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')
```

```
Mounted at /content/drive
```

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-batch
  # 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]:
```
len(pictures) , len(labels)
```

Out[5]: (50000, 50000)

In [6]:
```
pictures.shape
```

Out[6]: (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 [7]:
```
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 [8]:
```
np.array(dog_img).shape , np.array(cat_img).shape
```

Out[8]:
```
((5000, 3072), (5000, 3072))
```

In [9]:
```
img_all = [cat_img ,dog_img]
# np.array(img_all).shape
d_c_img = np.concatenate(np.array(img_all),axis = 0)
```

In [10]:
```
d_c_img.shape
```

Out[10]:
```
(10000, 3072)
```

In [11]:
```
labels_all = [cat_labels,dog_labels]
d_c_label = np.concatenate(np.array(labels_all),axis=0)
```

In [12]:
```
d_c_label.shape
```

Out[12]:
```
(10000,)
```

import test_batch

In [13]:
```
test = "/content/drive/MyDrive/Colab Notebooks/Lab 3/dataset/cifar-10-batches
y_test = unpickle(test)
y_test_data = y_test[b"data"]
y_test_labels = y_test[b"labels"]
```

In [14]:
```
len(y_test_data) , len(y_test_labels)
```

Out[14]:
```
(10000, 10000)
```

In [15]:

```
# 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 [16]:

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(random_state=42,max_iter=1000).fit(d_c_img, d_c_labe
y_pred = clf.predict(dc_img_test)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940:
ConvergenceWarning: 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-regres
sion
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

In [17]:

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

```
Accuracy: 0.556
```

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

load dataset

- X_train : pictures
- y_train : labels
- X_test : y_test_data
- y_test : y_test_labels

In [18]:

```
pictures.shape
```

Out[18]: (50000, 3072)

In [19]:

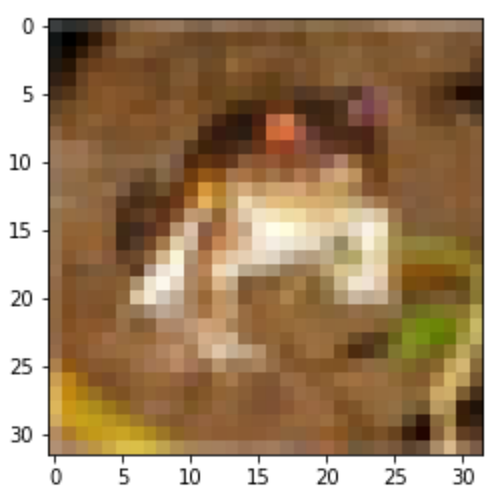
```
X_train = []
X_test = []
for i in range(len(pictures)):
  img = np.reshape(pictures[i], (3,32,32))
  X_train.append(np.transpose(img, (1, 2, 0)))
print(X_train[0].shape)

