03_cifar10_image_classification

September 29, 2021

1 CIFAR10 Image Classification

Fast-forward to 2012, and we move on to the deeper and more modern AlexNet architecture. We will use the CIFAR10 dataset that uses 60,000 ImageNet samples, compressed to 32x32 pixel resolution (from the original 224x224), but still with three color channels. There are only 10 of the original 1,000 classes. See the notebook cifar10_image_classification for implementation details; we will skip here over some repetitive steps.

1.1 Run inside docker container for GPU acceleration

See tensorflow guide and more detailed instructions

```
docker run -it -p 8889:8888 -v /path/to/machine-learning-for-trading/17_convolutions_neural_ne
--name tensorflow tensorflow/tensorflow:latest-gpu-py3 bash
```

Inside docker container: `jupyter notebook --ip 0.0.0.0 --no-browser --allow-root`

1.2 Imports

[1]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

import keras
from keras.utils import np_utils
from keras.datasets import cifar10
from keras.models import Sequential
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Dense, Dropout, Activation, Flatten, Conv2D,␣
 ↪MaxPooling2D
from keras.callbacks import ModelCheckpoint, TensorBoard
from keras.layers.normalization import BatchNormalization
from keras import backend as K
```

```
Using TensorFlow backend.
```

[4]:
```
np.random.seed(42)
```

1.3 Load CIFAR-10 Data

CIFAR10 can also be downloaded from keras, and we similarly rescale the pixel values and one-hot encode the ten class labels.

[5]:
```python
# load the pre-shuffled train and test data
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
```

1.3.1 Visualize the First 30 Training Images

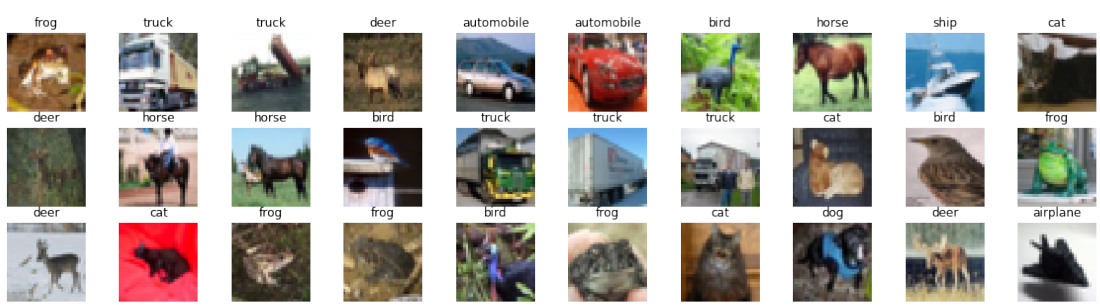
[6]:
```python
cifar10_labels = {0: 'airplane',
                  1: 'automobile',
                  2: 'bird',
                  3: 'cat',
                  4: 'deer',
                  5: 'dog',
                  6: 'frog',
                  7: 'horse',
                  8: 'ship',
                  9: 'truck'}
```

[7]:
```python
num_classes = len(cifar10_labels)
```

[8]:
```python
height, width, channels = X_train.shape[1:]
input_shape = height, width, channels
input_shape
```

[8]: (32, 32, 3)

[7]:
```python
fig, axes = plt.subplots(nrows=3, ncols=10, figsize=(20, 5))
axes = axes.flatten()
for i, ax in enumerate(axes):
    ax.imshow(np.squeeze(X_train[i]))
    ax.axis('off')
    ax.set_title(cifar10_labels[y_train[i, 0]])
```



1.3.2 Rescale the Images

[9]:
```
# rescale [0,255] --> [0,1]
X_train = X_train.astype('float32')/255
X_test = X_test.astype('float32')/255
```

1.3.3 One-hot label encoding

[10]:
```
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

1.3.4 Train-Test split

[11]:
```
X_train, X_valid = X_train[5000:], X_train[:5000]
y_train, y_valid = y_train[5000:], y_train[:5000]
```

[12]:
```
# shape of training set
X_train.shape
```

[12]:
```
(45000, 32, 32, 3)
```

[13]:
```
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')
print(X_valid.shape[0], 'validation samples')
```

```
45000 train samples
10000 test samples
5000 validation samples
```

1.4 Feedforward Neural Network

We first train a two-layer feedforward network on 50,000 training samples for training for 20 epochs to achieve a test accuracy of 44.22%. We also experiment with a three-layer convolutional net with 500K parameters for 67.07% test accuracy.

1.4.1 Model Architecture

[31]:
```
mlp = Sequential([
    Flatten(input_shape=input_shape, name='input'),
    Dense(1000, activation='relu', name='hidden_layer_1'),
    Dropout(0.2, name='droput_1'),
    Dense(512, activation='relu', name='hidden_layer_2'),
    Dropout(0.2, name='dropout_2'),
    Dense(num_classes, activation='softmax', name='output')
])
```

[32]:
```
mlp.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input (Flatten)              (None, 3072)              0
_________________________________________________________________
hidden_layer_1 (Dense)       (None, 1000)              3073000
_________________________________________________________________
droput_1 (Dropout)           (None, 1000)              0
_________________________________________________________________
hidden_layer_2 (Dense)       (None, 512)               512512
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
output (Dense)               (None, 10)                5130
=================================================================
Total params: 3,590,642
Trainable params: 3,590,642
Non-trainable params: 0
_________________________________________________________________
```

1.4.2 Compile the Model

[33]:
```
mlp.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

1.4.3 Train the Model

[34]:
```
mlp_path = 'models/cifar10.mlp.weights.best.hdf5'
```

[35]:
```
checkpointer = ModelCheckpoint(filepath=mlp_path,
                               verbose=1,
                               save_best_only=True)
```

[36]:
```
tensorboard = TensorBoard(log_dir='./logs/mlp',
                          histogram_freq=1,
                          batch_size=32,
                          write_graph=True,
                          write_grads=False,
                          update_freq='epoch')
```

[19]:
```
hist = mlp.fit(X_train,
               y_train,
               batch_size=32,
               epochs=20,
               validation_data=(x_valid, y_valid),
               callbacks=[checkpointer, tensorboard],
```

```
                verbose=2,
                shuffle=True)
```

```
Train on 45000 samples, validate on 5000 samples
Epoch 1/20
 - 4s - loss: 1.9870 - acc: 0.2702 - val_loss: 1.8040 - val_acc: 0.3470

Epoch 00001: val_loss improved from inf to 1.80397, saving model to
cifar10.mlp.weights.best.hdf5
Epoch 2/20
 - 4s - loss: 1.8563 - acc: 0.3209 - val_loss: 1.7918 - val_acc: 0.3574

Epoch 00002: val_loss improved from 1.80397 to 1.79179, saving model to
cifar10.mlp.weights.best.hdf5
Epoch 3/20
 - 4s - loss: 1.8163 - acc: 0.3374 - val_loss: 1.7227 - val_acc: 0.3746

Epoch 00003: val_loss improved from 1.79179 to 1.72267, saving model to
cifar10.mlp.weights.best.hdf5
Epoch 4/20
 - 4s - loss: 1.7955 - acc: 0.3450 - val_loss: 1.7177 - val_acc: 0.3956

Epoch 00004: val_loss improved from 1.72267 to 1.71774, saving model to
cifar10.mlp.weights.best.hdf5
Epoch 5/20
 - 4s - loss: 1.7679 - acc: 0.3573 - val_loss: 1.6933 - val_acc: 0.3926

Epoch 00005: val_loss improved from 1.71774 to 1.69330, saving model to
cifar10.mlp.weights.best.hdf5
Epoch 6/20
 - 4s - loss: 1.7540 - acc: 0.3613 - val_loss: 1.7462 - val_acc: 0.3840

Epoch 00006: val_loss did not improve from 1.69330
Epoch 7/20
 - 4s - loss: 1.7410 - acc: 0.3658 - val_loss: 1.6759 - val_acc: 0.3918

Epoch 00007: val_loss improved from 1.69330 to 1.67594, saving model to
cifar10.mlp.weights.best.hdf5
Epoch 8/20
 - 4s - loss: 1.7207 - acc: 0.3769 - val_loss: 1.6777 - val_acc: 0.4106

Epoch 00008: val_loss did not improve from 1.67594
Epoch 9/20
 - 4s - loss: 1.7106 - acc: 0.3833 - val_loss: 1.6812 - val_acc: 0.4034

Epoch 00009: val_loss did not improve from 1.67594
Epoch 10/20
```

```
 - 4s - loss: 1.7052 - acc: 0.3863 - val_loss: 1.6236 - val_acc: 0.4274

Epoch 00010: val_loss improved from 1.67594 to 1.62361, saving model to
cifar10.mlp.weights.best.hdf5
Epoch 11/20
 - 4s - loss: 1.6923 - acc: 0.3895 - val_loss: 1.6313 - val_acc: 0.4120

Epoch 00011: val_loss did not improve from 1.62361
Epoch 12/20
 - 4s - loss: 1.6843 - acc: 0.3922 - val_loss: 1.6428 - val_acc: 0.4130

Epoch 00012: val_loss did not improve from 1.62361
Epoch 13/20
 - 4s - loss: 1.6801 - acc: 0.3975 - val_loss: 1.6125 - val_acc: 0.4276

Epoch 00013: val_loss improved from 1.62361 to 1.61248, saving model to
cifar10.mlp.weights.best.hdf5
Epoch 14/20
 - 4s - loss: 1.6706 - acc: 0.3990 - val_loss: 1.6525 - val_acc: 0.4266

Epoch 00014: val_loss did not improve from 1.61248
Epoch 15/20
 - 4s - loss: 1.6693 - acc: 0.4014 - val_loss: 1.6088 - val_acc: 0.4218

Epoch 00015: val_loss improved from 1.61248 to 1.60883, saving model to
cifar10.mlp.weights.best.hdf5
Epoch 16/20
 - 4s - loss: 1.6572 - acc: 0.4046 - val_loss: 1.5909 - val_acc: 0.4412

Epoch 00016: val_loss improved from 1.60883 to 1.59088, saving model to
cifar10.mlp.weights.best.hdf5
Epoch 17/20
 - 4s - loss: 1.6548 - acc: 0.4044 - val_loss: 1.6040 - val_acc: 0.4360

Epoch 00017: val_loss did not improve from 1.59088
Epoch 18/20
 - 4s - loss: 1.6537 - acc: 0.4052 - val_loss: 1.5921 - val_acc: 0.4322

Epoch 00018: val_loss did not improve from 1.59088
Epoch 19/20
 - 4s - loss: 1.6469 - acc: 0.4062 - val_loss: 1.5967 - val_acc: 0.4416

Epoch 00019: val_loss did not improve from 1.59088
Epoch 20/20
 - 4s - loss: 1.6406 - acc: 0.4103 - val_loss: 1.5899 - val_acc: 0.4370

Epoch 00020: val_loss improved from 1.59088 to 1.58988, saving model to
cifar10.mlp.weights.best.hdf5
```

1.4.4 Load best model

[37]:
```
# load the weights that yielded the best validation accuracy
mlp.load_weights(mlp_path)
```

1.4.5 Test Classification Accuracy

[38]:
```
# evaluate and print test accuracy
accuracy = mlp.evaluate(X_test, y_test, verbose=0)[1]
print('Test accuracy: {:.2%}'.format(accuracy))
```

```
Test accuracy: 44.22%
```

1.5 Convolutional Neural Network

[39]:
```
# https://stackoverflow.com/questions/35114376/
↪error-when-computing-summaries-in-tensorflow/35117760#35117760
K.clear_session()
```

1.5.1 Model Architecture

[44]:
```
cnn = Sequential([
    Conv2D(filters=16, kernel_size=2, padding='same',
           activation='relu', input_shape=input_shape, name='CONV1'),
    MaxPooling2D(pool_size=2, name='POOL1'),
    Conv2D(filters=32, kernel_size=2, padding='same', activation='relu',␣
↪name='CONV2'),
    MaxPooling2D(pool_size=2, name='POOL2'),
    Conv2D(filters=64, kernel_size=2, padding='same', activation='relu',␣
↪name='CONV3'),
    MaxPooling2D(pool_size=2, name='POOL3'),
    Dropout(0.3, name='DROP1'),
    Flatten(name='FLAT1'),
    Dense(500, activation='relu', name='FC1'),
    Dropout(0.4, name='DROP2'),
    Dense(10, activation='softmax', name='FC2')]
)
```

[45]:
```
cnn.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
CONV1 (Conv2D)               (None, 32, 32, 16)        208
_________________________________________________________________
POOL1 (MaxPooling2D)         (None, 16, 16, 16)        0
_________________________________________________________________
```

```
CONV2 (Conv2D)               (None, 16, 16, 32)        2080
_________________________________________________________________
POOL2 (MaxPooling2D)         (None, 8, 8, 32)          0
_________________________________________________________________
CONV3 (Conv2D)               (None, 8, 8, 64)          8256
_________________________________________________________________
POOL3 (MaxPooling2D)         (None, 4, 4, 64)          0
_________________________________________________________________
DROP1 (Dropout)              (None, 4, 4, 64)          0
_________________________________________________________________
FLAT1 (Flatten)              (None, 1024)              0
_________________________________________________________________
FC1 (Dense)                  (None, 500)               512500
_________________________________________________________________
DROP2 (Dropout)              (None, 500)               0
_________________________________________________________________
FC2 (Dense)                  (None, 10)                5010
=================================================================
Total params: 528,054
Trainable params: 528,054
Non-trainable params: 0
_________________________________________________________________
```

1.5.2 Compile the Model

[50]:
```
cnn.compile(loss='categorical_crossentropy',
            optimizer='adam',
            metrics=['accuracy'])
```

1.5.3 Train the Model

[47]:
```
cnn_path = 'models/cifar10.cnn.weights.best.hdf5'
```

[40]:
```
checkpointer = ModelCheckpoint(filepath=cnn_path,
                               verbose=1,
                               save_best_only=True)
```

[41]:
```
tensorboard = TensorBoard(log_dir='./logs/cnn',
                          histogram_freq=1,
                          batch_size=32,
                          write_graph=True,
                          write_grads=False,
                          update_freq='epoch')
```

[42]:
```
hist = cnn.fit(x_train,
               y_train,
               batch_size=32,
```

```
                 epochs=20,
                 validation_data=(x_valid, y_valid),
                 callbacks=[checkpointer, tensorboard],
                 verbose=2,
                 shuffle=True)
```

```
Train on 45000 samples, validate on 5000 samples
Epoch 1/20
 - 3s - loss: 1.5899 - acc: 0.4226 - val_loss: 1.4604 - val_acc: 0.4636

Epoch 00001: val_loss improved from inf to 1.46040, saving model to
weights/cifar10.cnn.weights.best.hdf5
Epoch 2/20
 - 3s - loss: 1.2710 - acc: 0.5473 - val_loss: 1.2745 - val_acc: 0.5438

Epoch 00002: val_loss improved from 1.46040 to 1.27447, saving model to
weights/cifar10.cnn.weights.best.hdf5
Epoch 3/20
 - 3s - loss: 1.1529 - acc: 0.5928 - val_loss: 1.0080 - val_acc: 0.6506

Epoch 00003: val_loss improved from 1.27447 to 1.00804, saving model to
weights/cifar10.cnn.weights.best.hdf5
Epoch 4/20
 - 3s - loss: 1.0868 - acc: 0.6182 - val_loss: 1.0180 - val_acc: 0.6418

Epoch 00004: val_loss did not improve from 1.00804
Epoch 5/20
 - 3s - loss: 1.0414 - acc: 0.6385 - val_loss: 1.0208 - val_acc: 0.6478

Epoch 00005: val_loss did not improve from 1.00804
Epoch 6/20
 - 3s - loss: 1.0153 - acc: 0.6463 - val_loss: 0.9969 - val_acc: 0.6610

Epoch 00006: val_loss improved from 1.00804 to 0.99688, saving model to
weights/cifar10.cnn.weights.best.hdf5
Epoch 7/20
 - 3s - loss: 0.9978 - acc: 0.6548 - val_loss: 0.9361 - val_acc: 0.6750

