```python
import numpy as np


def random_flips(X):
  """
  Take random x-y flips of images.

  Input:
  - X: (N, C, H, W) array of image data.

  Output:
  - An array of the same shape as X, containing a copy of the data in X,
    but with half the examples flipped along the horizontal direction.
  """
  out = None
  #############################################################################
  # TODO: Implement the random_flips function. Store the result in out.       #
  #############################################################################
  N, C, H, W = X.shape

  flips = np.random.randint(2, size=N)
  out = np.zeros(X.shape)
  out[flips == 1] = X[flips == 1, :, :, ::-1]
  out[flips == 0] = X[flips == 0]
  #############################################################################
  #                            END OF YOUR CODE                               #
  #############################################################################
  return out


def random_crops(X, crop_shape):
  """
  Take random crops of images. For each input image we will generate a random
  crop of that image of the specified size.

  Input:
  - X: (N, C, H, W) array of image data
  - crop_shape: Tuple (HH, WW) to which each image will be cropped.

  Output:
  - Array of shape (N, C, HH, WW)
  """
  N, C, H, W = X.shape
  HH, WW = crop_shape
  assert HH < H and WW < W

  out = np.zeros((N, C, HH, WW), dtype=X.dtype)
  #############################################################################
  # TODO: Implement the random_crops function. Store the result in out.       #
  #############################################################################
  np.random.randint((H-HH), size=N)
  y_crop = np.random.randint((H-HH), size=N)
  x_crop = np.random.randint((W-WW), size=N)

  for i in xrange(N):
        out[i] = X[i, :, y_crop[i]:y_crop[i]+HH, x_crop[i]:x_crop[i]+WW]
  #############################################################################
  #                            END OF YOUR CODE                               #
  #############################################################################

  return out


def random_contrast(X, scale=(0.8, 1.2)):
  """
  Randomly adjust the contrast of images. For each input image, choose a
  number uniformly at random from the range given by the scale parameter,
  and multiply each pixel of the image by that number.

  Inputs:
  - X: (N, C, H, W) array of image data
  - scale: Tuple (low, high). For each image we sample a scalar in the
    range (low, high) and multiply the image by that scaler.

  Output:
  - Rescaled array out of shape (N, C, H, W) where out[i] is a contrast
    adjusted version of X[i].
  """
  low, high = scale
  N = X.shape[0]
  out = np.zeros_like(X)

  #############################################################################
  # TODO: Implement the random_contrast function. Store the result in out.    #
  #############################################################################
  contrast = (scale[1]-scale[0])*np.random.random_sample(N)+scale[0]
  out = X * contrast[:, None, None, None]
  #############################################################################
  #                             END OF YOUR CODE                              #
  #############################################################################

  return out


def random_tint(X, scale=(-10, 10)):
  """
  Randomly tint images. For each input image, choose a random color whose
  red, green, and blue components are each drawn uniformly at random from
  the range given by scale. Add that color to each pixel of the image.

  Inputs:
  - X: (N, C, W, H) array of image data
  - scale: A tuple (low, high) giving the bounds for the random color that
    will be generated for each image.

  Output:
  - Tinted array out of shape (N, C, H, W) where out[i] is a tinted version
    of X[i].
  """
  low, high = scale
  N, C = X.shape[:2]
  out = np.zeros_like(X)

  #############################################################################
  # TODO: Implement the random_tint function. Store the result in out.        #
  #############################################################################
  tint = (scale[1]-scale[0])*np.random.random_sample((N, C))+scale[0]
  out = X+tint[:, :, None, None]
  #############################################################################
  #                             END OF YOUR CODE                              #
  #############################################################################

  return out


def fixed_crops(X, crop_shape, crop_type):
  """
  Take center or corner crops of images.

  Inputs:
  - X: Input data, of shape (N, C, H, W)
  - crop_shape: Tuple of integers (HH, WW) giving the size to which each
    image will be cropped.
  - crop_type: One of the following strings, giving the type of crop to
```

```
    compute:
    'center': Center crop
    'ul': Upper left corner
    'ur': Upper right corner
    'bl': Bottom left corner
    'br': Bottom right corner

  Returns:
  Array of cropped data of shape (N, C, HH, WW)
  """
  N, C, H, W = X.shape
  HH, WW = crop_shape

  x0 = (W - WW) / 2
  y0 = (H - HH) / 2
  x1 = x0 + WW
  y1 = y0 + HH

  if crop_type == 'center':
    return X[:, :, y0:y1, x0:x1]
  elif crop_type == 'ul':
    return X[:, :, :HH, :WW]
  elif crop_type == 'ur':
    return X[:, :, :HH, -WW:]
  elif crop_type == 'bl':
    return X[:, :, -HH:, :WW]
  elif crop_type == 'br':
    return X[:, :, -HH:, -WW:]
  else:
    raise ValueError('Unrecognized crop type %s' % crop_type)
```