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
```

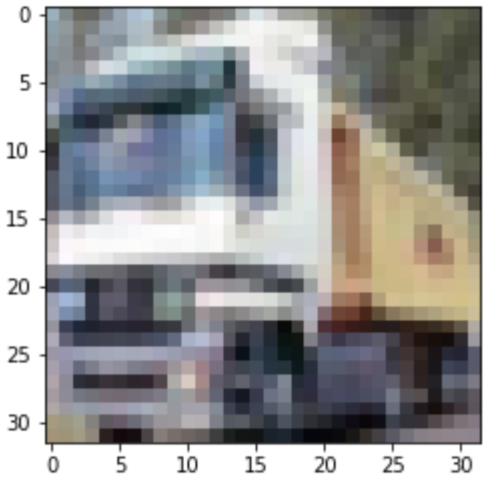
(32, 32, 3)

In [20]:

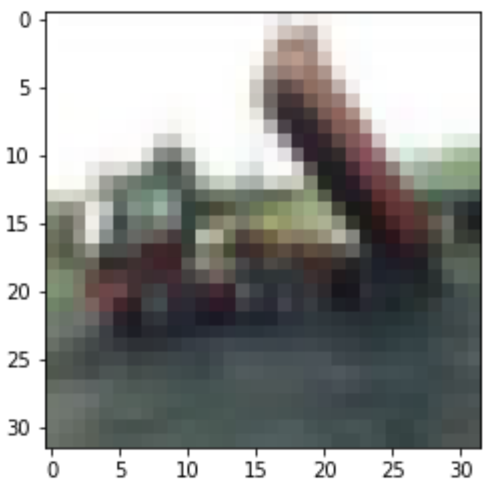
```
import matplotlib.pyplot as plt
for i in range(10):
  plt.imshow(X_train[i])
  plt.show()
  print(y_train[i])
```



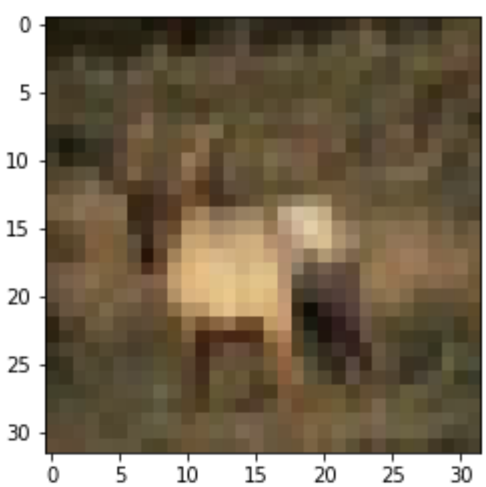
6


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

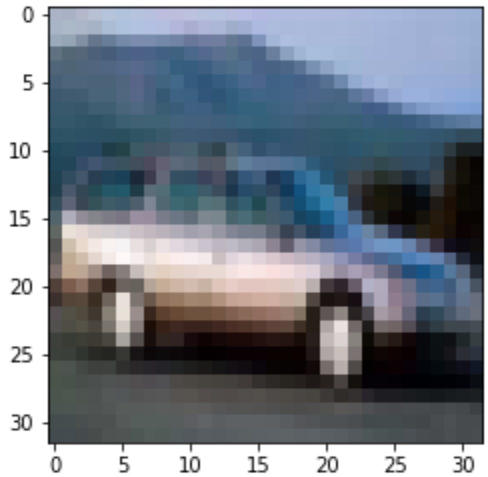
9


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

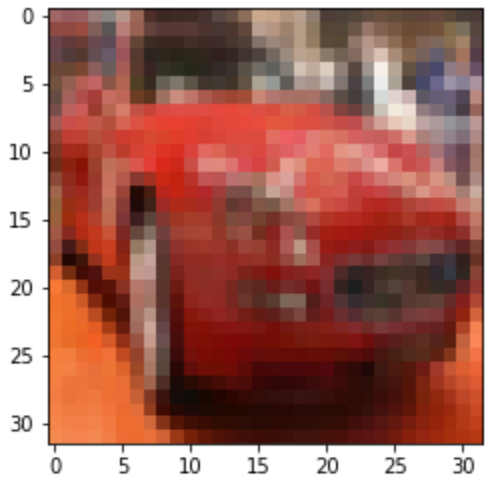
9


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

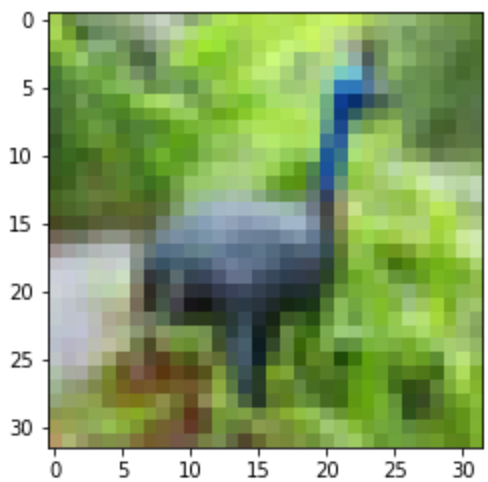
4


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

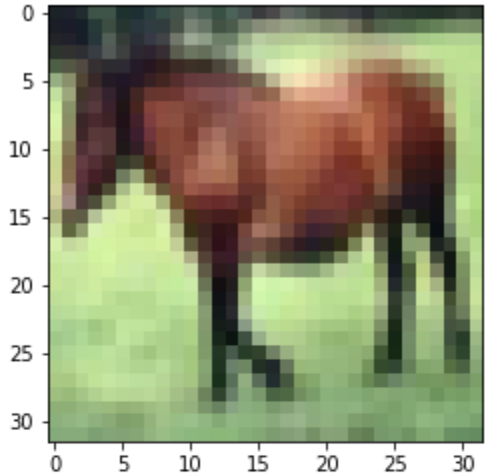
1


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

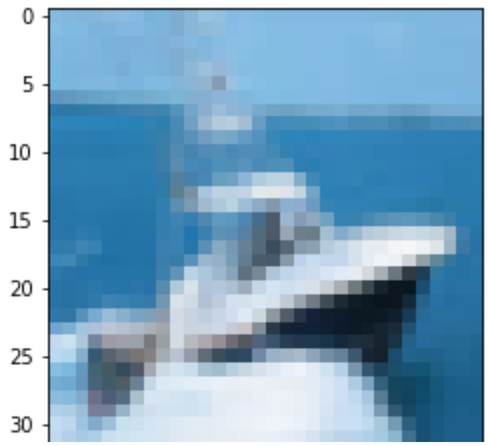
1


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

2

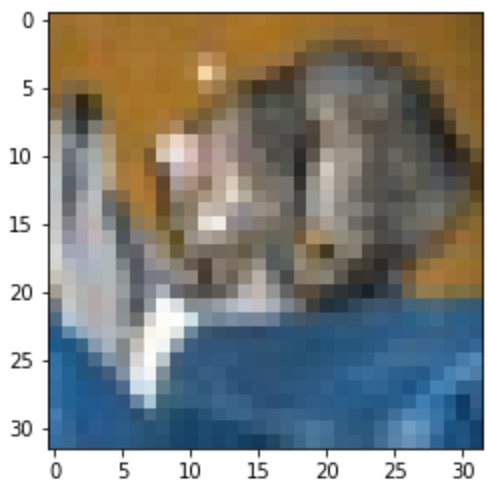


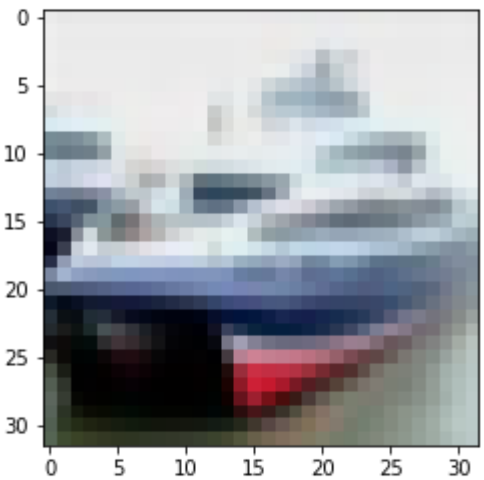
7

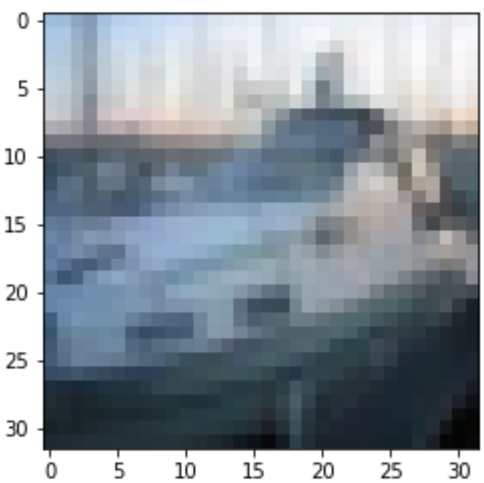


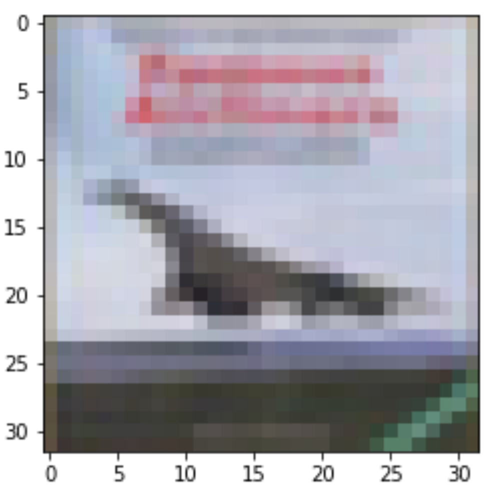
In [21]:

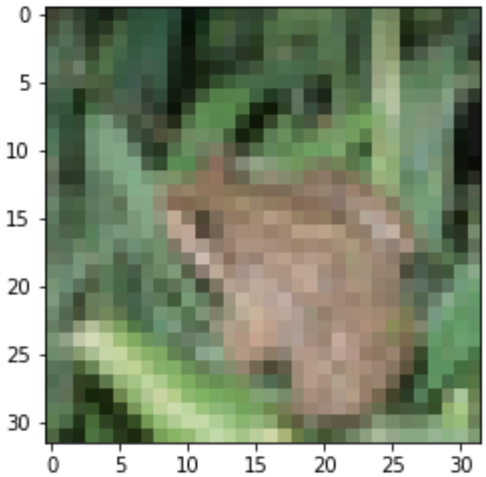
```
for i in range(10):
  plt.imshow(X_test[i])
  plt.show()
```

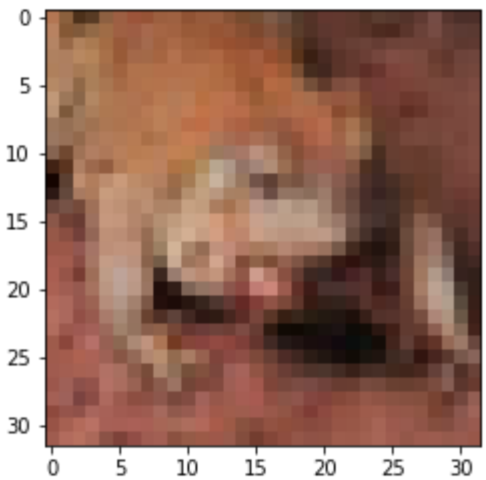


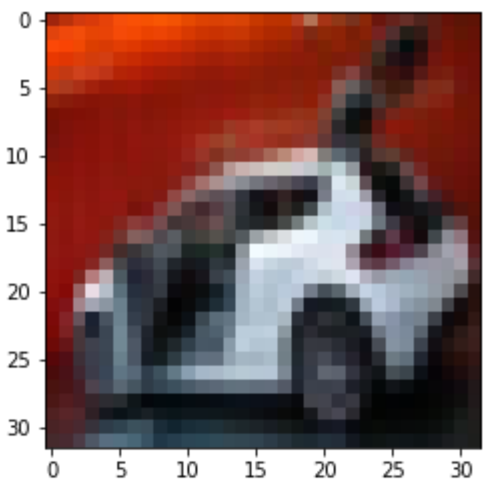

0
5
10
15
20
25
30
0 5 10 15 20 25 30

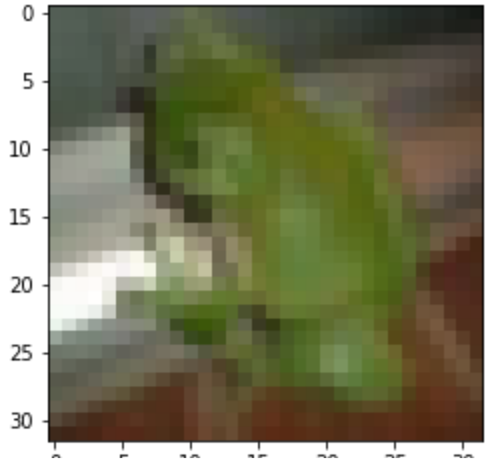

0
5
10
15
20
25
30
0 5 10 15 20 25 30


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









In [22]:
```
# 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 [23]:
```
train_images = train_images.reshape(train_images.shape[:-1])
print(train_images.shape)
```

