 - val_accuracy: 0.6678
Epoch 18/40
500/500 [==============================] - 2s 4ms/step - loss: 0.6863 - accuracy: 0.
7648 - val_loss: 0.9715 - val_accuracy: 0.6707
Epoch 19/40
500/500 [==============================] - 2s 4ms/step - loss: 0.6481 - accuracy: 0.
7772 - val_loss: 0.9816 - val_accuracy: 0.6681
Epoch 20/40
500/500 [==============================] - 2s 4ms/step - loss: 0.6203 - accuracy: 0.
7881 - val_loss: 1.0344 - val_accuracy: 0.6583
Epoch 21/40
500/500 [==============================] - 2s 4ms/step - loss: 0.5873 - accuracy: 0.
7996 - val_loss: 0.9935 - val_accuracy: 0.6690
Epoch 22/40
```

```
500/500 [==============================] - 2s 4ms/step - loss: 0.5597 - accuracy: 0.
8107 - val_loss: 0.9906 - val_accuracy: 0.6756
Epoch 23/40
500/500 [==============================] - 2s 4ms/step - loss: 0.5290 - accuracy: 0.
8211 - val_loss: 1.0260 - val_accuracy: 0.6678
Epoch 24/40
500/500 [==============================] - 2s 4ms/step - loss: 0.4953 - accuracy: 0.
8353 - val_loss: 1.0491 - val_accuracy: 0.6690
Epoch 25/40
500/500 [==============================] - 2s 4ms/step - loss: 0.4727 - accuracy: 0.
8422 - val_loss: 1.0432 - val_accuracy: 0.6710
Epoch 26/40
500/500 [==============================] - 2s 4ms/step - loss: 0.4403 - accuracy: 0.
8551 - val_loss: 1.0740 - val_accuracy: 0.6645
Epoch 27/40
500/500 [==============================] - 2s 4ms/step - loss: 0.4133 - accuracy: 0.
8657 - val_loss: 1.0871 - val_accuracy: 0.6633
Epoch 28/40
500/500 [==============================] - 2s 4ms/step - loss: 0.3806 - accuracy: 0.
8772 - val_loss: 1.1226 - val_accuracy: 0.6617
Epoch 29/40
500/500 [==============================] - 2s 4ms/step - loss: 0.3525 - accuracy: 0.
8861 - val_loss: 1.1696 - val_accuracy: 0.6600
Epoch 30/40
500/500 [==============================] - 2s 4ms/step - loss: 0.3272 - accuracy: 0.
8974 - val_loss: 1.1639 - val_accuracy: 0.6665
Epoch 31/40
500/500 [==============================] - 2s 5ms/step - loss: 0.2977 - accuracy: 0.
9078 - val_loss: 1.2233 - val_accuracy: 0.6606
Epoch 32/40
500/500 [==============================] - 2s 4ms/step - loss: 0.2705 - accuracy: 0.
9176 - val_loss: 1.2534 - val_accuracy: 0.6573
Epoch 33/40
500/500 [==============================] - 2s 4ms/step - loss: 0.2477 - accuracy: 0.
9260 - val_loss: 1.2951 - val_accuracy: 0.6588
Epoch 34/40
500/500 [==============================] - 2s 4ms/step - loss: 0.2276 - accuracy: 0.
9319 - val_loss: 1.3184 - val_accuracy: 0.6623
Epoch 35/40
500/500 [==============================] - 2s 4ms/step - loss: 0.1988 - accuracy: 0.
9438 - val_loss: 1.3586 - val_accuracy: 0.6609
Epoch 36/40
500/500 [==============================] - 2s 4ms/step - loss: 0.1770 - accuracy: 0.
9516 - val_loss: 1.3908 - val_accuracy: 0.6621
Epoch 37/40
500/500 [==============================] - 2s 4ms/step - loss: 0.1581 - accuracy: 0.
9578 - val_loss: 1.4702 - val_accuracy: 0.6539
Epoch 38/40
500/500 [==============================] - 2s 4ms/step - loss: 0.1425 - accuracy: 0.
9630 - val_loss: 1.4726 - val_accuracy: 0.6580
Epoch 39/40
500/500 [==============================] - 2s 4ms/step - loss: 0.1224 - accuracy: 0.
9711 - val_loss: 1.5212 - val_accuracy: 0.6575
Epoch 40/40
500/500 [==============================] - 2s 4ms/step - loss: 0.1077 - accuracy: 0.
9761 - val_loss: 1.5499 - val_accuracy: 0.6596
```

# Evaluate accuracy on the test dataset

In [37]:

```
test_loss, test_acc = model.evaluate(test_images , y_test)# TODO

print('Test accuracy:', test_acc)
```

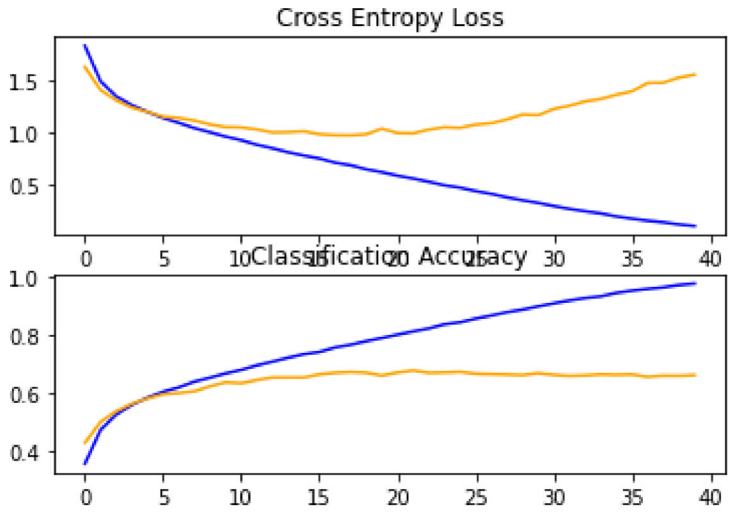
```
313/313 [==============================] - 1s 3ms/step - loss: 1.5499 - accuracy: 0.
6596
Test accuracy: 0.659600019454956
```

In [38]:
```
import sys
def summarize_diagnostics(history):
        # plot loss
        plt.subplot(211)
        plt.title('Cross Entropy Loss')
        plt.plot(history.history['loss'], color='blue', label='train')
        plt.plot(history.history['val_loss'], color='orange', label='test')
        # plot accuracy
        plt.subplot(212)
        plt.title('Classification Accuracy')
        plt.plot(history.history['accuracy'], color='blue', label='train')
        plt.plot(history.history['val_accuracy'], color='orange', label='test')
        plt.show()
```

In [39]:
```
print(H.history)
```

