Cifar10

February 3, 2022

1 Image Classification with CIFAR-10 dataset

In this notebook, I am going to classify images from the CIFAR-10 dataset. The dataset consists of airplanes, dogs, cats, and other objects. You'll preprocess the images, then train a convolutional neural network on all the samples. The images need to be normalized and the labels need to be one-hot encoded.

1.1 Get the Data

Run the following cell to download the CIFAR-10 dataset for python.

1.1.1 List of files

the dataset is broken into batches to **prevent** your machine from running **out of memory**. The CIFAR-10 dataset consists of 5 batches, named `data_batch_1`, `data_batch_2`, etc..

1.1.2 Understanding the original data

In order to feed an image data into a CNN model, the dimension of the tensor representing an image data should be either (width x height x num_channel) or (num_channel x width x height). I am going to use the dimension of the first choice because the default choice in tensorflow's CNN operation is so.

1.1.3 Understanding the original labels

The label data is just a list of 10000 numbers in the range 0-9, which corresponds to each of the 10 classes in CIFAR-10.

- **airplane**
- **automobile**
- **bird**
- **cat**
- **deer**
- **dog**
- **frog**
- **horse**
- **ship**
- **truck**

[1]:

```
from urllib.request import urlretrieve
from os.path import isfile, isdir
from tqdm import tqdm
import tarfile

cifar10_dataset_folder_path = 'cifar-10-batches-py'

class DownloadProgress(tqdm):
    last_block = 0

    def hook(self, block_num=1, block_size=1, total_size=None):
        self.total = total_size
        self.update((block_num - self.last_block) * block_size)
        self.last_block = block_num

"""
    check if the data (zip) file is already downloaded
    if not, download it from "https://www.cs.toronto.edu/~kriz/cifar-10-python.
 ↪tar.gz" and save as cifar-10-python.tar.gz
"""
if not isfile('cifar-10-python.tar.gz'):
    with DownloadProgress(unit='B', unit_scale=True, miniters=1, desc='CIFAR-10␣
 ↪Dataset') as pbar:
        urlretrieve(
            'https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz',
            'cifar-10-python.tar.gz',
            pbar.hook)

if not isdir(cifar10_dataset_folder_path):
    with tarfile.open('cifar-10-python.tar.gz') as tar:
        tar.extractall()
        tar.close()
```

```
CIFAR-10 Dataset: 171MB [00:02, 68.1MB/s]
```

[3]:

```
def load_cfar10_batch(cifar10_dataset_folder_path, batch_id):
    with open(cifar10_dataset_folder_path + '/data_batch_' + str(batch_id),␣
 ↪mode='rb') as file:
        # note the encoding type is 'latin1'
        batch = pickle.load(file, encoding='latin1')

    features = batch['data'].reshape((len(batch['data']), 3, 32, 32)).
 ↪transpose(0, 2, 3, 1)
    labels = batch['labels']

    return features, labels
```

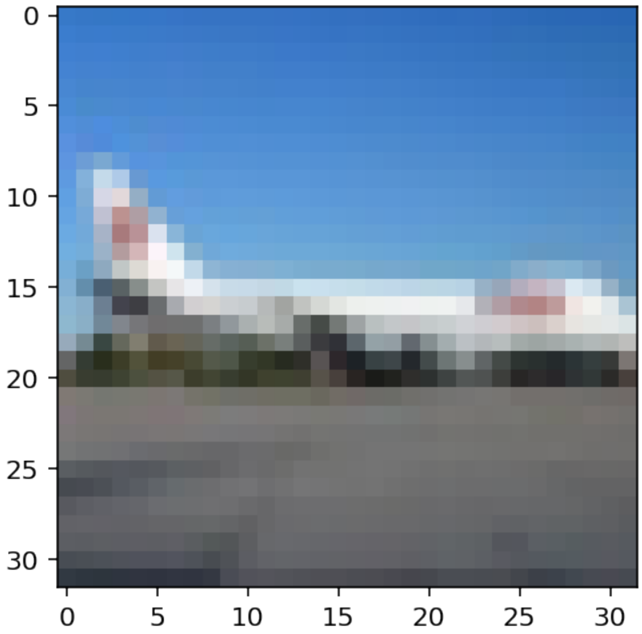
```
def load_label_names():
    return ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog',␣
 ↪'horse', 'ship', 'truck']

def display_stats(cifar10_dataset_folder_path, batch_id, sample_id):
    features, labels = load_cfar10_batch(cifar10_dataset_folder_path, batch_id)

    if not (0 <= sample_id < len(features)):
        print('{} samples in batch {}.  {} is out of range.'.
 ↪format(len(features), batch_id, sample_id))
        return None

    print('\nStats of batch #{}:'.format(batch_id))
    print('# of Samples: {}\n'.format(len(features)))

    label_names = load_label_names()
    label_counts = dict(zip(*np.unique(labels, return_counts=True)))
    for key, value in label_counts.items():
        print('Label Counts of [{}]({}) : {}'.format(key, label_names[key].
 ↪upper(), value))

    sample_image = features[sample_id]
    sample_label = labels[sample_id]

    print('\nExample of Image {}:'.format(sample_id))
    print('Image - Min Value: {} Max Value: {}'.format(sample_image.min(),␣
 ↪sample_image.max()))
    print('Image - Shape: {}'.format(sample_image.shape))
    print('Label - Label Id: {} Name: {}'.format(sample_label,␣
 ↪label_names[sample_label]))

    plt.imshow(sample_image)
```