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

In [24]:
```
test_images= test_images.reshape(test_images.shape[:-1])
print(test_images.shape)
```

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

Model1

In [25]:
```
def define_model():
        model = tf.keras.Sequential()
        model.add(tf.keras.layers.Convolution2D(32, (3, 3), activation='relu'
        model.add(tf.keras.layers.Convolution2D(32, (3, 3), activation='relu'
        model.add(tf.keras.layers.MaxPooling2D((2, 2)))
        model.add(tf.keras.layers.Flatten())
        model.add(tf.keras.layers.Dense(128, activation='relu', kernel_initia
        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
        return model
model = define_model()
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/optimizer_v2/o
ptimizer_v2.py:375: UserWarning: The `lr` argument is deprecated, use `learni
ng_rate` instead.
  "The `lr` argument is deprecated, use `learning_rate` instead.")
```

In [26]:
```
from sklearn import preprocessing
lb = preprocessing.LabelBinarizer()
lb.fit(labels)
y_train = lb.transform(labels)
y_test = lb.transform(y_test_labels)
```

In [27]:
```
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 [28]:
```
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 [30]:
```
# print(train_images)
```

In [31]:
```
test_images.shape
```

Out[31]:
```
(10000, 32, 32, 3)
```

In [32]:
```
y_test.shape
```

Out[32]:
```
(10000, 10)
```

Variable

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

In [33]:
```
# 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 n
H = model.fit(train_images,np.array(y_train), validation_data=(test_images ,
```

```
Epoch 1/40
500/500 [==============================] - 17s 5ms/step - loss: 1.8627 - accu
racy: 0.3410 - val_loss: 1.6043 - val_accuracy: 0.4343
Epoch 2/40
500/500 [==============================] - 2s 4ms/step - loss: 1.4989 - accur
acy: 0.4722 - val_loss: 1.3904 - val_accuracy: 0.5090
Epoch 3/40
500/500 [==============================] - 2s 4ms/step - loss: 1.3424 - accur
acy: 0.5258 - val_loss: 1.2958 - val_accuracy: 0.5358
Epoch 4/40
500/500 [==============================] - 2s 4ms/step - loss: 1.2416 - accur
acy: 0.5637 - val_loss: 1.2555 - val_accuracy: 0.5550
Epoch 5/40
500/500 [==============================] - 2s 4ms/step - loss: 1.1707 - accur
acy: 0.5919 - val_loss: 1.1741 - val_accuracy: 0.5929
Epoch 6/40
500/500 [==============================] - 2s 4ms/step - loss: 1.1100 - accur
acy: 0.6126 - val_loss: 1.1447 - val_accuracy: 0.5960
Epoch 7/40
500/500 [==============================] - 2s 4ms/step - loss: 1.0526 - accur
acy: 0.6339 - val_loss: 1.0920 - val_accuracy: 0.6204
Epoch 8/40
500/500 [==============================] - 2s 4ms/step - loss: 1.0069 - accur
acy: 0.6514 - val_loss: 1.0814 - val_accuracy: 0.6219
Epoch 9/40
500/500 [==============================] - 2s 4ms/step - loss: 0.9588 - accur
acy: 0.6676 - val_loss: 1.0560 - val_accuracy: 0.6387
Epoch 10/40
500/500 [==============================] - 2s 4ms/step - loss: 0.9256 - accur
acy: 0.6796 - val_loss: 1.0694 - val_accuracy: 0.6282
Epoch 11/40
500/500 [==============================] - 2s 4ms/step - loss: 0.8863 - accur
acy: 0.6929 - val_loss: 1.0186 - val_accuracy: 0.6474
Epoch 12/40
500/500 [==============================] - 2s 4ms/step - loss: 0.8563 - accur
acy: 0.7032 - val_loss: 1.0166 - val_accuracy: 0.6487
Epoch 13/40
500/500 [==============================] - 2s 4ms/step - loss: 0.8170 - accur
acy: 0.7183 - val_loss: 1.0079 - val_accuracy: 0.6517
Epoch 14/40
500/500 [==============================] - 2s 4ms/step - loss: 0.7860 - accur
acy: 0.7309 - val_loss: 0.9886 - val_accuracy: 0.6612
Epoch 15/40
500/500 [==============================] - 2s 4ms/step - loss: 0.7527 - accur
acy: 0.7427 - val_loss: 0.9942 - val_accuracy: 0.6599
Epoch 16/40
500/500 [==============================] - 2s 4ms/step - loss: 0.7249 - accur
acy: 0.7527 - val_loss: 1.0112 - val_accuracy: 0.6581
Epoch 17/40
500/500 [==============================] - 2s 4ms/step - loss: 0.6962 - accur
acy: 0.7619 - val_loss: 0.9804 - val_accuracy: 0.6670
```

```
Epoch 18/40
500/500 [==============================] - 2s 4ms/step - loss: 0.6623 - accur
acy: 0.7744 - val_loss: 0.9949 - val_accuracy: 0.6590
Epoch 19/40
500/500 [==============================] - 2s 4ms/step - loss: 0.6389 - accur
acy: 0.7815 - val_loss: 0.9907 - val_accuracy: 0.6617
Epoch 20/40
500/500 [==============================] - 2s 4ms/step - loss: 0.6051 - accur
acy: 0.7957 - val_loss: 1.0058 - val_accuracy: 0.6657
Epoch 21/40
500/500 [==============================] - 2s 4ms/step - loss: 0.5760 - accur
acy: 0.8055 - val_loss: 0.9981 - val_accuracy: 0.6684
Epoch 22/40
500/500 [==============================] - 2s 4ms/step - loss: 0.5434 - accur
acy: 0.8172 - val_loss: 1.0526 - val_accuracy: 0.6628
Epoch 23/40
500/500 [==============================] - 2s 4ms/step - loss: 0.5163 - accur
acy: 0.8287 - val_loss: 1.0369 - val_accuracy: 0.6616
Epoch 24/40
500/500 [==============================] - 2s 4ms/step - loss: 0.4841 - accur
acy: 0.8418 - val_loss: 1.0573 - val_accuracy: 0.6638
Epoch 25/40
500/500 [==============================] - 2s 4ms/step - loss: 0.4571 - accur
acy: 0.8508 - val_loss: 1.0498 - val_accuracy: 0.6668
Epoch 26/40
500/500 [==============================] - 2s 4ms/step - loss: 0.4281 - accur
acy: 0.8619 - val_loss: 1.0801 - val_accuracy: 0.6640
Epoch 27/40
500/500 [==============================] - 2s 4ms/step - loss: 0.3998 - accur
acy: 0.8710 - val_loss: 1.0963 - val_accuracy: 0.6667
Epoch 28/40
500/500 [==============================] - 2s 4ms/step - loss: 0.3683 - accur
acy: 0.8828 - val_loss: 1.1060 - val_accuracy: 0.6688
Epoch 29/40
500/500 [==============================] - 2s 4ms/step - loss: 0.3407 - accur
acy: 0.8934 - val_loss: 1.1222 - val_accuracy: 0.6703
Epoch 30/40
500/500 [==============================] - 2s 4ms/step - loss: 0.3120 - accur
acy: 0.9042 - val_loss: 1.1603 - val_accuracy: 0.6652
Epoch 31/40
500/500 [==============================] - 2s 4ms/step - loss: 0.2869 - accur
acy: 0.9127 - val_loss: 1.1879 - val_accuracy: 0.6646
Epoch 32/40
500/500 [==============================] - 2s 4ms/step - loss: 0.2571 - accur
acy: 0.9246 - val_loss: 1.2191 - val_accuracy: 0.6725
Epoch 33/40
500/500 [==============================] - 2s 4ms/step - loss: 0.2366 - accur
acy: 0.9321 - val_loss: 1.2637 - val_accuracy: 0.6669
Epoch 34/40
500/500 [==============================] - 2s 4ms/step - loss: 0.2126 - accur
acy: 0.9392 - val_loss: 1.2864 - val_accuracy: 0.6635
Epoch 35/40
500/500 [==============================] - 2s 4ms/step - loss: 0.1940 - accur
acy: 0.9468 - val_loss: 1.3412 - val_accuracy: 0.6675
Epoch 36/40
500/500 [==============================] - 2s 4ms/step - loss: 0.1665 - accur
acy: 0.9569 - val_loss: 1.3798 - val_accuracy: 0.6568
Epoch 37/40
500/500 [==============================] - 2s 4ms/step - loss: 0.1494 - accur
acy: 0.9639 - val_loss: 1.4530 - val_accuracy: 0.6561
```

```
Epoch 38/40
500/500 [==============================] - 2s 4ms/step - loss: 0.1304 - accur
acy: 0.9698 - val_loss: 1.4720 - val_accuracy: 0.6572
Epoch 39/40
500/500 [==============================] - 2s 4ms/step - loss: 0.1130 - accur
acy: 0.9765 - val_loss: 1.4845 - val_accuracy: 0.6583
Epoch 40/40
500/500 [==============================] - 2s 4ms/step - loss: 0.0969 - accur
acy: 0.9814 - val_loss: 1.5777 - val_accuracy: 0.6547
```

