```python
import numpy as np

from nndl.layers import *
from nndl.conv_layers import *
from cs231n.fast_layers import *
from nndl.layer_utils import *
from nndl.conv_layer_utils import *

import pdb

"""
This code was originally written for CS 231n at Stanford University
(cs231n.stanford.edu).  It has been modified in various areas for use in the
ECE 239AS class at UCLA.  This includes the descriptions of what code to
implement as well as some slight potential changes in variable names to be
consistent with class nomenclature.  We thank Justin Johnson & Serena Yeung for
permission to use this code.  To see the original version, please visit
cs231n.stanford.edu.
"""

class ThreeLayerConvNet(object):
  """
  A three-layer convolutional network with the following architecture:

  conv - relu - 2x2 max pool - affine - relu - affine - softmax

  The network operates on minibatches of data that have shape (N, C, H, W)
  consisting of N images, each with height H and width W and with C input
  channels.
  """

  def __init__(self, input_dim=(3, 32, 32), num_filters=32, filter_size=7,
               hidden_dim=100, num_classes=10, weight_scale=1e-3, reg=0.0,
               dtype=np.float32, use_batchnorm=False):
    """
    Initialize a new network.

    Inputs:
    - input_dim: Tuple (C, H, W) giving size of input data
    - num_filters: Number of filters to use in the convolutional layer
    - filter_size: Size of filters to use in the convolutional layer
    - hidden_dim: Number of units to use in the fully-connected hidden layer
    - num_classes: Number of scores to produce from the final affine layer.
    - weight_scale: Scalar giving standard deviation for random initialization
      of weights.
    - reg: Scalar giving L2 regularization strength
    - dtype: numpy datatype to use for computation.
    """
    self.use_batchnorm = use_batchnorm
    self.params = {}
    self.reg = reg
    self.dtype = dtype


    # ================================================================ #
    # YOUR CODE HERE:
    #   Initialize the weights and biases of a three layer CNN. To initialize:
    #     - the biases should be initialized to zeros.
    #     - the weights should be initialized to a matrix with entries
    #         drawn from a Gaussian distribution with zero mean and
    #         standard deviation given by weight_scale.
    # ================================================================ #
    C, H, W = input_dim
    F = num_filters
    filterHeight = filter_size
    filterWidth = filter_size
    stride = 1
    P = (filter_size - 1) / 2
    Hc = ((H + 2 * P - filterHeight) / stride )+ 1
```

```
    Wc = ((W + 2 * P - filterWidth) / stride) + 1

    W1 = weight_scale * np.random.randn(F, C, filterHeight, filterWidth)
    b1 = np.zeros((F))


    width_pool = 2
    height_pool = 2
    # stride_pool = 2
    # Hp = ((Hc - height_pool) / stride_pool) + 1
    # Wp = ((Wc - width_pool) / stride_pool ) + 1



    Hh = hidden_dim
    W2 = weight_scale * np.random.randn((F * Hc * Wc)/(width_pool* height_pool), Hh)
    b2 = np.zeros((Hh))



    Hc = num_classes
    W3 = weight_scale * np.random.randn(Hh, Hc)
    b3 = np.zeros((Hc))

    self.params['W1'], self.params['b1'] = W1, b1
    self.params['W2'], self.params['b2'] = W2, b2
    self.params['W3'], self.params['b3'] = W3, b3
    # ============================================================ #
    # END YOUR CODE HERE
    # ============================================================ #

    for k, v in self.params.items():
      self.params[k] = v.astype(dtype)


  def loss(self, X, y=None):
    """
    Evaluate loss and gradient for the three-layer convolutional network.

    Input / output: Same API as TwoLayerNet in fc_net.py.
    """
    W1, b1 = self.params['W1'], self.params['b1']
    W2, b2 = self.params['W2'], self.params['b2']
    W3, b3 = self.params['W3'], self.params['b3']

    # pass conv_param to the forward pass for the convolutional layer
    filter_size = W1.shape[2]
    conv_param = {'stride': 1, 'pad': (filter_size - 1) / 2}

    # pass pool_param to the forward pass for the max-pooling layer
    pool_param = {'pool_height': 2, 'pool_width': 2, 'stride': 2}

    scores = None

    # ============================================================ #
    # YOUR CODE HERE:
    #   Implement the forward pass of the three layer CNN.  Store the output
    #   scores as the variable "scores".
    # ============================================================ #
    out1, cache1 = conv_relu_pool_forward(X, W1, b1, conv_param, pool_param)
    out2, cache2 = affine_relu_forward(out1, W2, b2)
    scores, cache3 = affine_forward(out2, W3, b3)


    # N, F, Hp, Wp = out1.shape
    # out2 = out1.reshape((N, F * Hp * Wp))


    # ============================================================ #
    # END YOUR CODE HERE
```

```
140     # ================================================================ #
141
142     if y is None:
143       return scores
144
145     loss, grads = 0, {}
146     # ================================================================ #
147     # YOUR CODE HERE:
148     #   Implement the backward pass of the three layer CNN.  Store the grads
149     #   in the grads dictionary, exactly as before (i.e., the gradient of
150     #   self.params[k] will be grads[k]).  Store the loss as "loss", and
151     #   don't forget to add regularization on ALL weight matrices.
152     # ================================================================ #
153     loss, dscores = softmax_loss(scores, y)
154     reg_loss = 0.5 * self.reg * np.sum(W1**2)
155     reg_loss += 0.5 * self.reg * np.sum(W2**2)
156     reg_loss += 0.5 * self.reg * np.sum(W3**2)
157     loss = loss + reg_loss
158
159     dx3, grads['W3'], grads['b3'] = affine_backward(dscores, cache3)
160     dx2, grads['W2'], grads['b2'] = affine_relu_backward(dx3, cache2)
161     dx1, grads['W1'], grads['b1'] = conv_relu_pool_backward(dx2, cache1)
162
163     grads['W1'] += self.reg * self.params['W1']
164     grads['W2'] += self.reg * self.params['W2']
165     grads['W3'] += self.reg * self.params['W3']
166     # ================================================================ #
167     # END YOUR CODE HERE
168     # ================================================================ #
169
170     return loss, grads
171
172
173 pass
```