hw4.py


```
#!/usr/bin/env python3

import tensorflow as tf
import tensorflow.contrib.slim as slim
import numpy as np

# from tflearn.datasets import cifar10
from tflearn.datasets import cifar100

def residual(inputs, depth, kernel, scope='residual'):
    with tf.variable_scope(scope, 'residual', [inputs]) as sc:
        residual = inputs
        residual = slim.batch_norm(residual, activation_fn=tf.nn.relu, scope='bn1')
        residual = slim.conv2d(residual, depth, kernel, scope='conv1')
        residual = slim.batch_norm(residual, activation_fn=tf.nn.relu, scope='bn2')
        residual = slim.conv2d(residual, inputs.get_shape()[3], kernel, scope='conv2')
        residual = residual + inputs
    return residual

def resnet_small(inputs,
                 num_classes=None,
                 global_pool=True,
                 output_stride=None,
                 reuse=None,
                 scope='resnet_small'):
    with tf.variable_scope(scope, 'resnet_small', [inputs]) as sc:
        net = inputs
        net = slim.repeat(net, 1, slim.conv2d, 32, [5, 5], stride=1, scope='conv1')
        net = slim.max_pool2d(net, [2, 2], scope='pool1')
        net = slim.repeat(net, 1, slim.conv2d, 32, [5, 5], stride=1, scope='conv2')
        net = slim.max_pool2d(net, [2, 2], scope='pool2')
        net = residual(net, 32, [3, 3], scope='block3')
        net = residual(net, 32, [3, 3], scope='block4')
        net = residual(net, 64, [3, 3], scope='block5')
        net = residual(net, 64, [3, 3], scope='block6')
        net = slim.flatten(net, scope='flatten6')
        net = slim.fully_connected(net, 1024, scope='fc7')
        net = slim.dropout(net, 0.5, scope='dropout8')
        net = slim.fully_connected(net, num_classes, activation_fn=None, normalizer_fn=None, scope='logits')
    return net

class Model():
    def __init__(self, sess, n_batch, n_classes, learning_rate):
        self.sess = sess
        self.learning_rate = learning_rate
        self.n_batch = n_batch
        self.n_classes = n_classes
        self.build_model()

    def build_model(self):
        self.inputs = tf.placeholder(tf.float32, [None, 32, 32, 3])
```

```
        self.labels = tf.placeholder(tf.float32, [None, self.n_classes])
        with slim.arg_scope([slim.conv2d, slim.fully_connected],
                            activation_fn=tf.nn.relu,
                            weights_initializer=tf.truncated_normal_initializer(0.0, 0.01),
                            weights_regularizer=slim.l2_regularizer(0.002)):
            net = self.inputs
            net = resnet_small(net, num_classes=self.n_classes)
        self.predictions = net
        slim.losses.softmax_cross_entropy(self.predictions, self.labels)
        self.loss = slim.losses.get_total_loss(add_regularization_losses=True)
        self.optimizer = tf.train.MomentumOptimizer(learning_rate=self.learning_rate, momentum=0.9).minimize(self.los
s)

        correct = tf.equal(tf.argmax(self.predictions,1), tf.argmax(self.labels,1))
        self.accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))

    def train_minibatch(self, xs, ys):
        self.sess.run(self.optimizer, feed_dict={self.inputs: xs, self.labels: ys})

    def train(self, xs, ys, xvs, yvs, epochs):
        num_minibatches = xs.shape[0] // self.n_batch
        for epoch in range(epochs):
            p = np.random.permutation(xs.shape[0])
            for i in range(num_minibatches):
                # print("Minibatch " + str(i) + '/' + str(num_minibatches))
                start = i * self.n_batch
                end = (i + 1) * self.n_batch
                self.train_minibatch(xs[p][start:end], ys[p][start:end])
            accuracy = self.validate(xvs, yvs)
            print("Epoch {} validation accuracy: {}".format(epoch, accuracy))

    def validate(self, xs, ys):
        return self.sess.run(self.accuracy, feed_dict={self.inputs: xs, self.labels: ys})

    def test(self, xs):
        return self.sess.run(self.predictions, feed_dict={self.inputs: xs})

with tf.Session() as sess:
    # (train_images, train_labels), (test_images, test_labels) = cifar10.load_data(one_hot=True)
    # m = Model(sess, 128, 10, 0.01)
    (train_images, train_labels), (test_images, test_labels) = cifar100.load_data(one_hot=True)
    m = Model(sess, 128, 100, 0.01)
    sess.run(tf.global_variables_initializer())
    validation_images = train_images[0:5000]
    validation_labels = train_labels[0:5000]
    train_images = train_images[5000:]
    train_labels = train_labels[5000:]
    m.train(train_images, train_labels, validation_images, validation_labels, 100)
    test_accuracy = m.validate(test_images, test_labels)
    print("Test set accuracy: {}".format(test_accuracy))
```