Epoch 00007: val_loss improved from 0.99688 to 0.93611, saving model to
weights/cifar10.cnn.weights.best.hdf5
Epoch 8/20
 - 3s - loss: 0.9966 - acc: 0.6605 - val_loss: 0.9450 - val_acc: 0.6728

Epoch 00008: val_loss did not improve from 0.93611
Epoch 9/20
 - 3s - loss: 0.9863 - acc: 0.6662 - val_loss: 0.9678 - val_acc: 0.6760
```

```
Epoch 00009: val_loss did not improve from 0.93611
Epoch 10/20
 - 3s - loss: 0.9800 - acc: 0.6668 - val_loss: 1.1120 - val_acc: 0.6264

Epoch 00010: val_loss did not improve from 0.93611
Epoch 11/20
 - 3s - loss: 0.9862 - acc: 0.6672 - val_loss: 1.0151 - val_acc: 0.6590

Epoch 00011: val_loss did not improve from 0.93611
Epoch 12/20
 - 3s - loss: 0.9955 - acc: 0.6656 - val_loss: 1.0753 - val_acc: 0.6432

Epoch 00012: val_loss did not improve from 0.93611
Epoch 13/20
 - 3s - loss: 1.0046 - acc: 0.6668 - val_loss: 1.2666 - val_acc: 0.6052

Epoch 00013: val_loss did not improve from 0.93611
Epoch 14/20
 - 3s - loss: 1.0156 - acc: 0.6590 - val_loss: 1.3022 - val_acc: 0.5702

Epoch 00014: val_loss did not improve from 0.93611
Epoch 15/20
 - 3s - loss: 1.0317 - acc: 0.6593 - val_loss: 1.1539 - val_acc: 0.6310

Epoch 00015: val_loss did not improve from 0.93611
Epoch 16/20
 - 3s - loss: 1.0442 - acc: 0.6523 - val_loss: 1.0976 - val_acc: 0.6490

Epoch 00016: val_loss did not improve from 0.93611
Epoch 17/20
 - 3s - loss: 1.0782 - acc: 0.6472 - val_loss: 1.1564 - val_acc: 0.6316

Epoch 00017: val_loss did not improve from 0.93611
Epoch 18/20
 - 3s - loss: 1.0837 - acc: 0.6435 - val_loss: 0.9460 - val_acc: 0.6888

Epoch 00018: val_loss did not improve from 0.93611
Epoch 19/20
 - 3s - loss: 1.1012 - acc: 0.6361 - val_loss: 1.0408 - val_acc: 0.6428

Epoch 00019: val_loss did not improve from 0.93611
Epoch 20/20
 - 3s - loss: 1.1142 - acc: 0.6329 - val_loss: 1.6174 - val_acc: 0.6362

Epoch 00020: val_loss did not improve from 0.93611
```

1.5.4 Load best model

[51]:
```
cnn.load_weights(cnn_path)
```

1.5.5 Test set accuracy

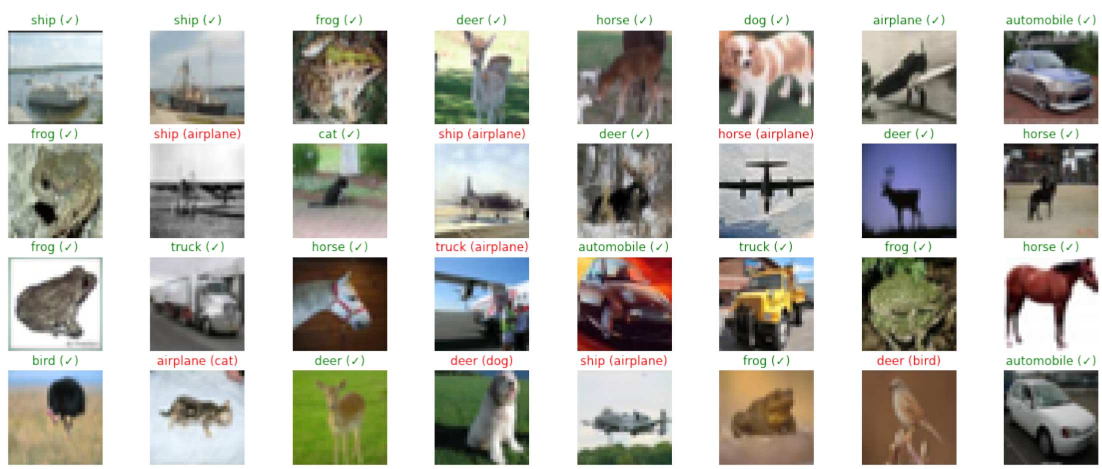
[52]:
```
accuracy = cnn.evaluate(x_test, y_test, verbose=0)[1]
print('Accuracy: {:.2%}'.format(accuracy))
```

