```python
from .layers import *

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 a batchnorm and ReLU

    Inputs:
    - x: Input to the affine layer
    - w, b: Weights for the affine layer
    - gamma: Scale parameter of shape (D,)
    - beta: Shift paremeter of shape (D,)
    - bn_param: Dictionary with the following keys:
      - mode: 'train' or 'test'; required
      - eps: Constant for numeric stability
      - momentum: Constant for running mean / variance.
      - running_mean: Array of shape (D,) giving running mean of features
      - running_var Array of shape (D,) giving running variance of features

    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_norm, bn_cache = batchnorm_forward(a, gamma, beta, bn_param)
    out, relu_cache = relu_forward(a_norm)
    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_norm = relu_backward(dout, relu_cache)
    da, dgamma, dbeta = batchnorm_backward(da_norm, bn_cache)
    dx, dw, db = affine_backward(da, fc_cache)
    return dx, dw, db, dgamma, dbeta
```