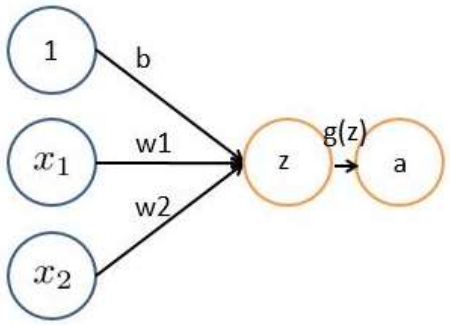
Experimental report on road detection in remote sensing images based on convolutional neural network

Ⅰ Significance and purpose of experiment

Learn the basic principles of convolutional neural networks, and be able to apply convolutional neural networks to remote sensing image classification problems.

Ⅱ Fundamentals of Convolutional Neural Networks

1. Fundamentals of Neural Networks



(1) Variable interpretation

Variable	Interpretation	Variable	Interpretation	Variable	Interpretation
x1,x2	Input	w1,w2	Weight	b	Bias
g(z)	Activation function	a	Output		

(2) Basic model

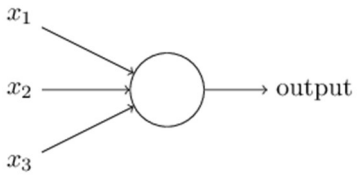


Figure 1

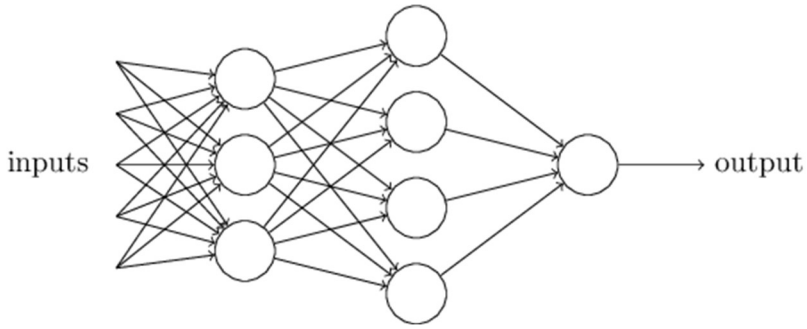


Figure 2

Figure (b) is a three-layer neural network structure composed of multiple single neurons shown in Figure (a)

(3) Explanation of neural network basic structure

The leftmost original input information in Figure (b) is called the input layer, the rightmost neuron is called the output layer (the output layer in the above figure has only one neuron), and the middle one is called the hidden layer.

In the input layer, many neurons receive a large amount of nonlinear input information, and the input information is called an input vector.

Output layer, the information is transmitted, analyzed, and weighed in the neuron connection to form the output result, and the output information is called the output vector.

Hidden layer, is a layer composed of many neurons and links between the input layer and the output layer. If there are multiple hidden layers, that means multiple activation functions.

(4) Training process

Step 1: Randomize initialization of parameters. Usually we need to initialize the parameters to a very small value close to 0. When doing logistic regression, we usually initialize the parameters to 0, but this is not feasible for neural networks. If they are all initialized to 0, then the value of each unit node in the second layer will be the same. If they are all initialized to the same non-zero numbers, the result is the same. Therefore, each parameter should be randomly selected within the range of positive and negative values around 0.

Step 2: Perform forward propagation. Calculate from left to right, for any input x(i), calculate h(x(i)), and the result is in the form of vector.

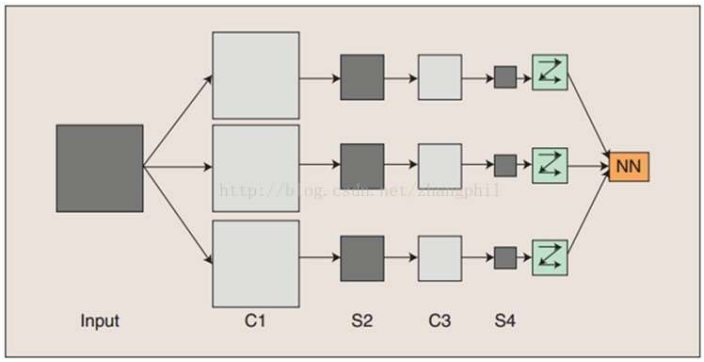
Step 3: Calculate loss, that is, the calculation of the cost function.

Step 4: Execute the backpropagation algorithm and calculate all the biases.

Step 5: Use the gradient descent method combined with the backpropagation algorithm to adjust the parameter value through the calculated biases to minimize the value of loss. However, it should be noted that since the neural network is not a linear model, its cost function is a non-concave

function, so the result obtained by using methods such as gradient descent is only a local optimal solution, and it is not guaranteed to be the global optimal solution.