```
Accuracy: 67.07%
```

1.5.6 Evaluate Predictions

[48]:
```
y_hat = cnn.predict(x_test)
```

[60]:
```
fig, axes = plt.subplots(nrows=4, ncols=8, figsize=(20, 8))
axes = axes.flatten()
images = np.random.choice(x_test.shape[0], size=32, replace=False)
for i, (ax, idx) in enumerate(zip(axes, images)):
    ax.imshow(np.squeeze(x_test[idx]))
    ax.axis('off')
    pred_idx, true_idx = np.argmax(y_hat[idx]), np.argmax(y_test[idx])
    if pred_idx == true_idx:
        ax.set_title('{} ( )'.format(cifar10_labels[pred_idx]), color="green")
    else:
        ax.set_title("{} ({})".format(cifar10_labels[pred_idx],␣
 ↪cifar10_labels[true_idx]), color='red')
```



1.6 CNN with Image Augmentation

A common trick to enhance performance is to artificially increase the size of the training set by creating synthetic data. This involves randomly shifting or horizontally flipping the image, or introducing noise into the image.

1.6.1 Create and configure augmented image generator

Keras includes an ImageDataGenerator for this purpose that we can configure and fit to the training data as follows:

[14]:
```
datagen = ImageDataGenerator(
    width_shift_range=0.1,  # randomly horizontal shift
    height_shift_range=0.1,  # randomly vertial shift
    horizontal_flip=True)  # randomly horizontalflip
```

[15]:
```
# fit augmented image generator on data
datagen.fit(X_train)
```

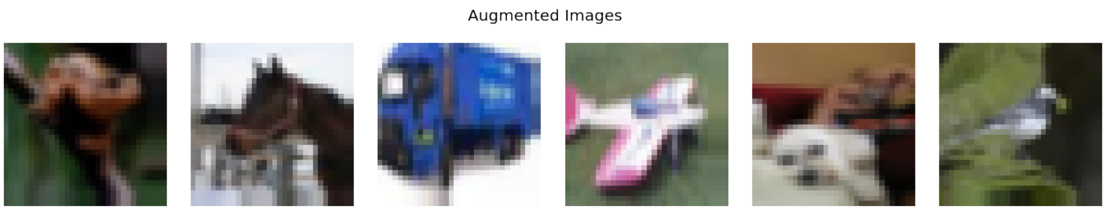
1.6.2 Visualize subset of training data

The result shows how the augmented images have been altered in various ways as expected:

[25]:
```
n_images = 6
x_train_subset = X_train[:n_images]
```

[29]:
```
# original images
fig, axes = plt.subplots(nrows=1, ncols=n_images, figsize=(20,4))
for i, (ax, img) in enumerate(zip(axes, x_train_subset)):
    ax.imshow(img)
    ax.axis('off')
fig.suptitle('Subset of Original Training Images', fontsize=20)
fig.tight_layout()
fig.subplots_adjust(top=.9)
fig.savefig('images/original')

# augmented images
fig, axes = plt.subplots(nrows=1, ncols=n_images, figsize=(20,4))
for x_batch in datagen.flow(x_train_subset, batch_size=n_images, shuffle=False):
    for i, ax in enumerate(axes):
        ax.imshow(x_batch[i])
        ax.axis('off')
#     fig.suptitle('Augmented Images', fontsize=20)
    break;
fig.suptitle('Augmented Images', fontsize=20);
fig.tight_layout()
fig.subplots_adjust(top=.9)
fig.savefig('images/augmented')
```



1.6.3 Train Augmented Images

[57]:
```
K.clear_session()
```

[58]:
```
cnn_aug_path = 'models/cifar10.augmented.cnn.weights.best.hdf5'
```

[59]:
```
checkpointer = ModelCheckpoint(filepath=cnn_aug_path,
                               verbose=1,
                               save_best_only=True)
```

[60]:
```
tensorboard = TensorBoard(log_dir='./logs/cnn_aug',
                          histogram_freq=1,
                          batch_size=32,
                          write_graph=True,
                          write_grads=False,
                          update_freq='epoch')
```

[64]:
```
cnn = Sequential([
    Conv2D(filters=16, kernel_size=2, padding='same',
           activation='relu', input_shape=input_shape, name='CONV1'),
    MaxPooling2D(pool_size=2, name='POOL1'),
    Conv2D(filters=32, kernel_size=2, padding='same', activation='relu',␣
 ↪name='CONV2'),
    MaxPooling2D(pool_size=2, name='POOL2'),
    Conv2D(filters=64, kernel_size=2, padding='same', activation='relu',␣
 ↪name='CONV3'),
    MaxPooling2D(pool_size=2, name='POOL3'),
    Dropout(0.3, name='DROP1'),
```

```
    Flatten(name='FLAT1'),
    Dense(500, activation='relu', name='FC1'),
    Dropout(0.4, name='DROP2'),
    Dense(10, activation='softmax', name='FC2')]
)
```

[65]:
```
cnn.compile(loss='categorical_crossentropy',
            optimizer='adam',
            metrics=['accuracy'])
```

[61]:
```
batch_size = 32
epochs = 20
```

[35]:
```
hist = cnn.fit_generator(datagen.flow(x_train, y_train, batch_size=batch_size),
                         steps_per_epoch=x_train.shape[0] // batch_size,
                         epochs=epochs,
                         validation_data=(x_valid, y_valid),
                         callbacks=[checkpointer, tensorboard],
                         verbose=2)
```

```
Epoch 1/20
 - 11s - loss: 1.6158 - acc: 0.4087 - val_loss: 1.3774 - val_acc: 0.5012

Epoch 00001: val_loss improved from inf to 1.37738, saving model to
weights/cifar10.augmented.cnn.weights.best.hdf5
Epoch 2/20
 - 11s - loss: 1.3582 - acc: 0.5123 - val_loss: 1.1724 - val_acc: 0.5846