```
{'loss': [1.8241266012191772, 1.4876279830932617, 1.343402624130249, 1.2595564126968
384, 1.1949219703674316, 1.1378787755966187, 1.0909010171890259, 1.0409411191940308,
1.001021146774292, 0.9599418640136719, 0.925335705280304, 0.8820815086364746, 0.8483
354449272156, 0.810336709022522, 0.7787525653839111, 0.7507606148719788, 0.711111903
1906128, 0.6862810254096985, 0.6480662822723389, 0.6203339099884033, 0.5872994661331
177, 0.559696614742279, 0.5289755463600159, 0.4953438937664032, 0.4726831614971161,
0.44025900959968567, 0.41333696246147156, 0.38063371181488037, 0.35253390669822693,
0.32717305421829224, 0.2977215647697449, 0.270526260137558, 0.2476857602596283, 0.22
7588951587677, 0.19875311851501465, 0.1770431250333786, 0.1581188440322876, 0.142541
15521907806, 0.12238401174545288, 0.10766598582267761], 'accuracy': [0.3540599942207
3364, 0.4703800082206726, 0.523639976978302, 0.5569999814033508, 0.5819000005722046,
0.6022599935531616, 0.6173200011253357, 0.6372399926185608, 0.6509400010108948, 0.66
56200289726257, 0.6778600215911865, 0.6935399770736694, 0.7063800096511841, 0.719860
0172996521, 0.7320799827575684, 0.7397400140762329, 0.7553399801254272, 0.7648199796
676636, 0.7771999835968018, 0.7880799770355225, 0.7996399998664856, 0.81072002649307
25, 0.8210999965667725, 0.835319995880127, 0.8422200083732605, 0.855139970779419, 0.
8657000064849854, 0.8771600127220154, 0.8860999941825867, 0.897379994392395, 0.90784
00135040283, 0.917639970779419, 0.9259799718856812, 0.9319000244140625, 0.9437999725
341797, 0.9515799880027771, 0.9577599763870239, 0.9629999995231628, 0.97111999988555
91, 0.9760599732398987], 'val_loss': [1.623051643371582, 1.4033665657043457, 1.30556
10656738281, 1.2357871532440186, 1.1950500011444092, 1.1488419771194458, 1.136832952
4993896, 1.1130682229995728, 1.0757778882980347, 1.0502017736434937, 1.0475515127182
007, 1.027968406677246, 0.9993643760681152, 1.0018354654312134, 1.0100650787353516,
0.9822205901145935, 0.9730322957038879, 0.9715232253074646, 0.9816243052482605, 1.03
43992710113525, 0.9935475587844849, 0.9906429648399353, 1.0260231494903564, 1.049094
2001342773, 1.0431628227233887, 1.074043869972229, 1.0871212482452393, 1.12264847755
43213, 1.1696045398712158, 1.1638693809509277, 1.223250150680542, 1.253430604934692
4, 1.2951337099075317, 1.3184103965759277, 1.3586204051971436, 1.390756368637085, 1.
4702092409133911, 1.4726176261901855, 1.5211753845214844, 1.549892783164978], 'val_a
ccuracy': [0.4259999990463257, 0.49799999594688416, 0.5351999998092651, 0.5608999729
156494, 0.579200029373169, 0.5934000015258789, 0.5978999733924866, 0.603799998760223
4, 0.6209999918937683, 0.6349999904632568, 0.6320000290870667, 0.642799973487854, 0.
6517000198364258, 0.6517999768257141, 0.6517000198364258, 0.6624000072479248, 0.6678
000092506409, 0.6707000136375427, 0.6680999994277954, 0.65829998254776, 0.6690000295
639038, 0.675599992275238, 0.6678000092506409, 0.6690000295639038, 0.671000003814697
3, 0.6644999980926514, 0.6632999777793884, 0.6617000102996826, 0.6600000262260437,
0.6664999723434448, 0.6606000065803528, 0.6572999954223633, 0.6588000059127808, 0.66
22999990653991 7, 0.6608999967575073, 0.6621000170707703, 0.6539000272750854, 0.657999
9923706055, 0.6575000286102295, 0.659600019454956]}
```

In [40]:
```
summarize_diagnostics(H)
```




# reference

https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/

In [41]:
```
# define cnn model
def define_model2():
        model = tf.keras.Sequential()
        model.add(tf.keras.layers.Convolution2D(32, (3, 3), activation='relu', kerne
        model.add(tf.keras.layers.Convolution2D(32, (3, 3), activation='relu', kerne
        model.add(tf.keras.layers.MaxPooling2D((2, 2)))
        model.add(tf.keras.layers.Dropout(0.2))
        model.add(tf.keras.layers.Convolution2D(64, (3, 3), activation='relu', kerne
        model.add(tf.keras.layers.Convolution2D(64, (3, 3), activation='relu', kerne
        model.add(tf.keras.layers.MaxPooling2D((2, 2)))
        model.add(tf.keras.layers.Dropout(0.2))
        model.add(tf.keras.layers.Convolution2D(128, (3, 3), activation='relu', kern
        model.add(tf.keras.layers.Convolution2D(128, (3, 3), activation='relu', kern
        model.add(tf.keras.layers.MaxPooling2D((2, 2)))
        model.add(tf.keras.layers.Dropout(0.2))
        model.add(tf.keras.layers.Flatten())
        model.add(tf.keras.layers.Dense(128, activation='relu', kernel_initializer='
        model.add(tf.keras.layers.Dropout(0.2))
        model.add(tf.keras.layers.Dense(10, activation='softmax'))
        # compile model
        opt = tf.keras.optimizers.SGD(lr=0.001, momentum=0.9)
        model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accu
        return model
```

In [42]:
```
model2 = define_model2()
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/optimizer_v2/optimize
r_v2.py:375: UserWarning: The `lr` argument is deprecated, use `learning_rate` inste
ad.
  "The `lr` argument is deprecated, use `learning_rate` instead.")
```

In [43]:
```
BATCH_SIZE = 100 #
EPOCHS = 40