[4]:

```
%matplotlib inline
%config InlineBackend.figure_format = 'retina'

import numpy as np
import pickle
import matplotlib.pyplot as plt
# Explore the dataset
batch_id = 3
sample_id = 7000
display_stats(cifar10_dataset_folder_path, batch_id, sample_id)
```

```

Stats of batch #3:
# of Samples: 10000
```

```
Label Counts of [0](AIRPLANE) : 994
Label Counts of [1](AUTOMOBILE) : 1042
Label Counts of [2](BIRD) : 965
Label Counts of [3](CAT) : 997
Label Counts of [4](DEER) : 990
Label Counts of [5](DOG) : 1029
Label Counts of [6](FROG) : 978
Label Counts of [7](HORSE) : 1015
Label Counts of [8](SHIP) : 961
Label Counts of [9](TRUCK) : 1029

Example of Image 7000:
Image - Min Value: 24 Max Value: 252
Image - Shape: (32, 32, 3)
Label - Label Id: 0 Name: airplane
```



[5]:
```
import numpy
from tensorflow import keras
from keras.constraints import maxnorm
from keras.utils import np_utils
```

[6]:
```
seed = 21
```

[7]:
```
from keras.datasets import cifar10
```

[8]:
```
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 2s 0us/step
170508288/170498071 [==============================] - 2s 0us/step
```

[9]:
```python
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train = X_train / 255.0
X_test = X_test / 255.0
```

[10]:
```python
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
class_num = y_test.shape[1]
```

[ ]:
```python
model = keras.Sequential()
```

[ ]:
```python
model.add(keras.layers.Conv2D(32, 3, input_shape=(32, 32, 3),␣
 ↪activation='relu', padding='same'))
model.add(keras.layers.Dropout(0.2))
model.add(keras.layers.BatchNormalization())

model.add(keras.layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model.add(keras.layers.MaxPooling2D(2))
model.add(keras.layers.Dropout(0.2))
model.add(keras.layers.BatchNormalization())

model.add(keras.layers.Conv2D(64, 3, padding='same', activation='relu'))
model.add(keras.layers.MaxPooling2D(2))
model.add(keras.layers.Dropout(0.2))
model.add(keras.layers.BatchNormalization())

model.add(keras.layers.Conv2D(128, (3, 3), padding='same', activation='relu'))
model.add(keras.layers.Dropout(0.2))
model.add(keras.layers.BatchNormalization())

model.add(keras.layers.Flatten())
model.add(keras.layers.Dropout(0.2))
model.add(keras.layers.Dense(32, activation='relu'))
model.add(keras.layers.Dropout(0.3))
model.add(keras.layers.BatchNormalization())

model.add(keras.layers.Dense(class_num, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])

print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
```

```
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896       
                                                                 
 dropout (Dropout)           (None, 32, 32, 32)        0         
                                                                 
 batch_normalization (BatchN  (None, 32, 32, 32)       128       
 ormalization)                                                   
                                                                 
 conv2d_1 (Conv2D)           (None, 32, 32, 64)        18496     
                                                                 
 max_pooling2d (MaxPooling2D  (None, 16, 16, 64)       0         
 )                                                               
                                                                 
 dropout_1 (Dropout)         (None, 16, 16, 64)        0         
                                                                 
 batch_normalization_1 (Batc  (None, 16, 16, 64)       256       
 hNormalization)                                                 
                                                                 
 conv2d_2 (Conv2D)           (None, 16, 16, 64)        36928     
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 8, 8, 64)         0         
 2D)                                                             
                                                                 
 dropout_2 (Dropout)         (None, 8, 8, 64)          0         
                                                                 
 batch_normalization_2 (Batc  (None, 8, 8, 64)         256       
 hNormalization)                                                 
                                                                 
 conv2d_3 (Conv2D)           (None, 8, 8, 128)         73856     
                                                                 
 dropout_3 (Dropout)         (None, 8, 8, 128)         0         
                                                                 
 batch_normalization_3 (Batc  (None, 8, 8, 128)        512       
 hNormalization)                                                 
                                                                 
 flatten (Flatten)           (None, 8192)              0         
                                                                 
 dropout_4 (Dropout)         (None, 8192)              0         
                                                                 
 dense (Dense)               (None, 32)                262176    
                                                                 
 dropout_5 (Dropout)         (None, 32)                0         
                                                                 
 batch_normalization_4 (Batc  (None, 32)               128       
 hNormalization)                                                 
                                                                 
 dense_1 (Dense)             (None, 10)                330       
```

```
=================================================================
Total params: 393,962
Trainable params: 393,322
Non-trainable params: 640
_________________________________________________________________
None
```