2. **Fundamentals of Convolutional Neural Networks**



The image is convolved with three trainable filters and biases. The filtering process is shown in the figure. After convolution, three feature maps are generated in the C1 layer, and then four pixels in each group in the feature map are processed summing, weighting, and biasing, the feature maps of the three S2 layers are obtained through a Sigmoid function. These maps then enter the filter to obtain the C3 layer. This hierarchy then produces S4 as S2 does. Finally, these pixel values are rasterized and connected into a vector input to the traditional neural network to obtain an output. At this time, backpropagation is performed to adjust the parameters to minimize the loss value. We can see that the convolutional neural network follows the basic process and principle of the neural network.

Ⅲ Method steps

(1) Download the UCMerced_LandUse dataset and convert the .tif file to .jpg file by python.

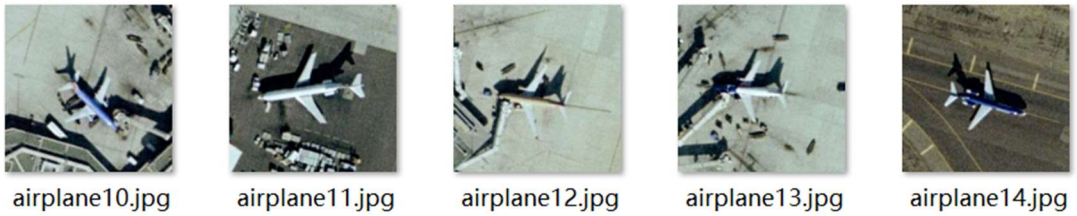
(2) Use python to generate the train.tfrecords file together with the .jpg file and self-made labels (resize the image size to 64*64 to speed up the training).

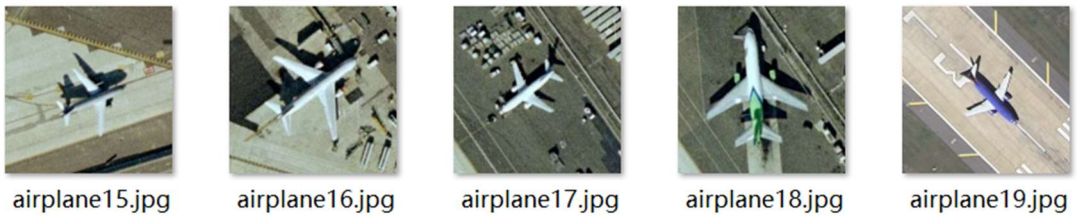
(3) Write a program to use python's tensorflow library to write CNN, and load the train.tfrecords file as a training sample.

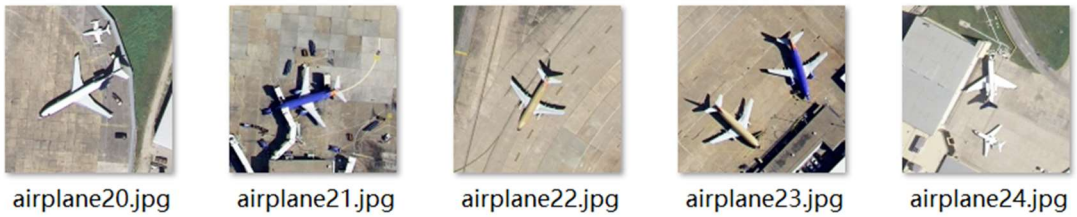
(4) Save the training network parameters in the .ckpt file so that don't have to train from beginning.

(5) Test the trained network again using the data set to judge the correct rate.

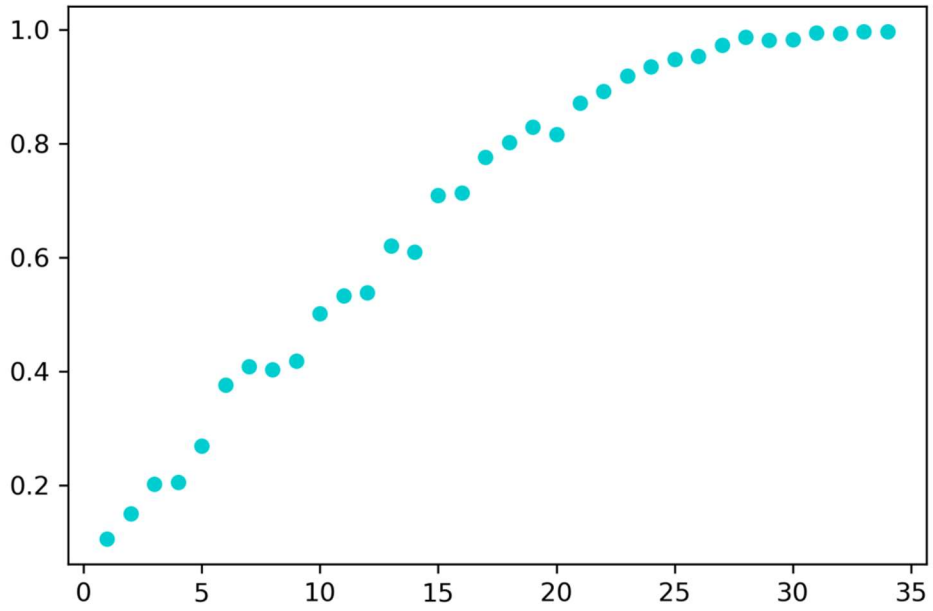
Ⅳ Sample dataset







V Experimental results



The convolutional neural network was trained 350 times using 200 sample images, and the accuracy rate was tested every 10 times using 1000 sample images. We can see from the figure that the accuracy rate gradually increases with the number of training times.