Epoch 00002: val_loss improved from 1.37738 to 1.17241, saving model to
weights/cifar10.augmented.cnn.weights.best.hdf5
Epoch 3/20
 - 11s - loss: 1.2461 - acc: 0.5545 - val_loss: 1.0761 - val_acc: 0.6208

Epoch 00003: val_loss improved from 1.17241 to 1.07606, saving model to
weights/cifar10.augmented.cnn.weights.best.hdf5
Epoch 4/20
 - 11s - loss: 1.1795 - acc: 0.5782 - val_loss: 1.0266 - val_acc: 0.6334

Epoch 00004: val_loss improved from 1.07606 to 1.02659, saving model to
weights/cifar10.augmented.cnn.weights.best.hdf5
Epoch 5/20
 - 11s - loss: 1.1184 - acc: 0.6008 - val_loss: 0.9402 - val_acc: 0.6626

Epoch 00005: val_loss improved from 1.02659 to 0.94016, saving model to
weights/cifar10.augmented.cnn.weights.best.hdf5
Epoch 6/20
 - 11s - loss: 1.0792 - acc: 0.6180 - val_loss: 0.9342 - val_acc: 0.6706
```

```
Epoch 00006: val_loss improved from 0.94016 to 0.93417, saving model to
weights/cifar10.augmented.cnn.weights.best.hdf5
Epoch 7/20
 - 11s - loss: 1.0535 - acc: 0.6270 - val_loss: 0.9041 - val_acc: 0.6808

Epoch 00007: val_loss improved from 0.93417 to 0.90409, saving model to
weights/cifar10.augmented.cnn.weights.best.hdf5
Epoch 8/20
 - 11s - loss: 1.0215 - acc: 0.6392 - val_loss: 0.8475 - val_acc: 0.7146

Epoch 00008: val_loss improved from 0.90409 to 0.84751, saving model to
weights/cifar10.augmented.cnn.weights.best.hdf5
Epoch 9/20
 - 11s - loss: 0.9996 - acc: 0.6464 - val_loss: 0.8171 - val_acc: 0.7162

Epoch 00009: val_loss improved from 0.84751 to 0.81715, saving model to
weights/cifar10.augmented.cnn.weights.best.hdf5
Epoch 10/20
 - 11s - loss: 0.9732 - acc: 0.6564 - val_loss: 0.8470 - val_acc: 0.7048

Epoch 00010: val_loss did not improve from 0.81715
Epoch 11/20
 - 11s - loss: 0.9637 - acc: 0.6639 - val_loss: 0.8323 - val_acc: 0.7102

Epoch 00011: val_loss did not improve from 0.81715
Epoch 12/20
 - 11s - loss: 0.9462 - acc: 0.6647 - val_loss: 0.8075 - val_acc: 0.7140

Epoch 00012: val_loss improved from 0.81715 to 0.80745, saving model to
weights/cifar10.augmented.cnn.weights.best.hdf5
Epoch 13/20
 - 11s - loss: 0.9319 - acc: 0.6704 - val_loss: 0.8284 - val_acc: 0.7156

Epoch 00013: val_loss did not improve from 0.80745
Epoch 14/20
 - 11s - loss: 0.9279 - acc: 0.6745 - val_loss: 0.8644 - val_acc: 0.6886

Epoch 00014: val_loss did not improve from 0.80745
Epoch 15/20
 - 11s - loss: 0.9077 - acc: 0.6787 - val_loss: 0.7595 - val_acc: 0.7368

Epoch 00015: val_loss improved from 0.80745 to 0.75952, saving model to
weights/cifar10.augmented.cnn.weights.best.hdf5
Epoch 16/20
 - 10s - loss: 0.9053 - acc: 0.6813 - val_loss: 0.7495 - val_acc: 0.7382

Epoch 00016: val_loss improved from 0.75952 to 0.74949, saving model to
```

```
weights/cifar10.augmented.cnn.weights.best.hdf5
Epoch 17/20
 - 11s - loss: 0.8983 - acc: 0.6842 - val_loss: 0.7426 - val_acc: 0.7404

Epoch 00017: val_loss improved from 0.74949 to 0.74256, saving model to
weights/cifar10.augmented.cnn.weights.best.hdf5
Epoch 18/20
 - 10s - loss: 0.8906 - acc: 0.6847 - val_loss: 0.7792 - val_acc: 0.7308

Epoch 00018: val_loss did not improve from 0.74256
Epoch 19/20
 - 11s - loss: 0.8757 - acc: 0.6931 - val_loss: 0.7281 - val_acc: 0.7566

Epoch 00019: val_loss improved from 0.74256 to 0.72814, saving model to
weights/cifar10.augmented.cnn.weights.best.hdf5
Epoch 20/20
 - 11s - loss: 0.8707 - acc: 0.6938 - val_loss: 0.7143 - val_acc: 0.7598

Epoch 00020: val_loss improved from 0.72814 to 0.71429, saving model to
weights/cifar10.augmented.cnn.weights.best.hdf5
```

1.6.4 Load best model

[66]:
```
cnn.load_weights(cnn_aug_path)
```

1.6.5 Test set accuracy

The test accuracy for the three-layer CNN improves markedly to 74.79% after training on the larger, augmented data.

[67]:
```
accuracy = cnn.evaluate(x_test, y_test, verbose=0)[1]
print('Accuracy: {:.2%}'.format(accuracy))
```

