layer_utils.py

```
from .layers import *

"""
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.
"""

def affine_relu_forward(x, w, b):
  """
  Convenience layer that performs an affine transform followed by a ReLU

  Inputs:
  - x: Input to the affine layer
  - w, b: Weights for the affine layer

  Returns a tuple of:
  - out: Output from the ReLU
  - cache: Object to give to the backward pass
  """
  a, fc_cache = affine_forward(x, w, b)
  out, relu_cache = relu_forward(a)
  cache = (fc_cache, relu_cache)
  return out, cache


def affine_relu_backward(dout, cache):
  """
  Backward pass for the affine-relu convenience layer
  """
  fc_cache, relu_cache = cache
  da = relu_backward(dout, relu_cache)
  dx, dw, db = affine_backward(da, fc_cache)
  return dx, dw, db

def affine_batchnorm_relu_forward(x, w, b, gamma, beta, bn_param):
  """
  Convenience layer that performs an affine transform followed by batch normalization and then ReLU

  Inputs:
  - x: Input to the affine layer
  - w, b: Weights for the affine layer
  - gamma, beta: the gamma and beta parameters associated with this layer.
  - bn_param: a set of parameters corresponding to the layer. Relevant mainly for testing

  Returns a tuple of:
  - out: Output from the ReLU
  - cache: Object to give to the backward pass
  """
  a, fc_cache = affine_forward(x, w, b)
  a, bn_cache = batchnorm_forward(a, gamma, beta, bn_param)
  out, relu_cache = relu_forward(a)
  cache = (fc_cache, bn_cache, relu_cache)
  return out, cache

def affine_batchnorm_relu_backward(dout, cache):
  """
  Backward pass for the affine-batchnorm-relu convenience layer
  """
  fc_cache, bn_cache, relu_cache = cache
  da = relu_backward(dout, relu_cache)
  da, dgamma, dbeta = batchnorm_backward(da, bn_cache)
  dx, dw, db = affine_backward(da, fc_cache)
  return dx, dw, db, dgamma, dbeta
```