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 [407]:

```
from matplotlib import pyplot
from keras.datasets import cifar10
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

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

In [408]:

```
X_train = X_train.reshape(50000, 3072)
y_train = y_train.reshape(-1)
print(X_train.shape)
print(y_train.shape)

X_test = X_test.reshape(10000, 3072)
y_test = y_test.reshape(-1)
print(X_test.shape)
print(y_test.shape)
```

```
(50000, 3072)
(50000,)
(10000, 3072)
(10000,)
```

In [409]:

```
X_train = X_train / 255
X_test = X_test / 255
```

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 [410]:

```
dog_images_train = []
for i in range (0, 50000):
    if labels[y_train[i]] == 'dog':
        dog_images_train.append(X_train[i])
print(len(dog_images_train))

dog_images_test = []
for i in range (0, 10000):
    if labels[y_test[i]] == 'dog':
        dog_images_test.append(X_test[i])
print(len(dog_images_test))
```

```
5000
1000
```

In [411]:

```
dog_labels_train = [3] * 5000
dog_labels_test = [3] * 1000
```

In [412]:

```
cat_images_train = []
for i in range (0, 50000):
    if labels[y_train[i]] == 'cat':
        cat_images_train.append(X_train[i])
print(len(dog_images_train))

cat_images_test = []
for i in range (0, 10000):
    if labels[y_test[i]] == 'cat':
        cat_images_test.append(X_test[i])
print(len(dog_images_test))
```

```
5000
1000
```

In [413]:

```
cat_labels_train = [5] * 5000
cat_labels_test = [5] * 1000
```

In [414]:

```
dog_cat_images_train = np.array(dog_images_train + cat_images_train)
dog_cat_labels_train = np.array(dog_labels_train + cat_labels_train)

dog_cat_images_test = np.array(dog_images_test + cat_images_test)
dog_cat_labels_test = np.array(dog_labels_test + cat_labels_test)
```

In [415]:

```
from sklearn.linear_model import LogisticRegression
LR = LogisticRegression(random_state = 0, max_iter = 2000)
LR.fit(dog_cat_images_train, dog_cat_labels)
```

Out[415]:

```
LogisticRegression(max_iter=2000, random_state=0)
```

In [416]:

```
y_pred = LR.predict(dog_cat_images_test)
```

In [417]:

```
from sklearn.metrics import accuracy_score
accuracy_score(dog_cat_labels_test, y_pred)
```

Out[417]:

```
0.5805
```

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 [512]:

```
# first experiment using CNN
import numpy
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense,Dropout,Flatten,Conv2D,MaxPooling2D
from keras.constraints import maxnorm
from keras.optimizers import SGD
from keras.utils import np_utils
from keras import backend
import keras as K
```

In [513]:

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

In [514]:

```
#normalize the inputs from 0-255 to 0-1 ,as pixels have value in the range 0-255 hence we n
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train = X_train / 255.0
X_test = X_test / 255.0
```

In [515]:

```
# one hot encode outputs for better prediction
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]
```

In [516]:

```
# Create the model using sequentia as it allows to create the model layer-by-layer
model = Sequential()
```

In [ ]:

```
# add all the required layers for CNN
model.add(Conv2D(32, (3, 3), input_shape=(32,32,3), activation='relu', padding='same'))
model.add(Dropout(0.2))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(Dropout(0.2))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(Dropout(0.2))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dropout(0.2))
model.add(Dense(1024, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
```

In [542]:

```
model.summary()
```

```
Model: "sequential_17"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_34 (Conv2D)           (None, 32, 32, 32)        896       
_________________________________________________________________
dropout_24 (Dropout)         (None, 32, 32, 32)        0         
_________________________________________________________________
conv2d_35 (Conv2D)           (None, 32, 32, 32)        9248      
_________________________________________________________________
max_pooling2d_20 (MaxPooling (None, 16, 16, 32)        0         
_________________________________________________________________
conv2d_36 (Conv2D)           (None, 16, 16, 64)        18496     
_________________________________________________________________
dropout_25 (Dropout)         (None, 16, 16, 64)        0         
_________________________________________________________________
conv2d_37 (Conv2D)           (None, 16, 16, 64)        36928     
_________________________________________________________________
max_pooling2d_21 (MaxPooling (None, 8, 8, 64)          0         
_________________________________________________________________
conv2d_38 (Conv2D)           (None, 8, 8, 128)         73856     
_________________________________________________________________
dropout_26 (Dropout)         (None, 8, 8, 128)         0         
_________________________________________________________________
conv2d_39 (Conv2D)           (None, 8, 8, 128)         147584    
_________________________________________________________________
max_pooling2d_22 (MaxPooling (None, 4, 4, 128)         0         
_________________________________________________________________
flatten_8 (Flatten)          (None, 2048)              0         
_________________________________________________________________
dropout_27 (Dropout)         (None, 2048)              0         
_________________________________________________________________
dense_16 (Dense)             (None, 1024)              2098176   
_________________________________________________________________
dropout_28 (Dropout)         (None, 1024)              0         
_________________________________________________________________
dense_17 (Dense)             (None, 512)               524800    
_________________________________________________________________
dropout_29 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_18 (Dense)             (None, 10)                5130      
=================================================================
Total params: 2,915,114
Trainable params: 2,915,114
Non-trainable params: 0
_________________________________________________________________
```

In [534]:

```
# Compile model
epochs = 10
Irate = 0.01
decay = lrate/epochs
sgd = SGD(lr = Irate, momentum=0.9, decay=lrate/epochs, nesterov=False,)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=40, batch_size=32)
```

```
Epoch 25/40
1563/1563 [==============================] - 151s 97ms/step - loss: 0.3487
- accuracy: 0.8763 - val_loss: 0.6233 - val_accuracy: 0.7918
Epoch 26/40
1563/1563 [==============================] - 150s 96ms/step - loss: 0.3385
- accuracy: 0.8792 - val_loss: 0.6207 - val_accuracy: 0.7947
Epoch 27/40
1563/1563 [==============================] - 150s 96ms/step - loss: 0.3388
- accuracy: 0.8795 - val_loss: 0.6176 - val_accuracy: 0.7962
Epoch 28/40
1563/1563 [==============================] - 151s 97ms/step - loss: 0.3369
- accuracy: 0.8795 - val_loss: 0.6216 - val_accuracy: 0.7952
Epoch 29/40
1563/1563 [==============================] - 150s 96ms/step - loss: 0.3265
- accuracy: 0.8841 - val_loss: 0.6167 - val_accuracy: 0.7971
Epoch 30/40
1563/1563 [==============================] - 149s 96ms/step - loss: 0.3309
- accuracy: 0.8823 - val_loss: 0.6147 - val_accuracy: 0.7980
Epoch 31/40
1563/1563 [==============================] - 150s 96ms/step - loss: 0.3216
```

In [537]:

```
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=32)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
```

```
Epoch 1/10
1563/1563 [==============================] - 154s 99ms/step - loss: 0.2813 -
accuracy: 0.8991 - val_loss: 0.6202 - val_accuracy: 0.8006
Epoch 2/10
1563/1563 [==============================] - 151s 97ms/step - loss: 0.2760 -
accuracy: 0.9001 - val_loss: 0.6246 - val_accuracy: 0.8013
Epoch 3/10
1563/1563 [==============================] - 151s 97ms/step - loss: 0.2733 -
accuracy: 0.9017 - val_loss: 0.6243 - val_accuracy: 0.8025
Epoch 4/10
1563/1563 [==============================] - 152s 97ms/step - loss: 0.2727 -
accuracy: 0.9031 - val_loss: 0.6287 - val_accuracy: 0.8007
Epoch 5/10
1563/1563 [==============================] - 149s 95ms/step - loss: 0.2675 -
accuracy: 0.9034 - val_loss: 0.6294 - val_accuracy: 0.7993
Epoch 6/10
1563/1563 [==============================] - 150s 96ms/step - loss: 0.2631 -
accuracy: 0.9055 - val_loss: 0.6294 - val_accuracy: 0.7990
Epoch 7/10
1563/1563 [==============================] - 154s 98ms/step - loss: 0.2630 -
accuracy: 0.9050 - val_loss: 0.6224 - val_accuracy: 0.8017
Epoch 8/10
1563/1563 [==============================] - 150s 96ms/step - loss: 0.2630 -
accuracy: 0.9058 - val_loss: 0.6331 - val_accuracy: 0.8011
Epoch 9/10
1563/1563 [==============================] - 152s 97ms/step - loss: 0.2595 -
accuracy: 0.9068 - val_loss: 0.6266 - val_accuracy: 0.8006
Epoch 10/10
1563/1563 [==============================] - 150s 96ms/step - loss: 0.2571 -
accuracy: 0.9069 - val_loss: 0.6311 - val_accuracy: 0.8014
```

In [541]:

```
# second experiment
model_2 = Sequential()
model_2.add(Conv2D(16, (3, 3), activation='relu', strides=(1, 1), padding='same', input_sha
model_2.add(Conv2D(32, (3, 3), activation='relu', strides=(1, 1), padding='same'))
model_2.add(Conv2D(64, (3, 3), activation='relu', strides=(1, 1), padding='same'))
model_2.add(MaxPooling2D((2, 2)))
model_2.add(Conv2D(16, (3, 3), activation='relu', strides=(1, 1), padding='same'))
model_2.add(Conv2D(32, (3, 3), activation='relu', strides=(1, 1),  padding='same'))
model_2.add(Conv2D(64, (3, 3), activation='relu', strides=(1, 1),  padding='same'))
model_2.add(MaxPooling2D((2, 2)))
model_2.add(Flatten())
model_2.add(Dense(256, activation='relu'))
model_2.add(Dropout(0.5))
model_2.add(Dense(128, activation='relu'))
model_2.add(Dense(64, activation='relu'))
model_2.add(Dense(64, activation='relu'))
model_2.add(Dense(10, activation='softmax'))
```

In [544]:

```
model_2.summary()
```

```
Model: "sequential_20"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_46 (Conv2D)           (None, 32, 32, 16)        448       
_________________________________________________________________
conv2d_47 (Conv2D)           (None, 32, 32, 32)        4640      
_________________________________________________________________
conv2d_48 (Conv2D)           (None, 32, 32, 64)        18496     
_________________________________________________________________
max_pooling2d_23 (MaxPooling (None, 16, 16, 64)        0         
_________________________________________________________________
conv2d_49 (Conv2D)           (None, 16, 16, 16)        9232      
_________________________________________________________________
conv2d_50 (Conv2D)           (None, 16, 16, 32)        4640      
_________________________________________________________________
conv2d_51 (Conv2D)           (None, 16, 16, 64)        18496     
_________________________________________________________________
max_pooling2d_24 (MaxPooling (None, 8, 8, 64)          0         
_________________________________________________________________
flatten_9 (Flatten)          (None, 4096)              0         
_________________________________________________________________
dense_19 (Dense)             (None, 256)               1048832   
_________________________________________________________________
dropout_30 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_20 (Dense)             (None, 128)               32896     
_________________________________________________________________
dense_21 (Dense)             (None, 64)                8256      
_________________________________________________________________
dense_22 (Dense)             (None, 64)                4160      
_________________________________________________________________
dense_23 (Dense)             (None, 10)                650       
=================================================================
Total params: 1,150,746
Trainable params: 1,150,746
Non-trainable params: 0
_________________________________________________________________
```

In [550]:

```
# Compile model
epochs = 12
Irate = 0.01
decay = lrate/epochs
sgd = SGD(lr = Irate, momentum= 1, decay=lrate/epochs, nesterov=False,)
model_2.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
model_2.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=20, batch_size=32)
```

```
Epoch 1/20
1563/1563 [==============================] - 106s 68ms/step - loss: nan -
accuracy: 0.1021 - val_loss: nan - val_accuracy: 0.1000
Epoch 2/20
1563/1563 [==============================] - 115s 73ms/step - loss: nan -
accuracy: 0.1000 - val_loss: nan - val_accuracy: 0.1000
Epoch 3/20
1563/1563 [==============================] - 114s 73ms/step - loss: nan -
accuracy: 0.1000 - val_loss: nan - val_accuracy: 0.1000
Epoch 4/20
1563/1563 [==============================] - 114s 73ms/step - loss: nan -
accuracy: 0.1000 - val_loss: nan - val_accuracy: 0.1000
Epoch 5/20
1563/1563 [==============================] - 112s 72ms/step - loss: nan -
accuracy: 0.1000 - val_loss: nan - val_accuracy: 0.1000
Epoch 6/20
1563/1563 [==============================] - 113s 72ms/step - loss: nan -
accuracy: 0.1000 - val_loss: nan - val_accuracy: 0.1000
Epoch 7/20
```

In [553]:

```
model_2.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=32)
# Final evaluation of the model
scores = model_2.evaluate(X_test, y_test, verbose=0)
```

```
Epoch 1/10
1563/1563 [==============================] - 117s 75ms/step - loss: nan - ac
curacy: 0.1000 - val_loss: nan - val_accuracy: 0.1000
Epoch 2/10
1563/1563 [==============================] - 118s 75ms/step - loss: nan - ac
curacy: 0.1000 - val_loss: nan - val_accuracy: 0.1000
Epoch 3/10
1563/1563 [==============================] - 117s 75ms/step - loss: nan - ac
curacy: 0.1000 - val_loss: nan - val_accuracy: 0.1000
Epoch 4/10
1563/1563 [==============================] - 144s 92ms/step - loss: nan - ac
curacy: 0.1000 - val_loss: nan - val_accuracy: 0.1000
Epoch 5/10
1563/1563 [==============================] - 117s 75ms/step - loss: nan - ac
curacy: 0.1000 - val_loss: nan - val_accuracy: 0.1000
Epoch 6/10
1563/1563 [==============================] - 117s 75ms/step - loss: nan - ac
curacy: 0.1000 - val_loss: nan - val_accuracy: 0.1000
Epoch 7/10
1563/1563 [==============================] - 119s 76ms/step - loss: nan - ac
curacy: 0.1000 - val_loss: nan - val_accuracy: 0.1000
Epoch 8/10
1563/1563 [==============================] - 118s 75ms/step - loss: nan - ac
curacy: 0.1000 - val_loss: nan - val_accuracy: 0.1000
Epoch 9/10
1563/1563 [==============================] - 117s 75ms/step - loss: nan - ac
curacy: 0.1000 - val_loss: nan - val_accuracy: 0.1000
Epoch 10/10
1563/1563 [==============================] - 118s 75ms/step - loss: nan - ac
curacy: 0.1000 - val_loss: nan - val_accuracy: 0.1000
```