```
Accuracy: 74.79%
```

1.7 AlexNet

We also need to simplify the AlexNet architecture in response to the lower dimensionality of CIFAR10 images relative to the ImageNet samples used in the competition. We use the original number of filters but make them smaller (see notebook for implementation). The summary shows the five convolutional layers followed by two fully-connected layers with frequent use of batch normalization, for a total of 21.5 million parameters:

1.7.1 Define Architecture

[27]:
```
K.clear_session()
```

[28]:
```
alexnet = Sequential([

    # 1st Convolutional Layer
    Conv2D(96, (3,3), strides=(2,2), activation='relu', padding='same',␣
↪input_shape=input_shape, name='CONV_1'),
    MaxPooling2D(pool_size=(2, 2), strides=(2, 2), name='POOL_1'),
    BatchNormalization(name='NORM_1'),

    # 2nd Convolutional Layer
    Conv2D(filters=256, kernel_size=(5, 5), padding='same', activation='relu',␣
↪name='CONV2'),
    MaxPooling2D(pool_size=(3, 3), strides=(2,2), name='POOL2'),
    BatchNormalization(name='NORM_2'),

    # 3rd Convolutional Layer
    Conv2D(filters=384, kernel_size=(3, 3), padding='same', activation='relu',␣
↪name='CONV3'),
    # 4th Convolutional Layer
    Conv2D(filters=384, kernel_size=(3, 3), padding='same', activation='relu',␣
↪name='CONV4'),
    # 5th Convolutional Layer
    Conv2D(filters=256, kernel_size=(3, 3), padding='same', activation='relu',␣
↪name='CONV5'),
    MaxPooling2D(pool_size=(3, 3), strides=(2, 2), name='POOL5'),
    BatchNormalization(name='NORM_5'),

    # Fully Connected Layers
    Flatten(name='FLAT'),
    Dense(4096, input_shape=(32*32*3,), activation='relu', name='FC1'),
    Dropout(0.4, name='DROP1'),
    Dense(4096, activation='relu', name='FC2'),
    Dropout(0.4, name='DROP2'),
    Dense(num_classes, activation='softmax')
])
```

[29]:
```
alexnet.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
CONV_1 (Conv2D)              (None, 16, 16, 96)        2688
_________________________________________________________________
POOL_1 (MaxPooling2D)        (None, 8, 8, 96)          0
_________________________________________________________________
NORM_1 (BatchNormalization)  (None, 8, 8, 96)          384
_________________________________________________________________
CONV2 (Conv2D)               (None, 8, 8, 256)         614656
```

```
_________________________________________________________________
POOL2 (MaxPooling2D)         (None, 3, 3, 256)         0
_________________________________________________________________
NORM_2 (BatchNormalization)  (None, 3, 3, 256)         1024
_________________________________________________________________
CONV3 (Conv2D)               (None, 3, 3, 384)         885120
_________________________________________________________________
CONV4 (Conv2D)               (None, 3, 3, 384)         1327488
_________________________________________________________________
CONV5 (Conv2D)               (None, 3, 3, 256)         884992
_________________________________________________________________
POOL5 (MaxPooling2D)         (None, 1, 1, 256)         0
_________________________________________________________________
NORM_5 (BatchNormalization)  (None, 1, 1, 256)         1024
_________________________________________________________________
FLAT (Flatten)               (None, 256)               0
_________________________________________________________________
FC1 (Dense)                  (None, 4096)              1052672
_________________________________________________________________
DROP1 (Dropout)              (None, 4096)              0
_________________________________________________________________
FC2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
DROP2 (Dropout)              (None, 4096)              0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                40970
=================================================================
Total params: 21,592,330
Trainable params: 21,591,114
Non-trainable params: 1,216
_________________________________________________________________
```

1.7.2 Compile Model

[30]:
```python
alexnet.compile(loss='categorical_crossentropy',
                optimizer='adam',
                metrics=['accuracy'])
```

1.7.3 Train Model

[31]:
```python
batch_size = 32
epochs = 20
```

[32]:
```python
alexnet_path = 'models/cifar10.augmented.alexnet.weights.best.hdf5'
```

[33]:
```python
checkpointer = ModelCheckpoint(filepath=alexnet_path,
                               verbose=1,
```

```
                                    save_best_only=True)
```

[34]:
```
tensorboard = TensorBoard(log_dir='./logs/alexnet',
                          histogram_freq=1,
                          batch_size=batch_size,
                          write_graph=True,
                          write_grads=False,
                          update_freq='epoch')
```

[35]:
```
alexnet.fit_generator(datagen.flow(X_train, y_train, batch_size=batch_size),
                      steps_per_epoch=X_train.shape[0] // batch_size,
                      epochs=epochs,
                      validation_data=(X_valid, y_valid),
                      callbacks=[checkpointer, tensorboard],
                      verbose=1)
```

```
Epoch 1/20
1406/1406 [==============================] - 24s 17ms/step - loss: 1.7045 - acc:
0.3855 - val_loss: 1.5554 - val_acc: 0.4586

Epoch 00001: val_loss improved from inf to 1.55539, saving model to
models/cifar10.augmented.alexnet.weights.best.hdf5
Epoch 2/20
1406/1406 [==============================] - 24s 17ms/step - loss: 1.4137 - acc:
0.5014 - val_loss: 1.5889 - val_acc: 0.4762

Epoch 00002: val_loss did not improve from 1.55539
Epoch 3/20
1406/1406 [==============================] - 24s 17ms/step - loss: 1.2087 - acc:
0.5785 - val_loss: 1.4027 - val_acc: 0.5114