Evaluate accuracy on the test dataset

In [34]:

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

print('Test accuracy:', test_acc) # จะเห็นว่าพอ wvaluate ออกมา accuracy ต่ำมากเห
```

```
313/313 [==============================] - 1s 2ms/step - loss: 1.5777 - accur
acy: 0.6547
Test accuracy: 0.654699981212616
```

In [45]:

```python
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 [46]:

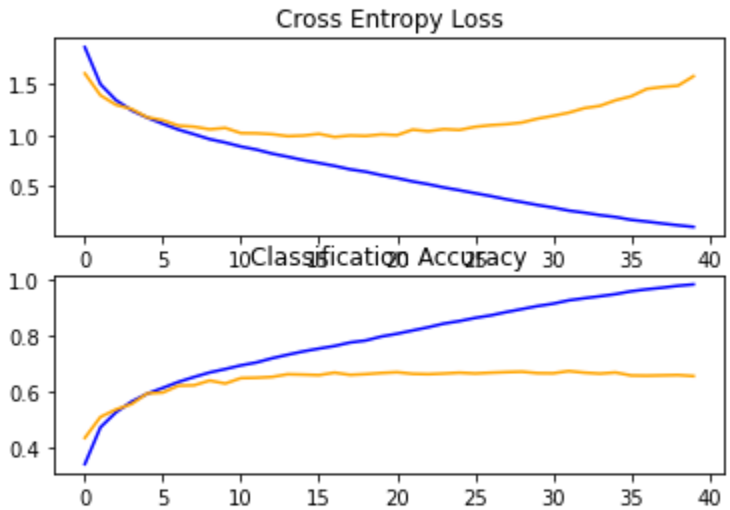
```python
print(H.history)
```

{'loss': [1.8626582622528076, 1.4989218711853027, 1.342417597770691, 1.241615
2954101562, 1.170661449432373, 1.1100300550460815, 1.0526463985443115, 1.0069
069862365723, 0.95875084400177, 0.9255715608596802, 0.8863099217414856, 0.856
3295602798462, 0.8169576525688171, 0.7860075831413269, 0.7526900172233582, 0.
724940836429596, 0.6962292790412903, 0.6622699499130249, 0.6389214992523193,
0.605068564414978, 0.5760378837585449, 0.543362021446228, 0.5162547826766968,
0.484083354473114, 0.45708608627319336, 0.4280926287174225, 0.399846762418746
95, 0.3683443069458008, 0.34072160720825195, 0.31204962730407715, 0.286935210
2279663, 0.25708726048469543, 0.2366359531879425, 0.21259021759033203, 0.1939
8343563079834, 0.16649742424488068, 0.14943167567253113, 0.13040457665920258,
0.11295243352651596, 0.09688997268676758], 'accuracy': [0.34097999334335327,
0.47218000888824463, 0.5258399844169617, 0.5636600255966187, 0.59193998575210
57, 0.6126000285148621, 0.6338800191879272, 0.6513599753379822, 0.66759997606
27747, 0.679639995098114, 0.6929000020027161, 0.7032399773597717, 0.718259990
2153015, 0.7308800220489502, 0.7427200078964233, 0.7527199983596802, 0.761940
0024414062, 0.7744399905204773, 0.7814599871635437, 0.7957000136375427, 0.805
5199980735779, 0.8172199726104736, 0.8286799788475037, 0.841759979724884, 0.8
507599830627441, 0.8619199991226196, 0.8709800243377686, 0.8828200101852417,
0.8934000134468079, 0.90420001745224, 0.9126999974250793, 0.9246199727058411,
0.9321399927139282, 0.9391800165176392, 0.9467800259590149, 0.956939995288848

9, 0.963919997215271, 0.969760000705719, 0.9765200018882751, 0.98144000768661
5], 'val_loss': [1.6043137311935425, 1.3903875350952148, 1.295823097229004,
1.2555264234542847, 1.1740790605545044, 1.1447032690048218, 1.091999650001525
9, 1.081357717514038, 1.0559580326080322, 1.0693773031234741, 1.0186221599578
857, 1.0166118144989014, 1.007931113243103, 0.9885696172714233, 0.99422878026
96228, 1.0111860036849976, 0.9804283976554871, 0.9948976635932922, 0.99067127
70462036, 1.0058215856552124, 0.9980943202972412, 1.0526105165481567, 1.03690
4215812683, 1.0572537183761597, 1.0497946739196777, 1.0801067352294922, 1.096
2709188461304, 1.1059794425964355, 1.1222352981567383, 1.1603283882141113, 1.
187942385673523, 1.2191479206085205, 1.2636942863464355, 1.2863949537277222,
1.3411788940429688, 1.379753589630127, 1.4529746770858765, 1.471972703933715
8, 1.484496831893921, 1.5776983499526978], 'val_accuracy': [0.434300005435943
6, 0.5090000033378601, 0.5357999801635742, 0.5550000071525574, 0.592899978160
8582, 0.5960000157356262, 0.6204000115394592, 0.6219000220298767, 0.638700008
392334, 0.6281999945640564, 0.6474000215530396, 0.6486999988555908, 0.6517000
198364258, 0.6611999869346619, 0.6599000096321106, 0.6581000089645386, 0.6669
999957084656, 0.6589999794960022, 0.6617000102996826, 0.6657000184059143, 0.6
6839998960495, 0.6628000140190125, 0.6615999937057495, 0.6638000011444092, 0.
6668000221252441, 0.6639999747276306, 0.666700005531311, 0.6687999963760376,
0.6703000068664551, 0.6651999950408936, 0.6646000146865845, 0.672500014305114
7, 0.6668999791145325, 0.6635000109672546, 0.6675000190734863, 0.656799972057
3425, 0.6560999751091003, 0.6571999788284302, 0.65829998254776, 0.65469998121
2616]}

In [47]:

```
summarize_diagnostics(H)
```



Model 2

In [64]:

```
def define_model_1():
        model = tf.keras.Sequential()
        model.add(tf.keras.layers.Convolution2D(32, (3, 3), activation='relu'
        model.add(tf.keras.layers.Convolution2D(32, (3, 3), activation='relu'
        model.add(tf.keras.layers.MaxPooling2D((2, 2)))
        model.add(tf.keras.layers.Convolution2D(64,(3,3),activation = 'relu',
        model.add(tf.keras.layers.Convolution2D(64,(3,3),activation = 'relu',
        model.add(tf.keras.layers.MaxPooling2D((2, 2)))
        model.add(tf.keras.layers.Dropout(0.25)) # กัน model overfit
        model.add(tf.keras.layers.Flatten())
        model.add(tf.keras.layers.Dense(128, activation='relu', kernel_initia
        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
        return model
```

In [65]:

```
model1 = define_model_1()
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/optimizer_v2/o
ptimizer_v2.py:375: UserWarning: The `lr` argument is deprecated, use `learni
ng_rate` instead.
  "The `lr` argument is deprecated, use `learning_rate` instead.")
```

In [66]:

```
BATCH_SIZE = 100 #
EPOCHS = 40

'''TODO: Use model.fit to train the CNN model, with the same batch_size and n
H = model1.fit(train_images,np.array(y_train), validation_data=(test_images ,
```

```
Epoch 1/40
500/500 [==============================] - 3s 6ms/step - loss: 1.9100 - accur
acy: 0.3044 - val_loss: 1.5756 - val_accuracy: 0.4380
Epoch 2/40
500/500 [==============================] - 3s 6ms/step - loss: 1.5420 - accur
acy: 0.4424 - val_loss: 1.4141 - val_accuracy: 0.4917
Epoch 3/40
500/500 [==============================] - 3s 6ms/step - loss: 1.4100 - accur
acy: 0.4903 - val_loss: 1.3659 - val_accuracy: 0.5177
Epoch 4/40
500/500 [==============================] - 3s 6ms/step - loss: 1.3200 - accur
acy: 0.5301 - val_loss: 1.2732 - val_accuracy: 0.5488
Epoch 5/40
500/500 [==============================] - 3s 6ms/step - loss: 1.2404 - accur
acy: 0.5578 - val_loss: 1.2008 - val_accuracy: 0.5758
Epoch 6/40
500/500 [==============================] - 3s 6ms/step - loss: 1.1709 - accur
acy: 0.5858 - val_loss: 1.1016 - val_accuracy: 0.6146
Epoch 7/40
500/500 [==============================] - 3s 6ms/step - loss: 1.1101 - accur
acy: 0.6091 - val_loss: 1.0614 - val_accuracy: 0.6250
Epoch 8/40
500/500 [==============================] - 3s 6ms/step - loss: 1.0536 - accur
acy: 0.6292 - val_loss: 1.0214 - val_accuracy: 0.6461
Epoch 9/40
500/500 [==============================] - 3s 6ms/step - loss: 1.0041 - accur
```

```
acy: 0.6479 - val_loss: 0.9748 - val_accuracy: 0.6565
Epoch 10/40
500/500 [==============================] - 3s 6ms/step - loss: 0.9652 - accur
acy: 0.6603 - val_loss: 0.9483 - val_accuracy: 0.6663
Epoch 11/40
500/500 [==============================] - 3s 6ms/step - loss: 0.9275 - accur
acy: 0.6757 - val_loss: 0.9326 - val_accuracy: 0.6717
Epoch 12/40
500/500 [==============================] - 3s 6ms/step - loss: 0.8911 - accur
acy: 0.6879 - val_loss: 0.9048 - val_accuracy: 0.6845
Epoch 13/40
500/500 [==============================] - 3s 6ms/step - loss: 0.8610 - accur
acy: 0.7002 - val_loss: 0.8788 - val_accuracy: 0.6928
Epoch 14/40
500/500 [==============================] - 3s 6ms/step - loss: 0.8347 - accur
acy: 0.7077 - val_loss: 0.8713 - val_accuracy: 0.6959
Epoch 15/40
500/500 [==============================] - 3s 6ms/step - loss: 0.8076 - accur
acy: 0.7175 - val_loss: 0.8437 - val_accuracy: 0.7074
Epoch 16/40
500/500 [==============================] - 3s 6ms/step - loss: 0.7808 - accur
acy: 0.7287 - val_loss: 0.8251 - val_accuracy: 0.7137
Epoch 17/40
500/500 [==============================] - 3s 6ms/step - loss: 0.7632 - accur
acy: 0.7341 - val_loss: 0.8175 - val_accuracy: 0.7166
Epoch 18/40
500/500 [==============================] - 3s 6ms/step - loss: 0.7383 - accur
acy: 0.7435 - val_loss: 0.8081 - val_accuracy: 0.7178
Epoch 19/40
500/500 [==============================] - 3s 6ms/step - loss: 0.7116 - accur
acy: 0.7527 - val_loss: 0.8017 - val_accuracy: 0.7243
Epoch 20/40
500/500 [==============================] - 3s 6ms/step - loss: 0.6946 - accur
acy: 0.7597 - val_loss: 0.7890 - val_accuracy: 0.7268
Epoch 21/40
500/500 [==============================] - 3s 6ms/step - loss: 0.6727 - accur
acy: 0.7670 - val_loss: 0.7863 - val_accuracy: 0.7294
Epoch 22/40
500/500 [==============================] - 3s 6ms/step - loss: 0.6558 - accur
acy: 0.7717 - val_loss: 0.7787 - val_accuracy: 0.7316
Epoch 23/40
500/500 [==============================] - 3s 6ms/step - loss: 0.6334 - accur
acy: 0.7808 - val_loss: 0.7816 - val_accuracy: 0.7358
Epoch 24/40
500/500 [==============================] - 3s 6ms/step - loss: 0.6186 - accur
acy: 0.7850 - val_loss: 0.7666 - val_accuracy: 0.7356
Epoch 25/40
500/500 [==============================] - 3s 6ms/step - loss: 0.6055 - accur
acy: 0.7883 - val_loss: 0.7611 - val_accuracy: 0.7398
Epoch 26/40
500/500 [==============================] - 3s 6ms/step - loss: 0.5856 - accur
acy: 0.7972 - val_loss: 0.7592 - val_accuracy: 0.7415
Epoch 27/40
500/500 [==============================] - 3s 6ms/step - loss: 0.5664 - accur
acy: 0.8032 - val_loss: 0.7502 - val_accuracy: 0.7439
Epoch 28/40
500/500 [==============================] - 3s 6ms/step - loss: 0.5498 - accur
acy: 0.8072 - val_loss: 0.7423 - val_accuracy: 0.7470
Epoch 29/40
500/500 [==============================] - 3s 6ms/step - loss: 0.5366 - accur
```

```
acy: 0.8129 - val_loss: 0.7457 - val_accuracy: 0.7493
Epoch 30/40
500/500 [==============================] - 3s 6ms/step - loss: 0.5207 - accur
acy: 0.8184 - val_loss: 0.7541 - val_accuracy: 0.7428
Epoch 31/40
500/500 [==============================] - 3s 6ms/step - loss: 0.5023 - accur
acy: 0.8241 - val_loss: 0.7434 - val_accuracy: 0.7484
Epoch 32/40
500/500 [==============================] - 3s 6ms/step - loss: 0.4928 - accur
acy: 0.8282 - val_loss: 0.7369 - val_accuracy: 0.7539
Epoch 33/40
500/500 [==============================] - 3s 6ms/step - loss: 0.4783 - accur
acy: 0.8346 - val_loss: 0.7398 - val_accuracy: 0.7566
Epoch 34/40
500/500 [==============================] - 3s 6ms/step - loss: 0.4624 - accur
acy: 0.8374 - val_loss: 0.7401 - val_accuracy: 0.7523
Epoch 35/40
500/500 [==============================] - 3s 6ms/step - loss: 0.4471 - accur
acy: 0.8426 - val_loss: 0.7434 - val_accuracy: 0.7567
Epoch 36/40
500/500 [==============================] - 3s 6ms/step - loss: 0.4303 - accur
acy: 0.8495 - val_loss: 0.7517 - val_accuracy: 0.7570
Epoch 37/40
500/500 [==============================] - 3s 6ms/step - loss: 0.4184 - accur
acy: 0.8529 - val_loss: 0.7590 - val_accuracy: 0.7544
Epoch 38/40
500/500 [==============================] - 3s 6ms/step - loss: 0.4042 - accur
acy: 0.8571 - val_loss: 0.7582 - val_accuracy: 0.7565
Epoch 39/40
500/500 [==============================] - 3s 6ms/step - loss: 0.3928 - accur
acy: 0.8613 - val_loss: 0.7803 - val_accuracy: 0.7512
Epoch 40/40
500/500 [==============================] - 3s 6ms/step - loss: 0.3797 - accur
acy: 0.8668 - val_loss: 0.7494 - val_accuracy: 0.7642
```