[ ]:
```
numpy.random.seed(seed)
history = model.fit(X_train, y_train, validation_data=(X_test, y_test),␣
 ↪epochs=25, batch_size=64)
```

```
Epoch 1/25
782/782 [==============================] - 358s 455ms/step - loss: 1.5580 -
accuracy: 0.4473 - val_loss: 1.1479 - val_accuracy: 0.5896
Epoch 2/25
782/782 [==============================] - 347s 443ms/step - loss: 1.1149 -
accuracy: 0.6097 - val_loss: 0.9357 - val_accuracy: 0.6611
Epoch 3/25
782/782 [==============================] - 351s 449ms/step - loss: 0.9590 -
accuracy: 0.6679 - val_loss: 0.8060 - val_accuracy: 0.7178
Epoch 4/25
782/782 [==============================] - 353s 452ms/step - loss: 0.8740 -
accuracy: 0.6980 - val_loss: 0.8086 - val_accuracy: 0.7138
Epoch 5/25
782/782 [==============================] - 376s 480ms/step - loss: 0.8149 -
accuracy: 0.7201 - val_loss: 0.6927 - val_accuracy: 0.7545
Epoch 6/25
782/782 [==============================] - 405s 518ms/step - loss: 0.7618 -
accuracy: 0.7378 - val_loss: 0.7311 - val_accuracy: 0.7489
Epoch 7/25
782/782 [==============================] - 413s 528ms/step - loss: 0.7284 -
accuracy: 0.7493 - val_loss: 0.6366 - val_accuracy: 0.7758
Epoch 8/25
782/782 [==============================] - 415s 531ms/step - loss: 0.6960 -
accuracy: 0.7587 - val_loss: 0.6262 - val_accuracy: 0.7848
Epoch 9/25
782/782 [==============================] - 418s 534ms/step - loss: 0.6628 -
accuracy: 0.7699 - val_loss: 0.5936 - val_accuracy: 0.7958
Epoch 10/25
782/782 [==============================] - 413s 528ms/step - loss: 0.6377 -
accuracy: 0.7791 - val_loss: 0.6802 - val_accuracy: 0.7730
Epoch 11/25
782/782 [==============================] - 411s 525ms/step - loss: 0.6248 -
accuracy: 0.7849 - val_loss: 0.6025 - val_accuracy: 0.7940
Epoch 12/25
782/782 [==============================] - 405s 518ms/step - loss: 0.6003 -
```

```
accuracy: 0.7936 - val_loss: 0.5708 - val_accuracy: 0.8063
Epoch 13/25
782/782 [==============================] - 408s 522ms/step - loss: 0.5918 -
accuracy: 0.7934 - val_loss: 0.5504 - val_accuracy: 0.8103
Epoch 14/25
782/782 [==============================] - 407s 520ms/step - loss: 0.5711 -
accuracy: 0.8019 - val_loss: 0.5959 - val_accuracy: 0.8007
Epoch 15/25
782/782 [==============================] - 407s 521ms/step - loss: 0.5567 -
accuracy: 0.8059 - val_loss: 0.6220 - val_accuracy: 0.7941
Epoch 16/25
782/782 [==============================] - 408s 522ms/step - loss: 0.5440 -
accuracy: 0.8111 - val_loss: 0.5609 - val_accuracy: 0.8120
Epoch 17/25
782/782 [==============================] - 408s 521ms/step - loss: 0.5361 -
accuracy: 0.8115 - val_loss: 0.5451 - val_accuracy: 0.8164
Epoch 18/25
782/782 [==============================] - 410s 524ms/step - loss: 0.5218 -
accuracy: 0.8208 - val_loss: 0.5411 - val_accuracy: 0.8141
Epoch 19/25
782/782 [==============================] - 401s 512ms/step - loss: 0.5180 -
accuracy: 0.8183 - val_loss: 0.5382 - val_accuracy: 0.8180
Epoch 20/25
782/782 [==============================] - 398s 509ms/step - loss: 0.5098 -
accuracy: 0.8227 - val_loss: 0.5175 - val_accuracy: 0.8298
Epoch 21/25
782/782 [==============================] - 403s 516ms/step - loss: 0.5032 -
accuracy: 0.8247 - val_loss: 0.5388 - val_accuracy: 0.8203
Epoch 22/25
782/782 [==============================] - 408s 521ms/step - loss: 0.4882 -
accuracy: 0.8303 - val_loss: 0.5200 - val_accuracy: 0.8313
Epoch 23/25
782/782 [==============================] - 412s 527ms/step - loss: 0.4775 -
accuracy: 0.8333 - val_loss: 0.5109 - val_accuracy: 0.8326
Epoch 24/25
782/782 [==============================] - 412s 527ms/step - loss: 0.4747 -
accuracy: 0.8329 - val_loss: 0.5201 - val_accuracy: 0.8260
Epoch 25/25
782/782 [==============================] - 419s 536ms/step - loss: 0.4669 -
accuracy: 0.8361 - val_loss: 0.5204 - val_accuracy: 0.8273
```

