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 (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.

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 [1]:
```
def unpickle(file):
    import pickle
    with open(file, 'rb') as fo:
        dict = pickle.load(fo, encoding='bytes')
    return dict[b'data'],dict[b'labels']
```

In [2]:
```
import glob
import os
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import numpy as np
from matplotlib import pyplot as plt
DATA_train =[]
Label_train =[]
DATA_TEST =[]
Label_TEST=[]
```

In [3]:
```
for i in range(1,6):
    Data,label=unpickle('data_batch_'+str(i))
    DATA_train.extend(Data)
    Label_train.extend(label)
```

In [4]:
```
Data_test,label_test =unpickle('test_batch')
DATA_TEST.extend(Data_test)
Label_TEST.extend(label_test)
```

In [5]:
```
print ("DATA_TRAIN =  ",len(DATA_train),"LABEL_TRAIN = ",len(Label_train))
print ("DATA_TEST  =  ",len(DATA_TEST),"LABEL_TEST  = ",len(Label_TEST))
```

```
DATA_TRAIN =   50000 LABEL_TRAIN =  50000
DATA_TEST  =   10000 LABEL_TEST  =  10000
```

In [28]:
```
# ผมได้เปลี่ยนภาพ โดนการอ่านไฟล์จาก ตัวอย่างโค้ดของ https://www.cs.toronto.edu/~kriz/
cifar.html โดยมี  fuction unpickle มาให้อยู่แล้วแต่ผมได้เติม
# ตรง returnเข้าไปให้ returnออกมาเป็น 2ตัวเลย ฝั่งซ้ายเป็นรูปภาพ โดยจะได้ค่าจากภาพและ lab
el
# โดยผมอ่าน data_batch_1-5 แล้ว extend เข้าไปใน DATAtrain และ LAbeltrain
#ทำเช่นนี้เหมือนกัน แต่ ทำใน DATATESTด้วย
# เป็น DATA_TRAIN มี50000 รูป และ label จะมีเท่ากันโดยเรียงlabelตามลำดับในlist เหมือนกัน
# เป็น DATA_TEST มี10000 รูป และ label จะมีเท่ากันโดยเรียงลำดับ labelคู่กับรูปภาพ เช่นกัน
```

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 [6]:
```
DATA_DOGCAT_TRAIN =[]
LABEL_DOGCAT_TRAIN =[]
DATA_DOGCAT_TEST =[]
LABEL_DOGCAT_TEST =[]
for i in range(0,len(Label_train)):
    if Label_train[i]==3 or Label_train[i] ==5 :
        DATA_DOGCAT_TRAIN.append(DATA_train[i])
        LABEL_DOGCAT_TRAIN.append(Label_train[i])
```

In [7]:
```
for i in range(0,len(Label_TEST)):
    if Label_TEST[i]==3 or Label_TEST[i] ==5 :
        DATA_DOGCAT_TEST.append(DATA_TEST[i])
        LABEL_DOGCAT_TEST.append(Label_TEST[i])
```

In [8]:
```
print ("DATA_TRAIN_DOGCAT =  ",len(DATA_DOGCAT_TRAIN),"LABEL_TRAIN_DOGCAT = ",
len(LABEL_DOGCAT_TRAIN))
print ("DATA_TEST_DOGCAT  =  ",len(DATA_DOGCAT_TEST),"LABEL_TEST_DOGCAT  = ",l
en(LABEL_DOGCAT_TEST))
```

```
DATA_TRAIN_DOGCAT =   10000 LABEL_TRAIN_DOGCAT =  10000
DATA_TEST_DOGCAT  =   2000 LABEL_TEST_DOGCAT  =  2000
```

In [9]:
```
DOGCAT_TRAIN_ARRAY =np.asarray(DATA_DOGCAT_TRAIN)
DOGCAT_TEST_ARRAY =np.asarray(DATA_DOGCAT_TEST)
LOGIS_Model = LogisticRegression()
LOGIS_Model.fit(DOGCAT_TRAIN_ARRAY,LABEL_DOGCAT_TRAIN)
LABEL_DOGCAT_PRED= LOGIS_Model.predict(DOGCAT_TEST_ARRAY)
accuracy_score(y_pred = LABEL_DOGCAT_PRED,y_true = LABEL_DOGCAT_TEST)
```

Out[9]: 0.5325

In [29]:
```
#ส่วนข้อนี้ ผมแยก label =3 และ label=5
#โดยรันผ่าน forloop ทำไปเรื่อยๆเช็คทุกรูปภาพ แต่เช็คผ่านlabel
# หากมี label =3,5 จะให้ append เข้าlist ใหม่ทั้ง ภาพและ label
# ทำเช่นนี้ทั้งtrain และ test จะทำให้ listใหม่ที่ได้มาจะเป็นlistที่มีlabel
# เพียงแค่3 และ 5 โดยเหลือ
#DATA_TRAIN_DOGCAT =   10000 LABEL_TRAIN_DOGCAT =  10000
#DATA_TEST_DOGCAT  =   2000 LABEL_TEST_DOGCAT  =  2000
#หลังจากนั้นผมไปแปลงเป็น numpy array เพื่อเข้าไป fit model
# model logisticregression
# ตามโจทย์ หลังจาก fit ด้วย data ,label ของtrainแล้ว
#ลอง predict label ด้วย data_test หลังจากนั้นนำมาเทียบกับ
# label_test_dogcat เพื่อหา accuracy และได้0.5325
```

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 [ ]:
```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
MLP_CLASSI = MLPClassifier(hidden_layer_sizes = 30)
MLP_CLASSI.fit(DATA_train,Label_train)
PRED_MLP = MLP_CLASSI.predict(DATA_TEST)
accuracy_score(Label_TEST,PRED_MLP)
```

Out[ ]: 0.1

In [27]:
```
#ข้อนี้ผมทำการทำmodelขึ้นมา3แบบ
#แบบแรกใช้ mlp โดยมี hiddenlayer =30
#และไปเทียบกับ test และ train  data ของเรา
#ทำเหมือนข้อบนๆคือ fit train และ
# นำ model ไป predict ไฟล์ test และเปรียบเทียบ
# accuracy ได้เพียง 0.1 = 10%
```

In [9]:
```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
MLP_CLASSI = MLPClassifier(hidden_layer_sizes = 800)
MLP_CLASSI.fit(DATA_train,Label_train)
PRED_MLP = MLP_CLASSI.predict(DATA_TEST)
accuracy_score(Label_TEST,PRED_MLP)
```

Out[9]: 0.1331

In [26]:
```
#แบบ2คือ ทำเหมือนmodelแบบแรกเปลี่ยน hiddenlayer30เป็น800
# ได้ค่า accuracyขึ้นมาเป็น 0.133= 13%
#ขึ้นมาเพียงนิดเดียว
```

In [9]:
```
import keras
import numpy
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.layers import BatchNormalization

from keras.constraints import maxnorm
from keras.optimizers import SGD
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.utils import np_utils
from keras import backend as K
K.set_image_dim_ordering('th')
```

Using TensorFlow backend.

In [10]:
```
seed = 8
numpy.random.seed(seed)
```

In [11]:
```
DATA_train = np.array(DATA_train)
LABEL_train = np.array(Label_train)
DATA_test = np.array(DATA_TEST)
LABEL_test = np.array(Label_TEST)
```

In [12]:
```
DATA_train = DATA_train.reshape((50000,32,32,3))
DATA_test = DATA_test.reshape((10000,32,32,3))
```

In [13]:
```
DATA_train = DATA_train.astype('float32')
DATA_test = DATA_test.astype('float32')
DATA_train = DATA_train / 255.0
DATA_test = DATA_test / 255.0
```

In [14]:
```
LABEL_train = np_utils.to_categorical(LABEL_train)
LABEL_test = np_utils.to_categorical(LABEL_test)
num_classes = LABEL_test.shape[1]
```

In [18]:
```
model = Sequential()
model.add(Conv2D(32, (3,3), input_shape=(32,32,3), padding='same', activatio
n='relu', data_format='channels_last'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, (1, 1), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(512, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['a
ccuracy'])
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_7 (Conv2D)            (None, 32, 32, 32)        896       
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 32, 16, 16)        0         
_________________________________________________________________
batch_normalization_7 (Batch (None, 32, 16, 16)        64        
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 32, 16, 16)        9248      
_________________________________________________________________
batch_normalization_8 (Batch (None, 32, 16, 16)        64        
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 32, 8, 8)          0         
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 32, 8, 8)          1056      
_________________________________________________________________
batch_normalization_9 (Batch (None, 32, 8, 8)          32        
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 32, 4, 4)          0         
_________________________________________________________________
flatten_3 (Flatten)          (None, 512)               0         
_________________________________________________________________
dense_5 (Dense)              (None, 512)               262656    
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_6 (Dense)              (None, 10)                5130      
=================================================================
Total params: 279,146
Trainable params: 279,066
Non-trainable params: 80
_________________________________________________________________
```

In [22]:
```
model.fit(DATA_train,LABEL_train, epochs=35, batch_size=32,validation_data=(DA
TA_test,LABEL_test))
```

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/35
50000/50000 [==============================] - 58s 1ms/step - loss: 0.6736
- acc: 0.7574 - val_loss: 1.3017 - val_acc: 0.5977
Epoch 2/35
50000/50000 [==============================] - 54s 1ms/step - loss: 0.6622
- acc: 0.7586 - val_loss: 1.3161 - val_acc: 0.5905
Epoch 3/35
50000/50000 [==============================] - 55s 1ms/step - loss: 0.6425
- acc: 0.7697 - val_loss: 1.2776 - val_acc: 0.6004
Epoch 4/35
50000/50000 [==============================] - 54s 1ms/step - loss: 0.6329
- acc: 0.7730 - val_loss: 1.2499 - val_acc: 0.6033
Epoch 5/35
50000/50000 [==============================] - 54s 1ms/step - loss: 0.6308
- acc: 0.7725 - val_loss: 1.2592 - val_acc: 0.6022
Epoch 6/35
50000/50000 [==============================] - 54s 1ms/step - loss: 0.6097
- acc: 0.7791 - val_loss: 1.3139 - val_acc: 0.6037
Epoch 7/35
50000/50000 [==============================] - 54s 1ms/step - loss: 0.5973
- acc: 0.7852 - val_loss: 1.3414 - val_acc: 0.5905
Epoch 8/35
50000/50000 [==============================] - 54s 1ms/step - loss: 0.5908
- acc: 0.7854 - val_loss: 1.3088 - val_acc: 0.6004
Epoch 9/35
50000/50000 [==============================] - 55s 1ms/step - loss: 0.5825
- acc: 0.7904 - val_loss: 1.3315 - val_acc: 0.5993
Epoch 10/35
50000/50000 [==============================] - 56s 1ms/step - loss: 0.5749
- acc: 0.7930 - val_loss: 1.3955 - val_acc: 0.5945
Epoch 11/35
50000/50000 [==============================] - 54s 1ms/step - loss: 0.5693
- acc: 0.7926 - val_loss: 1.3060 - val_acc: 0.5949
Epoch 12/35
50000/50000 [==============================] - 54s 1ms/step - loss: 0.5592
- acc: 0.7995 - val_loss: 1.2892 - val_acc: 0.5918
Epoch 13/35
50000/50000 [==============================] - 54s 1ms/step - loss: 0.5497
- acc: 0.8017 - val_loss: 1.3896 - val_acc: 0.5986
Epoch 14/35
50000/50000 [==============================] - 58s 1ms/step - loss: 0.5425
- acc: 0.8030 - val_loss: 1.3687 - val_acc: 0.5918
Epoch 15/35
50000/50000 [==============================] - 54s 1ms/step - loss: 0.5328
- acc: 0.8068 - val_loss: 1.4608 - val_acc: 0.5980
Epoch 16/35
50000/50000 [==============================] - 56s 1ms/step - loss: 0.5313
- acc: 0.8093 - val_loss: 1.4539 - val_acc: 0.6010
Epoch 17/35
50000/50000 [==============================] - 56s 1ms/step - loss: 0.5270
- acc: 0.8074 - val_loss: 1.4227 - val_acc: 0.5980
Epoch 18/35
50000/50000 [==============================] - 57s 1ms/step - loss: 0.5254
- acc: 0.8114 - val_loss: 1.3772 - val_acc: 0.5914
Epoch 19/35
50000/50000 [==============================] - 55s 1ms/step - loss: 0.5146
```

```
- acc: 0.8116 - val_loss: 1.3745 - val_acc: 0.5905
Epoch 20/35
50000/50000 [==============================] - 54s 1ms/step - loss: 0.5110
- acc: 0.8152 - val_loss: 1.4149 - val_acc: 0.5964
Epoch 21/35
50000/50000 [==============================] - 53s 1ms/step - loss: 0.5043
- acc: 0.8176 - val_loss: 1.4205 - val_acc: 0.5944
Epoch 22/35
50000/50000 [==============================] - 54s 1ms/step - loss: 0.4983
- acc: 0.8203 - val_loss: 1.4035 - val_acc: 0.5948
Epoch 23/35
50000/50000 [==============================] - 53s 1ms/step - loss: 0.4965
- acc: 0.8199 - val_loss: 1.3960 - val_acc: 0.5905
Epoch 24/35
50000/50000 [==============================] - 53s 1ms/step - loss: 0.4982
- acc: 0.8212 - val_loss: 1.4970 - val_acc: 0.5963
Epoch 25/35
50000/50000 [==============================] - 54s 1ms/step - loss: 0.4893
- acc: 0.8231 - val_loss: 1.5243 - val_acc: 0.5898
Epoch 26/35
50000/50000 [==============================] - 53s 1ms/step - loss: 0.4807
- acc: 0.8259 - val_loss: 1.5915 - val_acc: 0.5839
Epoch 27/35
50000/50000 [==============================] - 53s 1ms/step - loss: 0.4787
- acc: 0.8263 - val_loss: 1.5173 - val_acc: 0.5992
Epoch 28/35
50000/50000 [==============================] - 52s 1ms/step - loss: 0.4829
- acc: 0.8262 - val_loss: 1.5013 - val_acc: 0.5877
Epoch 29/35
50000/50000 [==============================] - 53s 1ms/step - loss: 0.4736
- acc: 0.8293 - val_loss: 1.5738 - val_acc: 0.5834
Epoch 30/35
50000/50000 [==============================] - 53s 1ms/step - loss: 0.4690
- acc: 0.8316 - val_loss: 1.5251 - val_acc: 0.5783
Epoch 31/35
50000/50000 [==============================] - 52s 1ms/step - loss: 0.4613
- acc: 0.8324 - val_loss: 1.5896 - val_acc: 0.5906
Epoch 32/35
50000/50000 [==============================] - 53s 1ms/step - loss: 0.4643
- acc: 0.8317 - val_loss: 1.5589 - val_acc: 0.5939
Epoch 33/35
50000/50000 [==============================] - 53s 1ms/step - loss: 0.4519
- acc: 0.8352 - val_loss: 1.5866 - val_acc: 0.5910
Epoch 34/35
50000/50000 [==============================] - 52s 1ms/step - loss: 0.4601
- acc: 0.8338 - val_loss: 1.4815 - val_acc: 0.5951
Epoch 35/35
50000/50000 [==============================] - 54s 1ms/step - loss: 0.4550
- acc: 0.8356 - val_loss: 1.4538 - val_acc: 0.5914
```

Out[22]:
```
<keras.callbacks.History at 0x273f2e4d390>
```

In [24]:
```python
# Final evaluation of the model
scores = model.evaluate(DATA_test, LABEL_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Accuracy: 59.14%
```

In [25]:
```
#ขั้นตอนสุดท้ายผมทำ cnn โดยมีเรฟเฟอเร้นจาก
# https://machinelearningmastery.com/object-recognition-convolutional-neural-n
etworks-keras-deep-learning-library/
#และปรับเปรียบค่า model ให้ค่า accuracyสูงขึ้น และทำไปทั้งหมด 35 รอบ
# โดยผมConfig Number of layer , Number of Neural , Activation Fucntion for  Ne
ural Network Model
#ไว้แล้ว หลังจากนั้น ก็รันและได้ค่า accuracy ที่สูงขึ้นจาก13%
#สูงมามากถึง 59% หากเรารันจำนวนรอบมากกว่านี้จะได้% ที่สูงมากกว่านี้
```