Epoch 00003: val_loss improved from 1.55539 to 1.40270, saving model to
models/cifar10.augmented.alexnet.weights.best.hdf5
Epoch 4/20
1406/1406 [==============================] - 24s 17ms/step - loss: 1.1051 - acc:
0.6204 - val_loss: 1.1670 - val_acc: 0.6126

Epoch 00004: val_loss improved from 1.40270 to 1.16702, saving model to
models/cifar10.augmented.alexnet.weights.best.hdf5
Epoch 5/20
1406/1406 [==============================] - 24s 17ms/step - loss: 1.0394 - acc:
0.6442 - val_loss: 0.9912 - val_acc: 0.6594

Epoch 00005: val_loss improved from 1.16702 to 0.99116, saving model to
models/cifar10.augmented.alexnet.weights.best.hdf5
Epoch 6/20
1406/1406 [==============================] - 24s 17ms/step - loss: 0.9506 - acc:
```

```
0.6781 - val_loss: 0.9463 - val_acc: 0.6726

Epoch 00006: val_loss improved from 0.99116 to 0.94629, saving model to
models/cifar10.augmented.alexnet.weights.best.hdf5
Epoch 7/20
1406/1406 [==============================] - 24s 17ms/step - loss: 0.8909 - acc:
0.6998 - val_loss: 0.8934 - val_acc: 0.6930

Epoch 00007: val_loss improved from 0.94629 to 0.89337, saving model to
models/cifar10.augmented.alexnet.weights.best.hdf5
Epoch 8/20
1406/1406 [==============================] - 24s 17ms/step - loss: 0.8525 - acc:
0.7154 - val_loss: 0.8005 - val_acc: 0.7336

Epoch 00008: val_loss improved from 0.89337 to 0.80054, saving model to
models/cifar10.augmented.alexnet.weights.best.hdf5
Epoch 9/20
1406/1406 [==============================] - 24s 17ms/step - loss: 0.7979 - acc:
0.7314 - val_loss: 0.9394 - val_acc: 0.6994

Epoch 00009: val_loss did not improve from 0.80054
Epoch 10/20
1406/1406 [==============================] - 24s 17ms/step - loss: 0.7739 - acc:
0.7417 - val_loss: 0.8567 - val_acc: 0.7094

Epoch 00010: val_loss did not improve from 0.80054
Epoch 11/20
1406/1406 [==============================] - 24s 17ms/step - loss: 0.7395 - acc:
0.7523 - val_loss: 0.8152 - val_acc: 0.7286

Epoch 00011: val_loss did not improve from 0.80054
Epoch 12/20
1406/1406 [==============================] - 24s 17ms/step - loss: 0.7138 - acc:
0.7626 - val_loss: 0.7294 - val_acc: 0.7544

Epoch 00012: val_loss improved from 0.80054 to 0.72942, saving model to
models/cifar10.augmented.alexnet.weights.best.hdf5
Epoch 13/20
1406/1406 [==============================] - 24s 17ms/step - loss: 0.6673 - acc:
0.7772 - val_loss: 0.7657 - val_acc: 0.7456

Epoch 00013: val_loss did not improve from 0.72942
Epoch 14/20
1406/1406 [==============================] - 23s 16ms/step - loss: 0.6710 - acc:
0.7772 - val_loss: 0.8086 - val_acc: 0.7388

Epoch 00014: val_loss did not improve from 0.72942
Epoch 15/20
```

```
1406/1406 [==============================] - 23s 16ms/step - loss: 0.6272 - acc:
0.7897 - val_loss: 0.6494 - val_acc: 0.7820

Epoch 00015: val_loss improved from 0.72942 to 0.64940, saving model to
models/cifar10.augmented.alexnet.weights.best.hdf5
Epoch 16/20
1406/1406 [==============================] - 23s 16ms/step - loss: 0.6164 - acc:
0.7949 - val_loss: 1.5029 - val_acc: 0.6598

Epoch 00016: val_loss did not improve from 0.64940
Epoch 17/20
1406/1406 [==============================] - 23s 16ms/step - loss: 0.5968 - acc:
0.8044 - val_loss: 0.7309 - val_acc: 0.7574

Epoch 00017: val_loss did not improve from 0.64940
Epoch 18/20
1406/1406 [==============================] - 23s 16ms/step - loss: 0.6588 - acc:
0.7818 - val_loss: 0.6926 - val_acc: 0.7852

Epoch 00018: val_loss did not improve from 0.64940
Epoch 19/20
1406/1406 [==============================] - 23s 16ms/step - loss: 0.5598 - acc:
0.8152 - val_loss: 0.8458 - val_acc: 0.7644

Epoch 00019: val_loss did not improve from 0.64940
Epoch 20/20
1406/1406 [==============================] - 23s 16ms/step - loss: 0.5492 - acc:
0.8179 - val_loss: 0.6891 - val_acc: 0.7756

Epoch 00020: val_loss did not improve from 0.64940
```

[35]:
```
<keras.callbacks.History at 0x7fda96ce4eb8>
```

[36]:
```python
alexnet.load_weights(alexnet_path)
```

After training for 20 episodes, each of which takes a little under 30 seconds on a single GPU, we obtain 76.84% test accuracy.

[37]:
```python
accuracy = alexnet.evaluate(X_test, y_test, verbose=0)[1]
print('Accuracy: {:.2%}'.format(accuracy))
```

```
Accuracy: 76.84%
```

[ ]:
```
```