[ ]:

```
model.evaluate(X_test, y_test, verbose=0)
```

[ ]:

```
import pandas as pd
import matplotlib.pyplot as plt

pd.DataFrame(history.history).plot()
plt.show()
```

[ ]:
```
model.save("model.h5")
print("Saved model to disk")
```

1.2 Results of the Model

Model scored 82.73% accuracy in recognizing pictures from CIFAR-10 data-set.

[12]:
```
# load and evaluate a saved model
from numpy import loadtxt
from keras.models import load_model

# load model
model = load_model('model.h5')
# summarize model.
model.summary()
# evaluate the model
score = model.evaluate(X_test, y_test, verbose=0)
print("%s: %.2f%%" % (model.metrics_names[1], score[1]*100))
```

```
Model: "sequential"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 dropout (Dropout)           (None, 32, 32, 32)        0
```

```
 batch_normalization (BatchN  (None, 32, 32, 32)       128
 ormalization)

 conv2d_1 (Conv2D)           (None, 32, 32, 64)       18496

 max_pooling2d (MaxPooling2D  (None, 16, 16, 64)       0
 )

 dropout_1 (Dropout)         (None, 16, 16, 64)       0

 batch_normalization_1 (Batc  (None, 16, 16, 64)       256
 hNormalization)

 conv2d_2 (Conv2D)           (None, 16, 16, 64)       36928

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 dropout_2 (Dropout)         (None, 8, 8, 64)         0

 batch_normalization_2 (Batc  (None, 8, 8, 64)         256
 hNormalization)

 conv2d_3 (Conv2D)           (None, 8, 8, 128)        73856

 dropout_3 (Dropout)         (None, 8, 8, 128)        0

 batch_normalization_3 (Batc  (None, 8, 8, 128)        512
 hNormalization)

 flatten (Flatten)           (None, 8192)             0

 dropout_4 (Dropout)         (None, 8192)             0

 dense (Dense)               (None, 32)               262176

 dropout_5 (Dropout)         (None, 32)               0

 batch_normalization_4 (Batc  (None, 32)               128
 hNormalization)

 dense_1 (Dense)             (None, 10)               330

=================================================================
Total params: 393,962
Trainable params: 393,322
Non-trainable params: 640
```

```
-----------------------------------------------------------------
accuracy: 82.73%
```

2 Generate Documentation

[ ]:

```
!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
from colab_pdf import colab_pdf
colab_pdf('Cifar10.ipynb')
```

[ ]:

[ ]: