Data visualization

COSC 480B

Reyan Ahmed

rahmed1@colgate.edu

Lecture 24

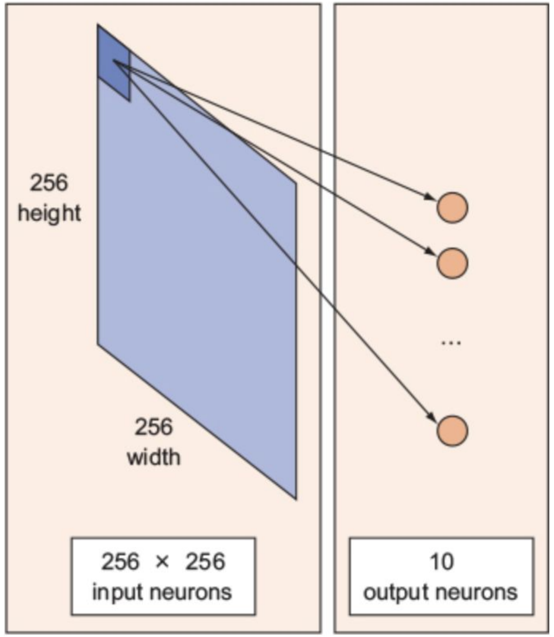
Convolutional neural networks

Overview

- Examining the components of a convolutional neural network
- Classifying natural images using deep learning
- Improving neural network performance—tips and tricks

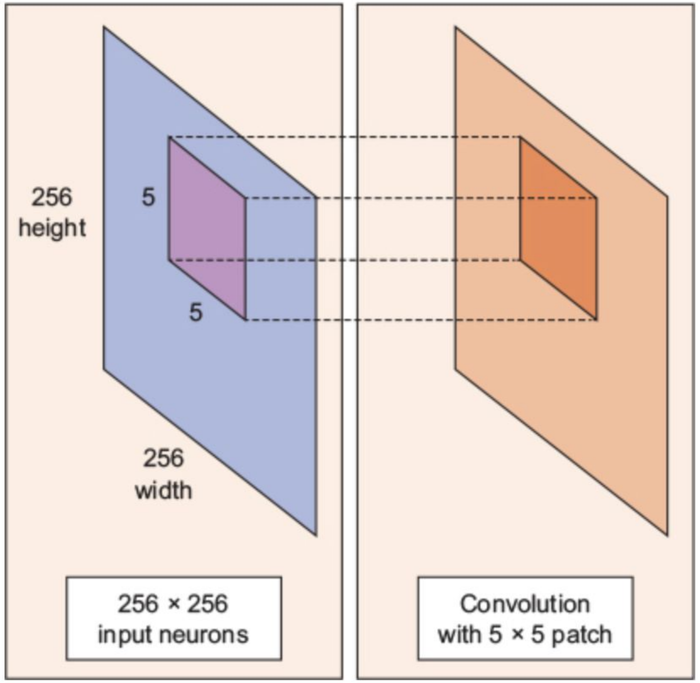
Drawback of neural networks

In a fully connected network, each pixel of an image is treated as an input. For a grayscale image of size 256 × 256, that’s 256 × 256 neurons! Connecting each neuron to 10 outputs yields 256 × 256 × 10 = 655,360 weights.



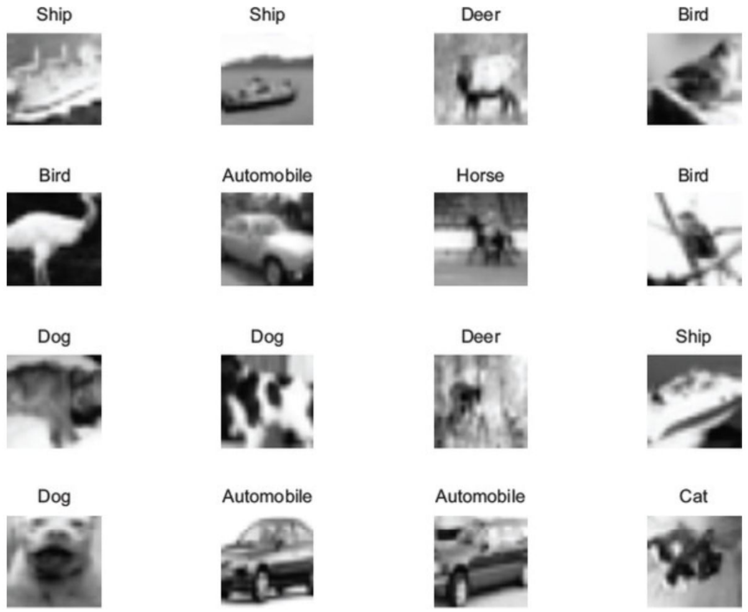
Convolutional neural networks

Convolving a 5 × 5 patch over an image, as shown on the left, produces another image, as shown on the right. In this case, the produced image is the same size as the original. Converting an original image to a convolved image requires only 5 × 5 = 25 parameters!



Preparing the image

Images from the CIFAR-10 dataset. Because they're only 32 × 32 in size, they're a bit difficult to see, but you can generally recognize some of the objects.



Preparing the image

Loading images from a CIFAR-10 file in Python

```
import pickle

def unpickle(file):
    fo = open(file, 'rb')
    dict = pickle.load(fo, encoding='latin1')
    fo.close()
    return dict
```

Preparing the image

Cleaning data

```
import numpy as np

def clean(data):
    imgs = data.reshape(data.shape[0], 3, 32, 32)
1
    grayscale_imgs = imgs.mean(1)                          2
    cropped_imgs = grayscale_imgs[:, 4:28, 4:28]
3
    img_data = cropped_imgs.reshape(data.shape[0], -1)
    img_size = np.shape(img_data)[1]
    means = np.mean(img_data, axis=1)
    meansT = means.reshape(len(means), 1)
    stds = np.std(img_data, axis=1)
    stdsT = stds.reshape(len(stds), 1)
    adj_stds = np.maximum(stdsT, 1.0 / np.sqrt(img_size))
    normalized = (img_data - meansT) / adj_stds
4
```

1 Reorganizes the data so it’s a 32 × 32 matrix with three channels
2 Grayscales the image by averaging the color intensities
3 Crops the 32 × 32 image to a 24 × 24 image
4 Normalizes the pixels’ values by subtracting the mean and dividing by standard deviation

Preparing the image

Preprocessing all CIFAR-10 files

```
def read_data(directory):
    names =
unpickle('{}/batches.meta'.format(directory))['label_names']
    print('names', names)

    data, labels = [], []
    for i in range(1, 6):
        filename = '{}/data_batch_{}'.format(directory, i)
        batch_data = unpickle(filename)
        if len(data) > 0:
            data = np.vstack((data, batch_data['data']))
            labels = np.hstack((labels, batch_data['labels']))
        else:
            data = batch_data['data']
            labels = batch_data['labels']

    print(np.shape(data), np.shape(labels))

    data = clean(data)
    data = data.astype(np.float32)
    return names, data, labels
```

Preparing the image

Using the cifar_tools helper function

```
import cifar_tools

names, data, labels = \

cifar_tools.read_data('your/location/to/cifar-10-batch
es-py')
```

Preparing the image

Visualizing images from the dataset

```
import numpy as np
import matplotlib.pyplot as plt
import random
def show_some_examples(names, data, labels):
    plt.figure()
    rows, cols = 4, 4                                          1
    random_idxs = random.sample(range(len(data)), rows * cols)    2
    for i in range(rows * cols):
        plt.subplot(rows, cols, i + 1)
        j = random_idxs[i]
        plt.title(names[labels[j]])
        img = np.reshape(data[j, :], (24, 24))
        plt.imshow(img, cmap='Greys_r')
        plt.axis('off')
    plt.tight_layout()
    plt.savefig('cifar_examples.png')

show_some_examples(names, data, labels)
```

1 Change this to as many rows and columns as you desire.
2 Randomly pick images from the dataset to show

Preparing the image

Generating and visualizing random filters

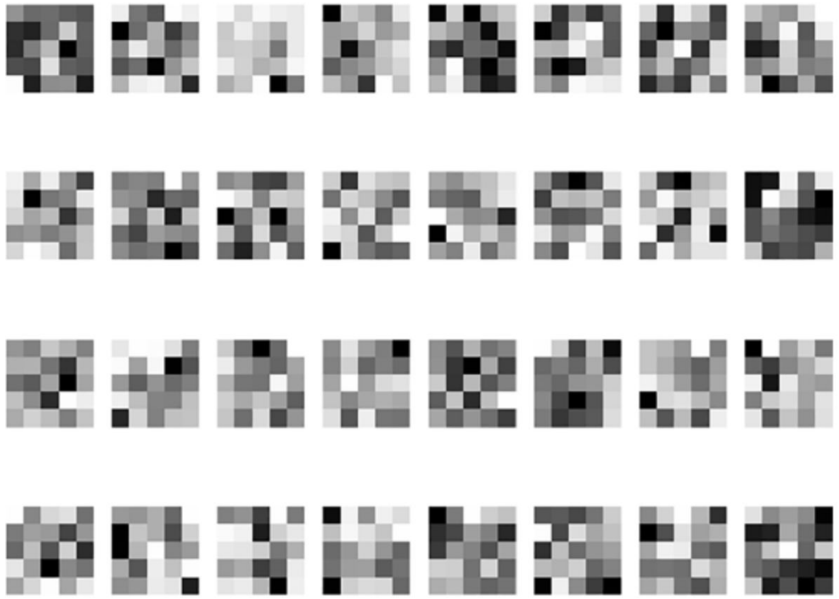
```
W = tf.Variable(tf.random_normal([5, 5, 1, 32]))            1

def show_weights(W, filename=None):
    plt.figure()
    rows, cols = 4, 8                                   2
    for i in range(np.shape(W)[3]):                         3
        img = W[:, :, 0, i]
        plt.subplot(rows, cols, i + 1)
        plt.imshow(img, cmap='Greys_r', interpolation='none')
        plt.axis('off')
    if filename:
        plt.savefig(filename)
    else:
        plt.show()
```

1 Defines the tensor representing the random filters
2 Defines just enough rows and columns to show the 32 figures in following figure
3 Visualizes each filter matrix

Preparing the image

These are 32 randomly initialized matrices, each of size 5 × 5. They represent the filters you’ll use to convolve an input image.



Preparing the image

Exercise

Change the above code to generate 64 filters of size 3 × 3.

Preparing the image

Exercise

Change the above code to generate 64 filters of size 3 × 3.

ANSWER

```
W = tf.Variable(tf.random_normal([3, 3, 1, 64]))
```

Preparing the image

Using a session to initialize weights

```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    W_val = sess.run(W)
    show_weights(W_val, 'step0_weights.png')
```

Preparing the image

Showing convolution results

```
def show_conv_results(data, filename=None):
    plt.figure()
    rows, cols = 4, 8
    for i in range(np.shape(data)[3]):
        img = data[0, :, :, i]                           1
        plt.subplot(rows, cols, i + 1)
        plt.imshow(img, cmap='Greys_r',
interpolation='none')
        plt.axis('off')
    if filename:
        plt.savefig(filename)
    else:
        plt.show()
```

1 Unlike in previous listing, this time the shape of the tensor is different.

Preparing the image

Visualizing convolutions

```
raw_data = data[4, :]                                    1
raw_img = np.reshape(raw_data, (24, 24))                 1
plt.figure()                                             1
plt.imshow(raw_img, cmap='Greys_r')                      1
plt.savefig('input_image.png')                           1

x = tf.reshape(raw_data, shape=[-1, 24, 24, 1])          2

b = tf.Variable(tf.random_normal([32]))                  3
conv = tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')
3
conv_with_b = tf.nn.bias_add(conv, b)                    3
conv_out = tf.nn.relu(conv_with_b)                       3
```

1 Gets an image from the CIFAR dataset, and visualizes it
2 Defines the input tensor for the 24 × 24 image
3 Defines the filters and corresponding parameters

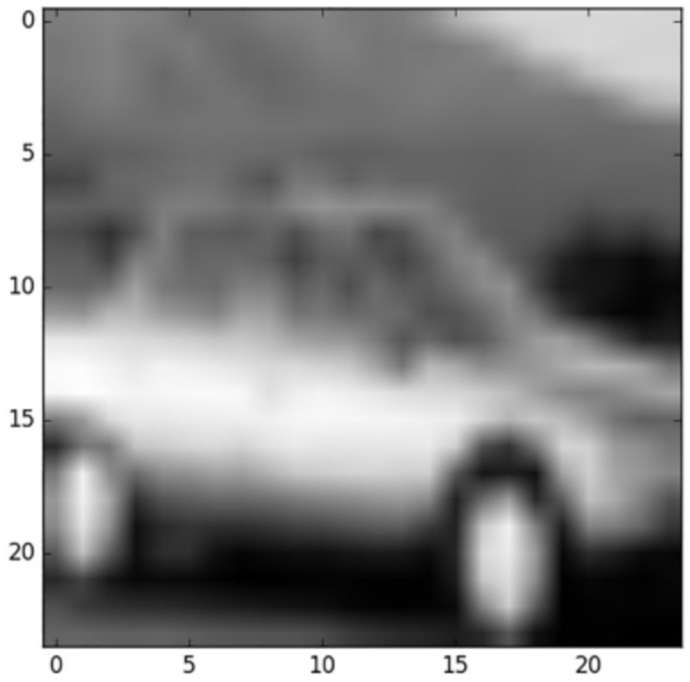
Preparing the image

```
with tf.Session() as sess:                                4
    sess.run(tf.global_variables_initializer())               4
                                                      4
    conv_val = sess.run(conv)                              4
    show_conv_results(conv_val, 'step1_convs.png')            4
    print(np.shape(conv_val))                              4
                                                      4
conv_out_val = sess.run(conv_out)                             4
    show_conv_results(conv_out_val, 'step2_conv_outs.png')      4
    print(np.shape(conv_out_val))                            4
```

4 Runs the convolution on the selected image

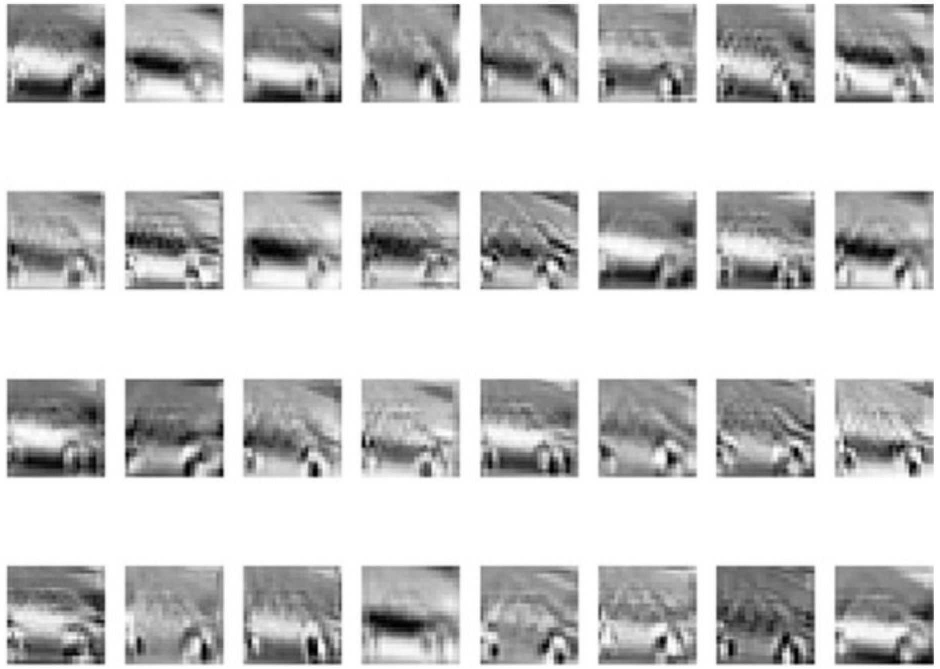
Preparing the image

An example 24 × 24 image from the CIFAR-10 dataset



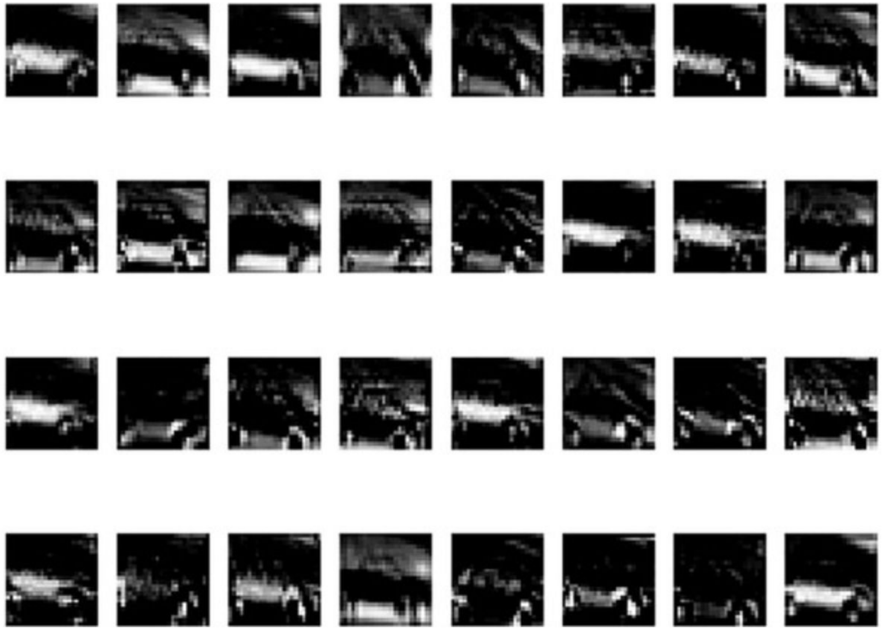
Preparing the image

Resulting images from convolving the random filters on an image of a car



Preparing the image

After you add a bias term and an activation function, the resulting convolutions can capture more-powerful patterns within images.



Preparing the image

Exercise

Let’s say you want to max pool over a 32 × 32 image. If the window size is 2 × 2 and the stride length is 2, how big will the resulting max-pooled image be?

Preparing the image

Exercise

Let’s say you want to max pool over a 32 × 32 image. If the window size is 2 × 2 and the stride length is 2, how big will the resulting max-pooled image be?

ANSWER

The 2 × 2 window would need to move 16 times in each direction to span the 32 × 32 image, so the image would shrink by half: 16 × 16. Because it shrank by half in both dimensions, the image is one-fourth the size of the original image (½ × ½).

Preparing the image

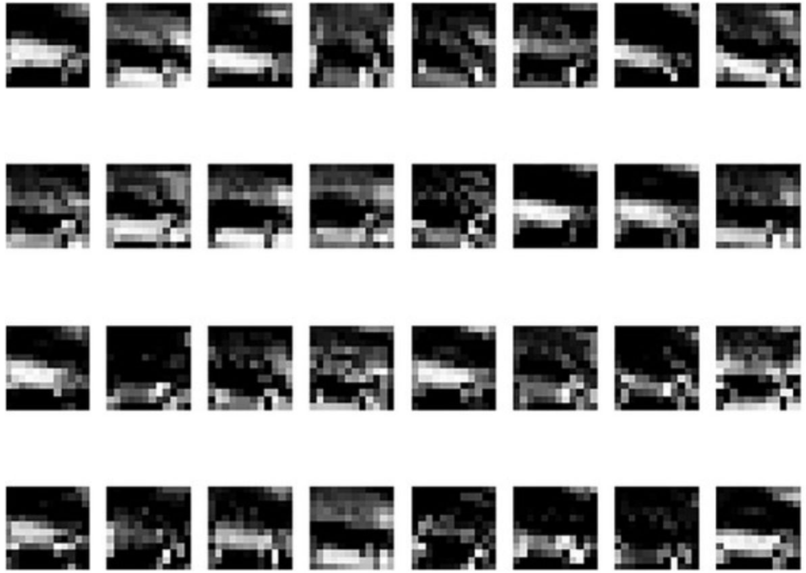
Running the maxpool function to subsample convolved images

```
 k = 2
 maxpool = tf.nn.max_pool(conv_out,
                          ksize=[1, k, k, 1],
                          strides=[1, k, k, 1],
                          padding='SAME')

with tf.Session() as sess:
   maxpool_val = sess.run(maxpool)
   show_conv_results(maxpool_val, 'step3_maxpool.png')
   print(np.shape(maxpool_val))
```

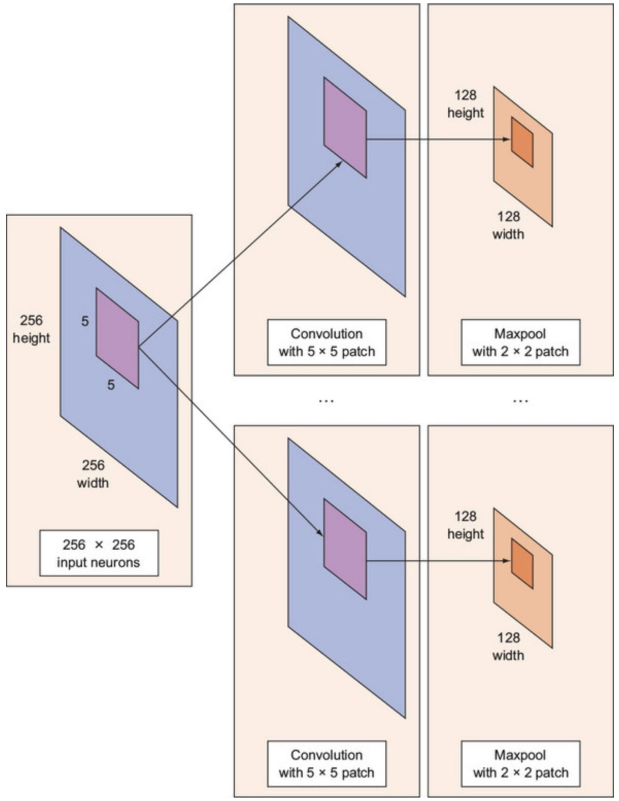
Preparing the image

After running maxpool, the convolved outputs are halved in size, making the algorithm computationally faster without losing too much information.



Implementing a convolutional neural network

An input image is convolved by multiple 5 × 5 filters. The convolution layer includes an added bias term with an activation function, resulting in 5 × 5 + 5 = 30 parameters. Next, a max-pooling layer reduces the dimensionality of the data (which requires no extra parameters).



Implementing a convolutional neural network

Setting up CNN weights

```
import numpy as np
import matplotlib.pyplot as plt
import cifar_tools
import tensorflow as tf
names, data, labels = \
    cifar_tools.read_data('/home/binroot/res/cifar-10-batches-py')     1

x = tf.placeholder(tf.float32, [None, 24 * 24])                    2
y = tf.placeholder(tf.float32, [None, len(names)])                   2

W1 = tf.Variable(tf.random_normal([5, 5, 1, 64]))                     3
b1 = tf.Variable(tf.random_normal([64]))                             3

W2 = tf.Variable(tf.random_normal([5, 5, 64, 64]))                    4
b2 = tf.Variable(tf.random_normal([64]))                             4

W3 = tf.Variable(tf.random_normal([6*6*64, 1024]))                     5
b3 = tf.Variable(tf.random_normal([1024]))                            5

W_out = tf.Variable(tf.random_normal([1024, len(names)]))              6
b_out = tf.Variable(tf.random_normal([len(names)]))                  6
```

1 Loads the dataset
2 Defines the input and output placeholders
3 Applies 64 convolutions of window size 5 × 5
4 Applies 64 more convolutions of window size 5 × 5
5 Introduces a fully connected layer
6 Defines the variables for a fully connected linear layer

Implementing a convolutional neural network

Creating a convolution layer

```
def conv_layer(x, W, b):
    conv = tf.nn.conv2d(x, W, strides=[1, 1, 1, 1],
padding='SAME')
    conv_with_b = tf.nn.bias_add(conv, b)
    conv_out = tf.nn.relu(conv_with_b)
    return conv_out
```

Implementing a convolutional neural network

Creating a max-pool layer

```
def maxpool_layer(conv, k=2):
    return tf.nn.max_pool(conv, ksize=[1, k, k, 1],
strides=[1, k, k, 1],
     padding='SAME')
```

Implementing a convolutional neural network

The full CNN model

```
def model():
    x_reshaped = tf.reshape(x, shape=[-1, 24, 24, 1])

    conv_out1 = conv_layer(x_reshaped, W1, b1)                              1
    maxpool_out1 = maxpool_layer(conv_out1)                                 1
    norm1 = tf.nn.lrn(maxpool_out1, 4, bias=1.0, alpha=0.001 / 9.0, beta=0.75)1

    conv_out2 = conv_layer(norm1, W2, b2)                                   2
    norm2 = tf.nn.lrn(conv_out2, 4, bias=1.0, alpha=0.001 / 9.0, beta=0.75)   2
    maxpool_out2 = maxpool_layer(norm2)                                     2

    maxpool_reshaped = tf.reshape(maxpool_out2, [-1,                          3
     W3.get_shape().as_list()[0]])                                        3
    local = tf.add(tf.matmul(maxpool_reshaped, W3), b3)                       3
    local_out = tf.nn.relu(local)                                         3
                                                                     3
    out = tf.add(tf.matmul(local_out, W_out), b_out)                        3
    return out                                                           3
```

1 Constructs the first layer of convolution and max pooling
2 Constructs the second layer
3 Constructs the concluding fully connected layers

Implementing a convolutional neural network

Defining ops to measure the cost and accuracy

```
model_op = model()

cost = tf.reduce_mean(                                            1
    tf.nn.softmax_cross_entropy_with_logits(logits=model_op,
labels=y)
)

train_op =
tf.train.AdamOptimizer(learning_rate=0.001).minimize(cost)    2

correct_pred = tf.equal(tf.argmax(model_op, 1), tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
```

1 Defines the classification loss function
2 Defines the training op to minimize the loss function

Implementing a convolutional neural network

Training the neural network by using the CIFAR-10 dataset

```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    onehot_labels = tf.one_hot(labels, len(names), on_value=1.,
off_value=0.,
     axis=-1)
    onehot_vals = sess.run(onehot_labels)
    batch_size = len(data) // 200
    print('batch size', batch_size)
    for j in range(0, 1000):                                    1
        print('EPOCH', j)
        for i in range(0, len(data), batch_size):               2
            batch_data = data[i:i+batch_size, :]
            batch_onehot_vals = onehot_vals[i:i+batch_size, :]
            _, accuracy_val = sess.run([train_op, accuracy], feed_dict={x:
     batch_data, y: batch_onehot_vals})
            if i % 1000 == 0:
                print(i, accuracy_val)
        print('DONE WITH EPOCH')
```

1 Loops through 1,000 epochs
2 Trains the network in batches

Tips and tricks to improve performance

- Augmenting data
- Early stopping
- Regularizing weights
- Dropout
- Deeper architecture

Tips and tricks to improve performance

Exercise

After the first iteration of this CNN architecture, try applying a couple of tips and tricks mentioned in this chapter.

ANSWER

Fine-tuning is, unfortunately, part of the process. You should begin by adjusting the hyperparameters and retraining the algorithm until you find a setting that works best.

Application of convolutional neural networks

- VGG Face Dataset: www.robots.ox.ac.uk/~vgg/data/vgg_face/
- FDDB: Face Detection Data Set and Benchmark: http://vis-www.cs.umass.edu/fddb/
- Databases for Face Detection and Pose Estimation: http://mng.bz/25N6
- YouTube Faces Database: www.cs.tau.ac.il/~wolf/ytfaces/