In [67]:

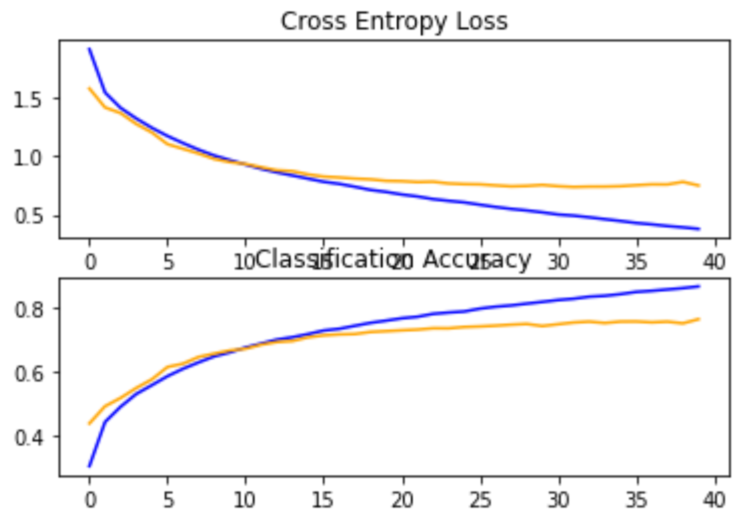
```
test_loss, test_acc = model1.evaluate(test_images , y_test)# TODO

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

```
313/313 [==============================] - 1s 3ms/step - loss: 0.7494 - accur
acy: 0.7642
Test accuracy: 0.7641999721527100
```

In [68]:

```
summarize_diagnostics(H)
```



reference

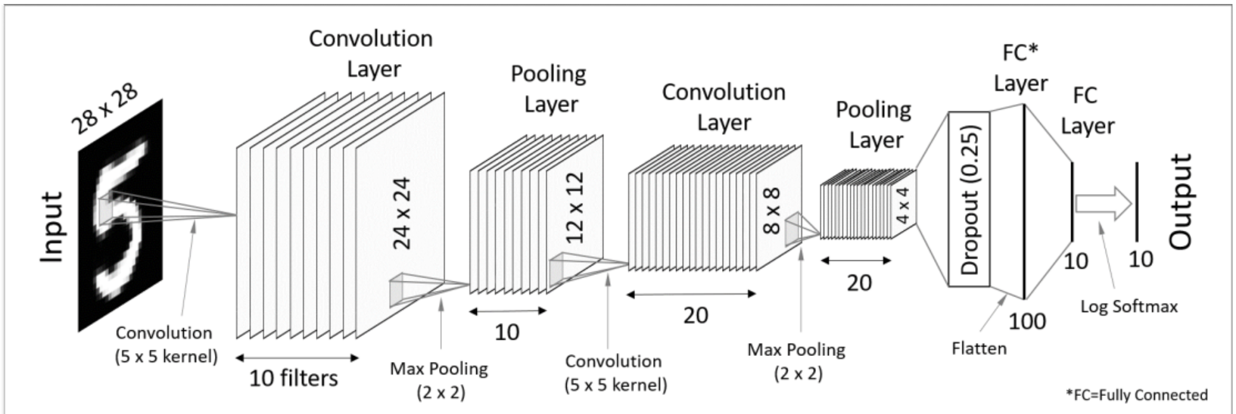
https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/

In [48]:

```
from IPython.display import Image
Image("/content/drive/MyDrive/Colab Notebooks/Lab 3/CNN network.png")

# refer : https://www.visionatics.com.tw/machine-learning-service.html
```

Out[48]:



In [73]:

```
# define cnn model
def define_model2():
        model = tf.keras.Sequential()
        model.add(tf.keras.layers.Convolution2D(32, (3, 3), activation='relu'
        model.add(tf.keras.layers.Convolution2D(32, (3, 3), activation='relu'
        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'
        model.add(tf.keras.layers.Convolution2D(64, (3, 3), activation='relu'
        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
        model.add(tf.keras.layers.Convolution2D(128, (3, 3), activation='relu
        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_initia
        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
        return model
```

In [74]:

```
model2 = define_model2()
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/optimizer_v2/o
ptimizer_v2.py:375: UserWarning: The `lr` argument is deprecated, use `learni
ng_rate` instead.
  "The `lr` argument is deprecated, use `learning_rate` instead.")
```

In [75]:

```
BATCH_SIZE = 100 #
EPOCHS = 40

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

```
Epoch 1/40
500/500 [==============================] - 4s 7ms/step - loss: 2.1596 - accur
acy: 0.1935 - val_loss: 1.8835 - val_accuracy: 0.3272
Epoch 2/40
500/500 [==============================] - 3s 7ms/step - loss: 1.7955 - accur
acy: 0.3336 - val_loss: 1.6417 - val_accuracy: 0.4101
Epoch 3/40
500/500 [==============================] - 4s 7ms/step - loss: 1.6567 - accur
acy: 0.3856 - val_loss: 1.5228 - val_accuracy: 0.4495
Epoch 4/40
500/500 [==============================] - 4s 7ms/step - loss: 1.5507 - accur
acy: 0.4280 - val_loss: 1.4301 - val_accuracy: 0.4771
Epoch 5/40
500/500 [==============================] - 4s 7ms/step - loss: 1.4722 - accur
acy: 0.4611 - val_loss: 1.3544 - val_accuracy: 0.5127
Epoch 6/40
500/500 [==============================] - 4s 7ms/step - loss: 1.4188 - accur
acy: 0.4826 - val_loss: 1.3341 - val_accuracy: 0.5166
```

```
Epoch 7/40
500/500 [==============================] - 4s 7ms/step - loss: 1.3645 - accur
acy: 0.5035 - val_loss: 1.2769 - val_accuracy: 0.5481
Epoch 8/40
500/500 [==============================] - 4s 7ms/step - loss: 1.3204 - accur
acy: 0.5204 - val_loss: 1.1931 - val_accuracy: 0.5762
Epoch 9/40
500/500 [==============================] - 4s 7ms/step - loss: 1.2699 - accur
acy: 0.5427 - val_loss: 1.1612 - val_accuracy: 0.5954
Epoch 10/40
500/500 [==============================] - 3s 7ms/step - loss: 1.2232 - accur
acy: 0.5591 - val_loss: 1.1001 - val_accuracy: 0.6124
Epoch 11/40
500/500 [==============================] - 3s 7ms/step - loss: 1.1803 - accur
acy: 0.5755 - val_loss: 1.0671 - val_accuracy: 0.6268
Epoch 12/40
500/500 [==============================] - 3s 7ms/step - loss: 1.1350 - accur
acy: 0.5917 - val_loss: 1.0655 - val_accuracy: 0.6202
Epoch 13/40
500/500 [==============================] - 3s 7ms/step - loss: 1.1039 - accur
acy: 0.6049 - val_loss: 0.9799 - val_accuracy: 0.6552
Epoch 14/40
500/500 [==============================] - 3s 7ms/step - loss: 1.0658 - accur
acy: 0.6184 - val_loss: 0.9682 - val_accuracy: 0.6615
Epoch 15/40
500/500 [==============================] - 3s 7ms/step - loss: 1.0324 - accur
acy: 0.6327 - val_loss: 0.9347 - val_accuracy: 0.6733
Epoch 16/40
500/500 [==============================] - 3s 7ms/step - loss: 1.0036 - accur
acy: 0.6419 - val_loss: 0.9136 - val_accuracy: 0.6811
Epoch 17/40
500/500 [==============================] - 3s 7ms/step - loss: 0.9768 - accur
acy: 0.6532 - val_loss: 0.8821 - val_accuracy: 0.6889
Epoch 18/40
500/500 [==============================] - 3s 7ms/step - loss: 0.9566 - accur
acy: 0.6599 - val_loss: 0.8634 - val_accuracy: 0.7006
Epoch 19/40
500/500 [==============================] - 3s 7ms/step - loss: 0.9287 - accur
acy: 0.6711 - val_loss: 0.8344 - val_accuracy: 0.7105
Epoch 20/40
500/500 [==============================] - 3s 7ms/step - loss: 0.9068 - accur
acy: 0.6792 - val_loss: 0.8180 - val_accuracy: 0.7166
Epoch 21/40
500/500 [==============================] - 3s 7ms/step - loss: 0.8843 - accur
acy: 0.6887 - val_loss: 0.8151 - val_accuracy: 0.7120
Epoch 22/40
500/500 [==============================] - 3s 7ms/step - loss: 0.8637 - accur
acy: 0.6937 - val_loss: 0.8011 - val_accuracy: 0.7232
Epoch 23/40
500/500 [==============================] - 3s 7ms/step - loss: 0.8506 - accur
acy: 0.7008 - val_loss: 0.7880 - val_accuracy: 0.7269
Epoch 24/40
500/500 [==============================] - 3s 7ms/step - loss: 0.8316 - accur
acy: 0.7058 - val_loss: 0.7648 - val_accuracy: 0.7287
Epoch 25/40
500/500 [==============================] - 3s 7ms/step - loss: 0.8183 - accur
acy: 0.7116 - val_loss: 0.7545 - val_accuracy: 0.7332
Epoch 26/40
500/500 [==============================] - 3s 7ms/step - loss: 0.8049 - accur
acy: 0.7165 - val_loss: 0.7346 - val_accuracy: 0.7449
```

```
Epoch 27/40
500/500 [==============================] - 4s 7ms/step - loss: 0.7909 - accur
acy: 0.7204 - val_loss: 0.7289 - val_accuracy: 0.7469
Epoch 28/40
500/500 [==============================] - 4s 7ms/step - loss: 0.7715 - accur
acy: 0.7273 - val_loss: 0.7309 - val_accuracy: 0.7422
Epoch 29/40
500/500 [==============================] - 4s 7ms/step - loss: 0.7655 - accur
acy: 0.7287 - val_loss: 0.7047 - val_accuracy: 0.7534
Epoch 30/40
500/500 [==============================] - 3s 7ms/step - loss: 0.7500 - accur
acy: 0.7343 - val_loss: 0.6979 - val_accuracy: 0.7572
Epoch 31/40
500/500 [==============================] - 3s 7ms/step - loss: 0.7403 - accur
acy: 0.7401 - val_loss: 0.6988 - val_accuracy: 0.7562
Epoch 32/40
500/500 [==============================] - 3s 7ms/step - loss: 0.7274 - accur
acy: 0.7430 - val_loss: 0.6899 - val_accuracy: 0.7571
Epoch 33/40
500/500 [==============================] - 3s 7ms/step - loss: 0.7183 - accur
acy: 0.7478 - val_loss: 0.6920 - val_accuracy: 0.7601
Epoch 34/40
500/500 [==============================] - 3s 7ms/step - loss: 0.7063 - accur
acy: 0.7489 - val_loss: 0.6781 - val_accuracy: 0.7624
Epoch 35/40
500/500 [==============================] - 3s 7ms/step - loss: 0.7016 - accur
acy: 0.7535 - val_loss: 0.6896 - val_accuracy: 0.7584
Epoch 36/40
500/500 [==============================] - 3s 7ms/step - loss: 0.6895 - accur
acy: 0.7556 - val_loss: 0.6668 - val_accuracy: 0.7689
Epoch 37/40
500/500 [==============================] - 3s 7ms/step - loss: 0.6801 - accur
acy: 0.7606 - val_loss: 0.6494 - val_accuracy: 0.7718
Epoch 38/40
500/500 [==============================] - 3s 7ms/step - loss: 0.6706 - accur
acy: 0.7630 - val_loss: 0.6771 - val_accuracy: 0.7634
Epoch 39/40
500/500 [==============================] - 3s 7ms/step - loss: 0.6623 - accur
acy: 0.7662 - val_loss: 0.6562 - val_accuracy: 0.7696
Epoch 40/40
500/500 [==============================] - 4s 7ms/step - loss: 0.6546 - accur
```

In [76]:

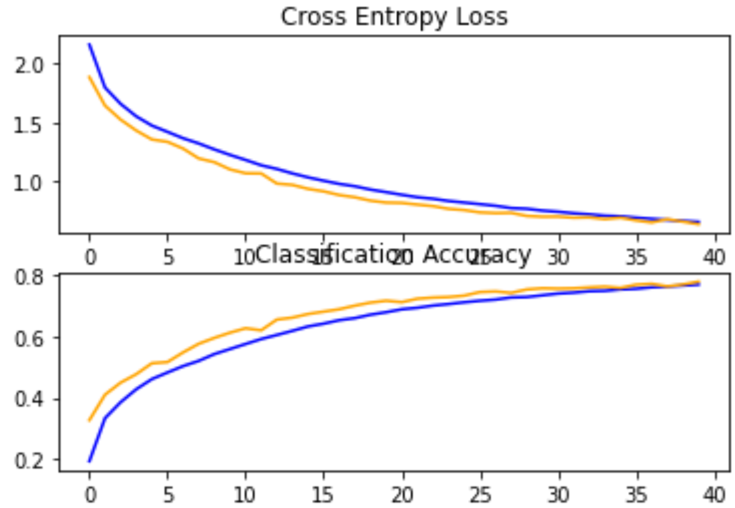
```
test_loss, test_acc = model2.evaluate(test_images , y_test)# TODO

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

```
313/313 [==============================] - 1s 3ms/step - loss: 0.6346 - accur
acy: 0.7789
Test accuracy: 0.7789000272750854
```

In [77]:

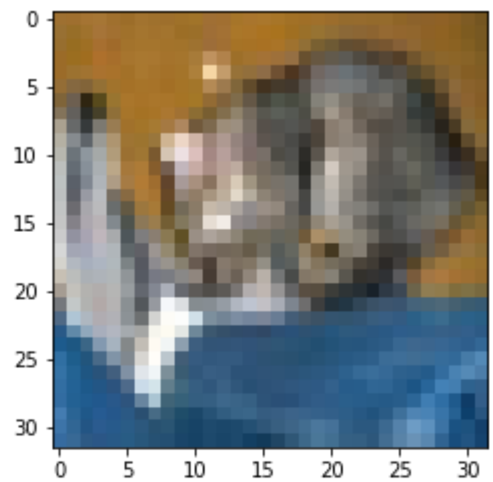
```
summarize_diagnostics(H)
```



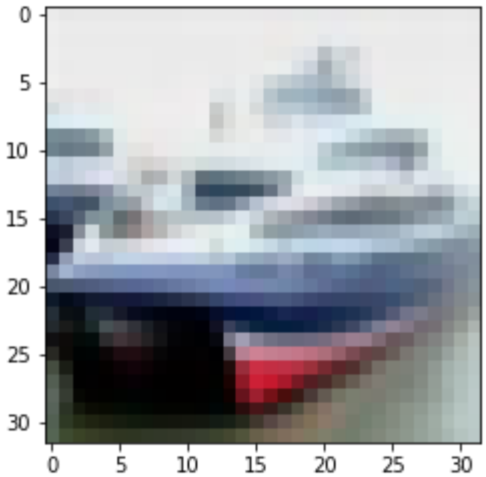
In [78]:

```
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()
```

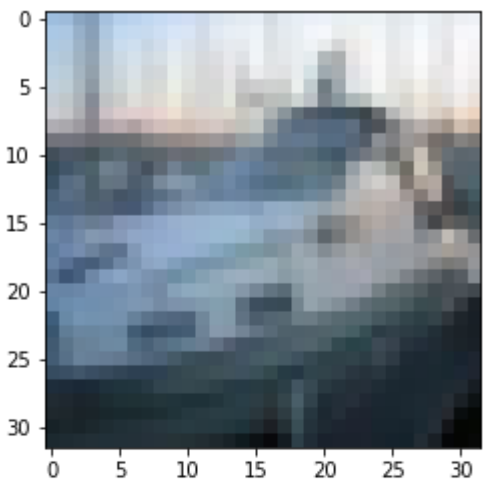
```
Class image :  4
Label of this digit is: [0 0 0 1 0 0 0 0 0 0]
```



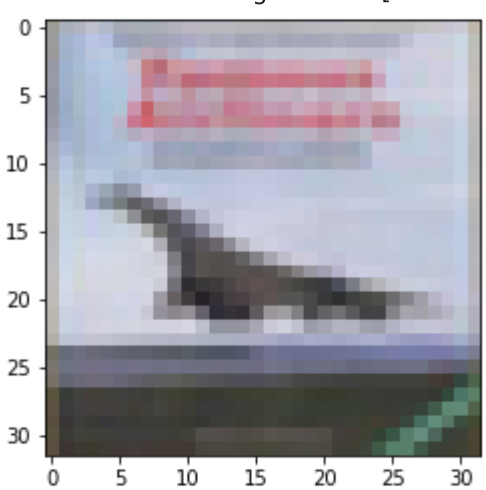
```
Class image :  9
Label of this digit is: [0 0 0 0 0 0 0 0 1 0]
```



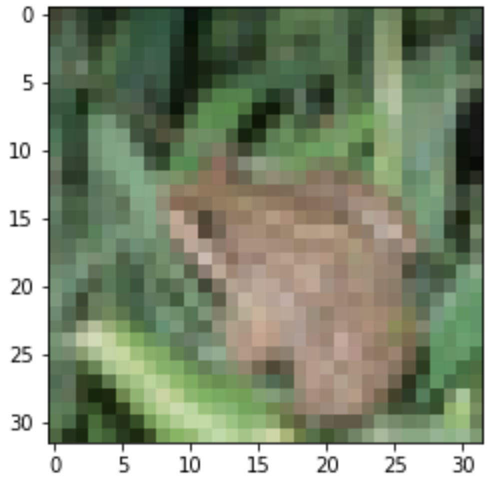

```
Class image :  9
Label of this digit is: [0 0 0 0 0 0 0 0 1 0]
```



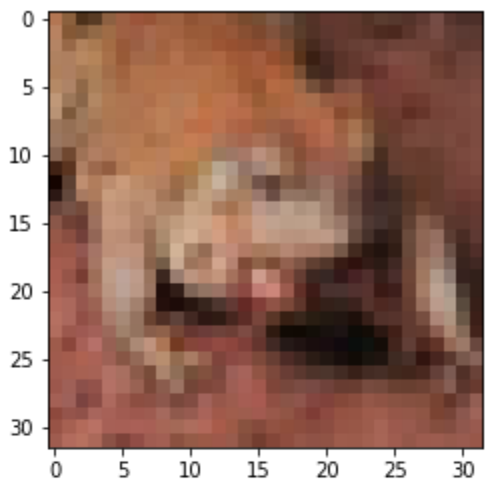

```
Class image :  1
Label of this digit is: [1 0 0 0 0 0 0 0 0 0]
```



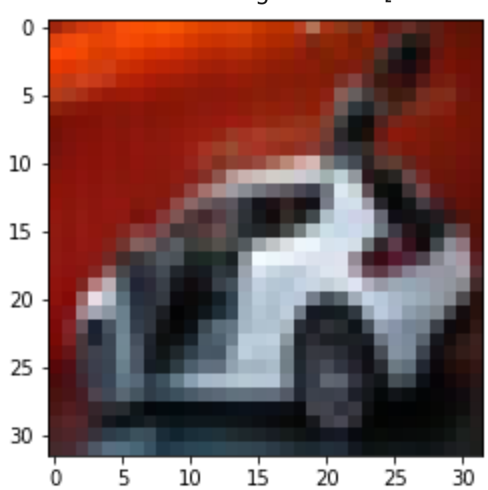

```
Class image :  7
Label of this digit is: [0 0 0 0 0 0 1 0 0 0]
```



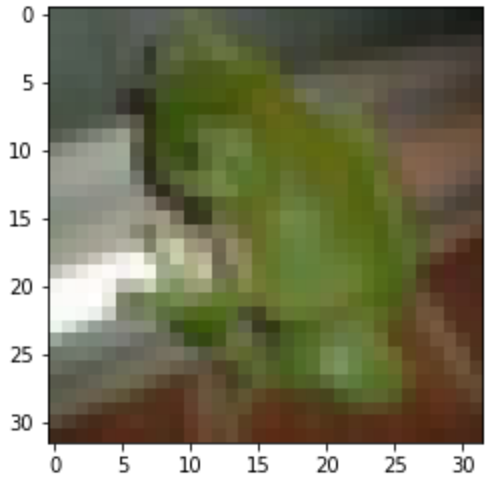
```
Class image :  7
Label of this digit is: [0 0 0 0 0 0 1 0 0 0]
```



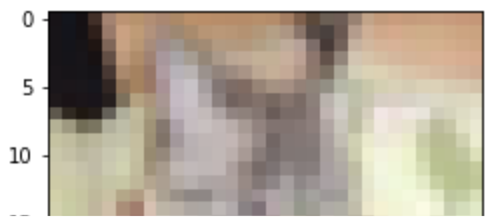
```
Class image :  2
Label of this digit is: [0 1 0 0 0 0 0 0 0 0]
```



```
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]
```



In [ ]:

```python
# predictions = model2.predict(np.array([test_images[2]]))
```

In [ ]:

```python
# predictions
```

Out[ ]:

```
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 [ ]:

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

```
Class image :  9
```

In [ ]:

```python
# 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[ ]: <matplotlib.image.AxesImage at 0x7f3625ee0890>

In [79]:
```
from tensorflow.python.client import device_lib
device_lib.list_local_devices()
```

Out[79]:
```
[name: "/device:CPU:0"
 device_type: "CPU"
 memory_limit: 268435456
 locality {
 }
 incarnation: 11203134324537477335, name: "/device:GPU:0"
 device_type: "GPU"
 memory_limit: 16183459840
 locality {
   bus_id: 1
   links {
   }
 }
 incarnation: 1557725047161276091
 physical_device_desc: "device: 0, name: Tesla P100-PCIE-16GB, pci bus id: 00
00:00:04.0, compute capability: 6.0"]
```