Save the training result as .ckpt file and use all 2100 sample images to test again, the accuracy rate is 0.9947619.

VI Experimental code

```
import tensorflow.compat.v1 as tf
import numpy as np
tf.disable_v2_behavior()

def read_tfRecord(tfRecord_path):

    filename_queue = tf.train.string_input_producer([tfRecord_path], shuffle=True)
    reader = tf.TFRecordReader()
    _, serialized_example = reader.read(filename_queue)
    features = tf.parse_single_example(serialized_example,
                                       features={
                                        'label': tf.FixedLenFeature([21], tf.int64),
                                        'img_raw': tf.FixedLenFeature([], tf.string)
                                        })
    label = features['label']
    img = features['img_raw']
    img = tf.decode_raw(img, tf.uint8)
    img = tf.reshape(img, [64, 64, 3])
```

```
    img = tf.cast(img, tf.float32) * (1. / 255)-0.5
    label = tf.cast(label, tf.int32)
    return img, label


def get_tfrecord(num, tfRecord_path):

    img, label = read_tfRecord(tfRecord_path)
    img_batch, label_batch = tf.train.shuffle_batch([img, label],
                                                    batch_size = num,
                                                    capacity = 2000,
                                                    min_after_dequeue = 0)
    return img_batch, label_batch

def compute_accuracy(v_xs,v_ys):
    global prediction
    y_pre = sess.run(prediction,feed_dict={xs:v_xs})
    correct_prediction = tf.equal(tf.argmax(y_pre,1),
                                  tf.argmax(v_ys,1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction,
                                      tf.float32))
    result = sess.run(accuracy,feed_dict={xs:v_xs,
                                          ys:v_ys})
    return result
def weight_variable(shape):
    inital = tf.truncated_normal(shape,stddev=0.1)
    return tf.Variable(inital,name='w')

def bias_variable(shape):
    inital = tf.constant(0.1,shape=shape)
    return tf.Variable(inital,name='b')

def conv2d(x,W):
    # Convolute the input x with W
    return tf.nn.conv2d(x,W,strides=[1,1,1,1],
                        padding='SAME')
def max_pool_2x2(x):
    # Perform pooling on the input x to halve the image size
    return tf.nn.max_pool(x,ksize=[1,2,2,1],
                          strides=[1,2,2,1],
                          padding='SAME')
xs = tf.placeholder(tf.float32,[None,64,64,3])
ys = tf.placeholder(tf.float32,[None,21])
x_image = tf.reshape(xs,[-1,64,64,3])

with tf.variable_scope("conv1"):
    W_conv1 = weight_variable([5,5,3,32])
    b_conv1 = bias_variable([32])
```

```
    h_conv1 = tf.nn.relu(conv2d(x_image,W_conv1) + b_conv1)
    h_pool1 = max_pool_2x2(h_conv1)

with tf.variable_scope("conv2"):
    W_conv2 = weight_variable([5,5,32,64])
    b_conv2 = bias_variable([64])
    h_conv2 = tf.nn.relu(conv2d(h_pool1,W_conv2)+b_conv2)
    h_pool2 = max_pool_2x2(h_conv2)

with tf.variable_scope("fc1"):
    W_fc1 = weight_variable([16*16*64,1024])
    b_fc1 = bias_variable([1024])
    h_pool2_flat = tf.reshape(h_pool2,[-1,16*16*64])
    h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat,W_fc1)+b_fc1)

with tf.variable_scope("fc2"):
    W_fc2 = weight_variable([1024,21])
    b_fc2 = bias_variable([21])
    prediction = tf.nn.softmax(tf.matmul(h_fc1,W_fc2)+b_fc2)
cross_entropy = tf.reduce_mean(-tf.reduce_sum(ys*tf.log(prediction),
                                              reduction_indices=[1]))
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)

saver = tf.train.Saver(max_to_keep=2)
sess = tf.Session()
sess.run(tf.initialize_all_variables())

path = 'F:/deepLearning/trainingData1/UCMerced_LandUse/train.tfrecords'

train_xs,train_ys = get_tfrecord(200,path)
test_xs,test_ys = get_tfrecord(1000,path)
threads=tf.train.start_queue_runners(sess=sess)

for i in range(350):
    train_images,train_labels = sess.run([train_xs,train_ys])

    sess.run(train_step,feed_dict={xs:train_images,ys:train_labels})

    test_images,test_labels = sess.run([test_xs,test_ys])

    saver.save(sess,
               "F:/deepLearning/trainingData1/UCMerced_LandUse/Model/model.ckpt",
               global_step=i)
    if i%10 == 0:
        print(compute_accuracy(
        test_images,test_labels))
print('Finish')
```