'''TODO: Use model.fit to train the CNN model, with the same batch_size and number o
H = model2.fit(train_images,np.array(y_train), validation_data=(test_images , y_test
```

```
Epoch 1/40
500/500 [==============================] - 4s 7ms/step - loss: 2.0894 - accuracy: 0.
2239 - val_loss: 1.8062 - val_accuracy: 0.3672
Epoch 2/40
```

```
500/500 [==============================] - 4s 7ms/step - loss: 1.7696 - accuracy: 0.
3501 - val_loss: 1.6139 - val_accuracy: 0.4131
Epoch 3/40
500/500 [==============================] - 4s 7ms/step - loss: 1.6299 - accuracy: 0.
4024 - val_loss: 1.4927 - val_accuracy: 0.4573
Epoch 4/40
500/500 [==============================] - 4s 7ms/step - loss: 1.5320 - accuracy: 0.
4385 - val_loss: 1.4038 - val_accuracy: 0.4933
Epoch 5/40
500/500 [==============================] - 4s 7ms/step - loss: 1.4709 - accuracy: 0.
4601 - val_loss: 1.3716 - val_accuracy: 0.5079
Epoch 6/40
500/500 [==============================] - 4s 7ms/step - loss: 1.4003 - accuracy: 0.
4880 - val_loss: 1.2995 - val_accuracy: 0.5288
Epoch 7/40
500/500 [==============================] - 4s 7ms/step - loss: 1.3449 - accuracy: 0.
5112 - val_loss: 1.2440 - val_accuracy: 0.5522
Epoch 8/40
500/500 [==============================] - 4s 7ms/step - loss: 1.2944 - accuracy: 0.
5309 - val_loss: 1.2388 - val_accuracy: 0.5566
Epoch 9/40
500/500 [==============================] - 4s 7ms/step - loss: 1.2482 - accuracy: 0.
5496 - val_loss: 1.1324 - val_accuracy: 0.5959
Epoch 10/40
500/500 [==============================] - 4s 7ms/step - loss: 1.1982 - accuracy: 0.
5724 - val_loss: 1.0941 - val_accuracy: 0.6098
Epoch 11/40
500/500 [==============================] - 4s 7ms/step - loss: 1.1582 - accuracy: 0.
5833 - val_loss: 1.0665 - val_accuracy: 0.6172
Epoch 12/40
500/500 [==============================] - 4s 7ms/step - loss: 1.1245 - accuracy: 0.
5974 - val_loss: 1.0010 - val_accuracy: 0.6423
Epoch 13/40
500/500 [==============================] - 4s 7ms/step - loss: 1.0941 - accuracy: 0.
6096 - val_loss: 0.9900 - val_accuracy: 0.6506
Epoch 14/40
500/500 [==============================] - 4s 7ms/step - loss: 1.0624 - accuracy: 0.
6216 - val_loss: 0.9531 - val_accuracy: 0.6643
Epoch 15/40
500/500 [==============================] - 4s 7ms/step - loss: 1.0301 - accuracy: 0.
6342 - val_loss: 0.9251 - val_accuracy: 0.6740
Epoch 16/40
500/500 [==============================] - 4s 7ms/step - loss: 1.0062 - accuracy: 0.
6416 - val_loss: 0.9171 - val_accuracy: 0.6764
Epoch 17/40
500/500 [==============================] - 4s 7ms/step - loss: 0.9801 - accuracy: 0.
6517 - val_loss: 0.9035 - val_accuracy: 0.6776
Epoch 18/40
500/500 [==============================] - 4s 7ms/step - loss: 0.9582 - accuracy: 0.
6587 - val_loss: 0.8643 - val_accuracy: 0.6957
Epoch 19/40
500/500 [==============================] - 4s 7ms/step - loss: 0.9360 - accuracy: 0.
6685 - val_loss: 0.8467 - val_accuracy: 0.7018
Epoch 20/40
500/500 [==============================] - 4s 7ms/step - loss: 0.9147 - accuracy: 0.
6766 - val_loss: 0.8420 - val_accuracy: 0.7035
Epoch 21/40
500/500 [==============================] - 4s 7ms/step - loss: 0.8927 - accuracy: 0.
6846 - val_loss: 0.8094 - val_accuracy: 0.7180
Epoch 22/40
500/500 [==============================] - 4s 7ms/step - loss: 0.8773 - accuracy: 0.
6901 - val_loss: 0.7961 - val_accuracy: 0.7223
Epoch 23/40
500/500 [==============================] - 4s 7ms/step - loss: 0.8570 - accuracy: 0.
6978 - val_loss: 0.7980 - val_accuracy: 0.7239
Epoch 24/40
500/500 [==============================] - 4s 7ms/step - loss: 0.8423 - accuracy: 0.
7029 - val_loss: 0.7731 - val_accuracy: 0.7325
Epoch 25/40
```

```
500/500 [==============================] - 4s 7ms/step - loss: 0.8280 - accuracy: 0.
7078 - val_loss: 0.7639 - val_accuracy: 0.7332
Epoch 26/40
500/500 [==============================] - 4s 7ms/step - loss: 0.8096 - accuracy: 0.
7145 - val_loss: 0.7530 - val_accuracy: 0.7366
Epoch 27/40
500/500 [==============================] - 4s 7ms/step - loss: 0.8009 - accuracy: 0.
7178 - val_loss: 0.7472 - val_accuracy: 0.7411
Epoch 28/40
500/500 [==============================] - 4s 7ms/step - loss: 0.7834 - accuracy: 0.
7241 - val_loss: 0.7358 - val_accuracy: 0.7456
Epoch 29/40
500/500 [==============================] - 4s 7ms/step - loss: 0.7698 - accuracy: 0.
7296 - val_loss: 0.7307 - val_accuracy: 0.7462
Epoch 30/40
500/500 [==============================] - 4s 7ms/step - loss: 0.7615 - accuracy: 0.
7319 - val_loss: 0.7132 - val_accuracy: 0.7520
Epoch 31/40
500/500 [==============================] - 4s 7ms/step - loss: 0.7519 - accuracy: 0.
7365 - val_loss: 0.7328 - val_accuracy: 0.7470
Epoch 32/40
500/500 [==============================] - 4s 7ms/step - loss: 0.7315 - accuracy: 0.
7426 - val_loss: 0.7010 - val_accuracy: 0.7589
Epoch 33/40
500/500 [==============================] - 4s 7ms/step - loss: 0.7223 - accuracy: 0.
7453 - val_loss: 0.6956 - val_accuracy: 0.7567
Epoch 34/40
500/500 [==============================] - 4s 7ms/step - loss: 0.7188 - accuracy: 0.
7477 - val_loss: 0.6992 - val_accuracy: 0.7601
Epoch 35/40
500/500 [==============================] - 4s 7ms/step - loss: 0.7042 - accuracy: 0.
7521 - val_loss: 0.6831 - val_accuracy: 0.7653
Epoch 36/40
500/500 [==============================] - 4s 7ms/step - loss: 0.6943 - accuracy: 0.
7560 - val_loss: 0.6846 - val_accuracy: 0.7631
Epoch 37/40
500/500 [==============================] - 4s 7ms/step - loss: 0.6842 - accuracy: 0.
7600 - val_loss: 0.6575 - val_accuracy: 0.7718
Epoch 38/40
500/500 [==============================] - 4s 7ms/step - loss: 0.6754 - accuracy: 0.
7627 - val_loss: 0.6557 - val_accuracy: 0.7754
Epoch 39/40
500/500 [==============================] - 4s 7ms/step - loss: 0.6631 - accuracy: 0.
7681 - val_loss: 0.6635 - val_accuracy: 0.7720
Epoch 40/40
500/500 [==============================] - 4s 7ms/step - loss: 0.6537 - accuracy: 0.
7681 - val_loss: 0.6500 - val_accuracy: 0.7752
```

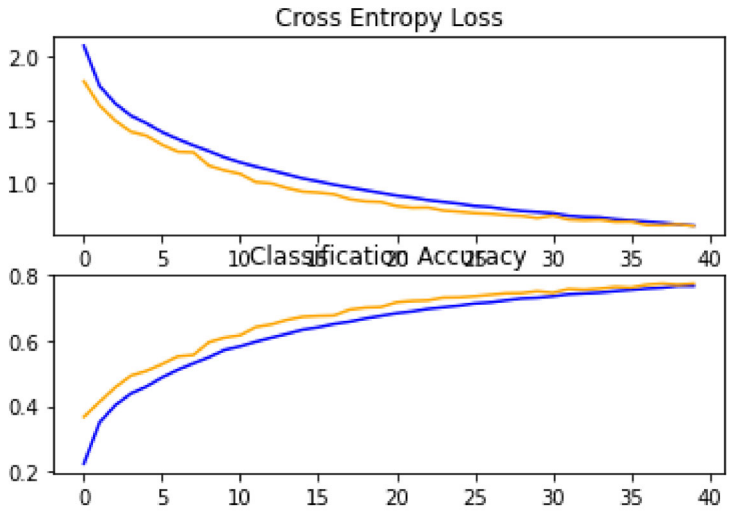
In [44]:

```
test_loss, test_acc = model2.evaluate(test_images , y_test)# TODO

print('Test accuracy:', test_acc)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.6500 - accuracy: 0.
7752
Test accuracy: 0.7752000093460083
```

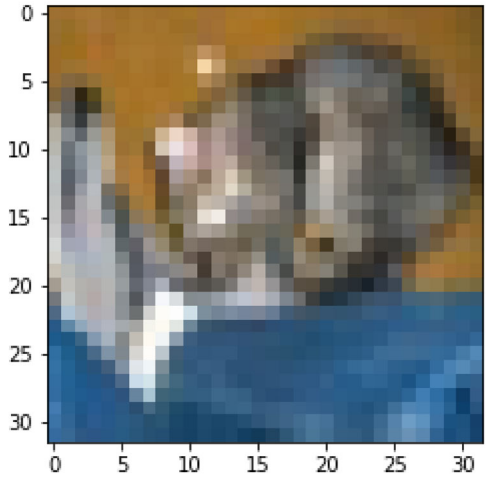
In [45]:

```
summarize_diagnostics(H)
```

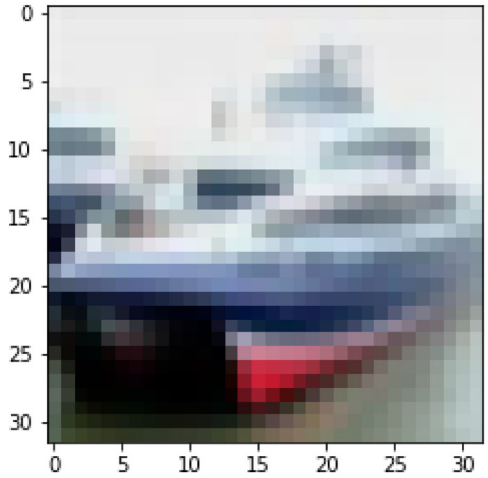



In [72]:
```
for i in range(10):
  predictions = model2.predict(np.array([test_images[i]]))
  prediction = np.argmax(predictions[0])# TODO
  print(f"Class image :  {prediction +1}")
  print("Label of this digit is:", y_test[i])
  plt.imshow(test_images[i,:,:], cmap=plt.cm.binary)
  plt.show()
```

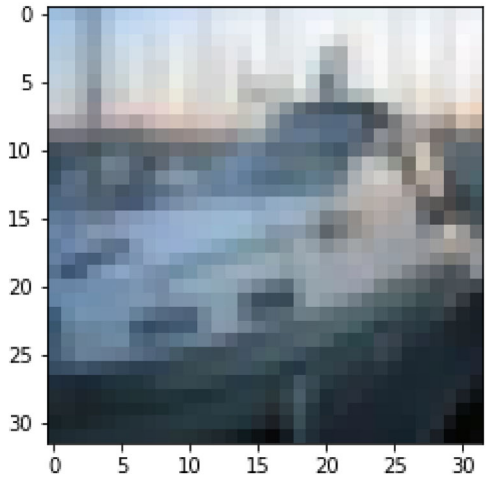
```
Class image :  4
Label of this digit is: [0 0 0 1 0 0 0 0 0 0]
```



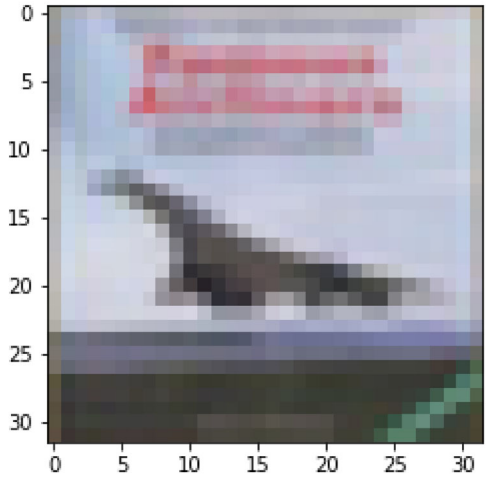
```
Class image :  9
Label of this digit is: [0 0 0 0 0 0 0 0 1 0]
```



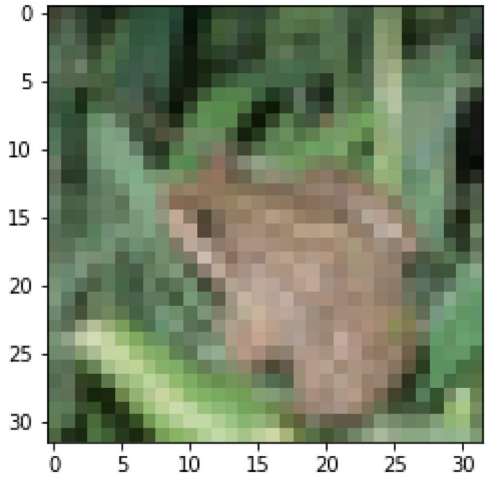
```
Class image :  9
Label of this digit is: [0 0 0 0 0 0 0 0 1 0]
```



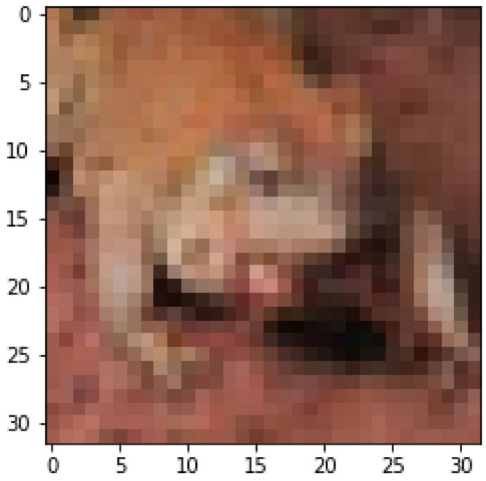
```
Class image :  1
Label of this digit is: [1 0 0 0 0 0 0 0 0 0]
```



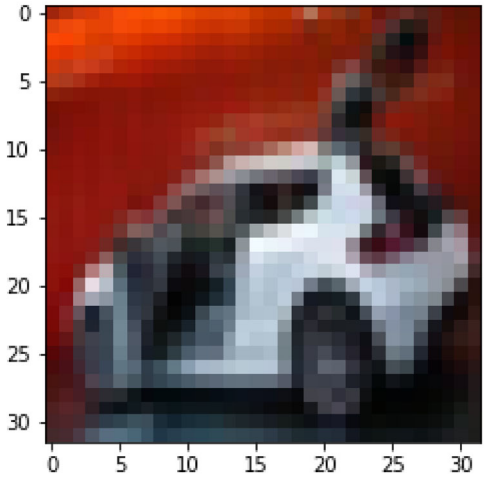
```
Class image :  7
Label of this digit is: [0 0 0 0 0 0 1 0 0 0]
```



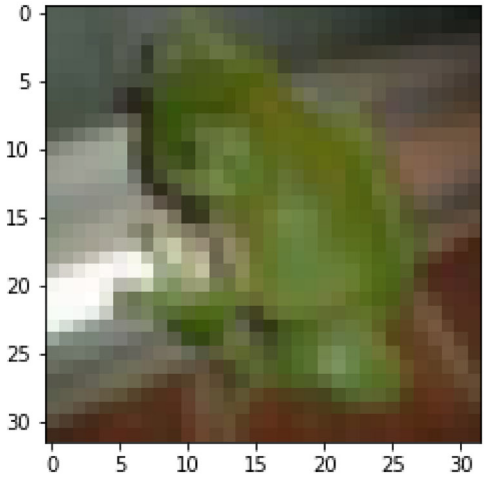
```
Class image :  7
Label of this digit is: [0 0 0 0 0 0 1 0 0 0]
```



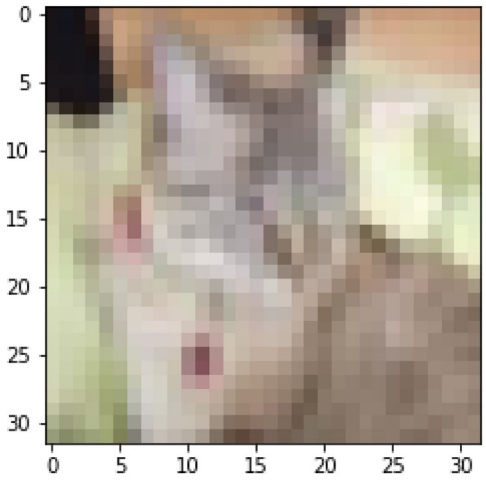
```
Class image :  2
Label of this digit is: [0 1 0 0 0 0 0 0 0 0]
```



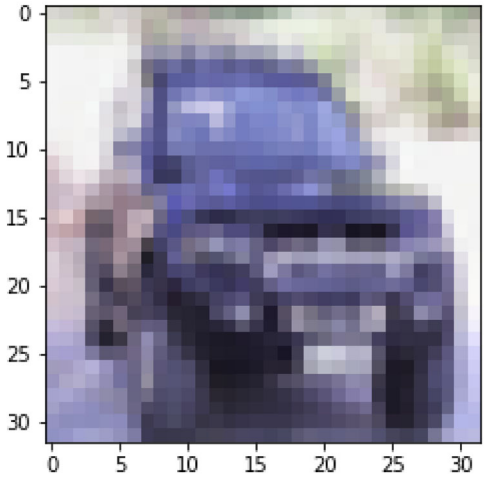
```
Class image :  7
Label of this digit is: [0 0 0 0 0 0 1 0 0 0]
```



```
Class image :  4
Label of this digit is: [0 0 0 1 0 0 0 0 0 0]
```



```
Class image :  2
Label of this digit is: [0 1 0 0 0 0 0 0 0 0]
```



In [62]:
```
# predictions = model2.predict(np.array([test_images[2]]))
```

In [63]:
```
# predictions
```

Out[63]:
```
array([[8.0983527e-03, 6.2280190e-03, 9.2255854e-05, 1.2325000e-03,
        3.0086128e-05, 2.9635085e-05, 1.0467141e-05, 7.0712020e-05,
        9.7792393e-01, 6.2839459e-03]], dtype=float32)
```

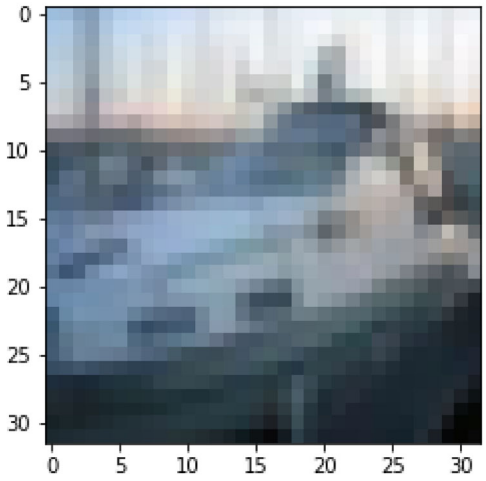
In [64]:
```
# prediction = np.argmax(predictions[0])# TODO
# print(f"Class image :  {prediction +1}")
```

```
Class image :  9
```

In [65]:
```
# print("Label of this digit is:", y_test[2])
# plt.imshow(test_images[2,:,:], cmap=plt.cm.binary)
```

```
Label of this digit is: [0 0 0 0 0 0 0 0 1 0]
```

Out[65]:
```
<matplotlib.image.AxesImage at 0x7f3625ee0890>
```




In [74]:
```
from tensorflow.python.client import device_lib
device_lib.list_local_devices()
```

Out[74]:
```
[name: "/device:CPU:0"
 device_type: "CPU"
 memory_limit: 268435456
 locality {
 }
 incarnation: 11052314551703897330, name: "/device:GPU:0"
 device_type: "GPU"
 memory_limit: 16183459840
 locality {
   bus_id: 1
   links {
   }
 }
 incarnation: 10008767861492315563
 physical_device_desc: "device: 0, name: Tesla P100-PCIE-16GB, pci bus id: 0000:00:0
4.0, compute capability: 6.0"]
```