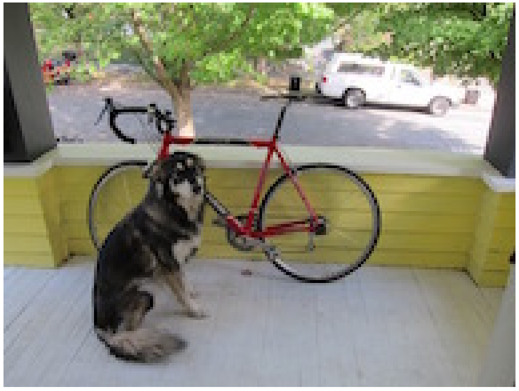
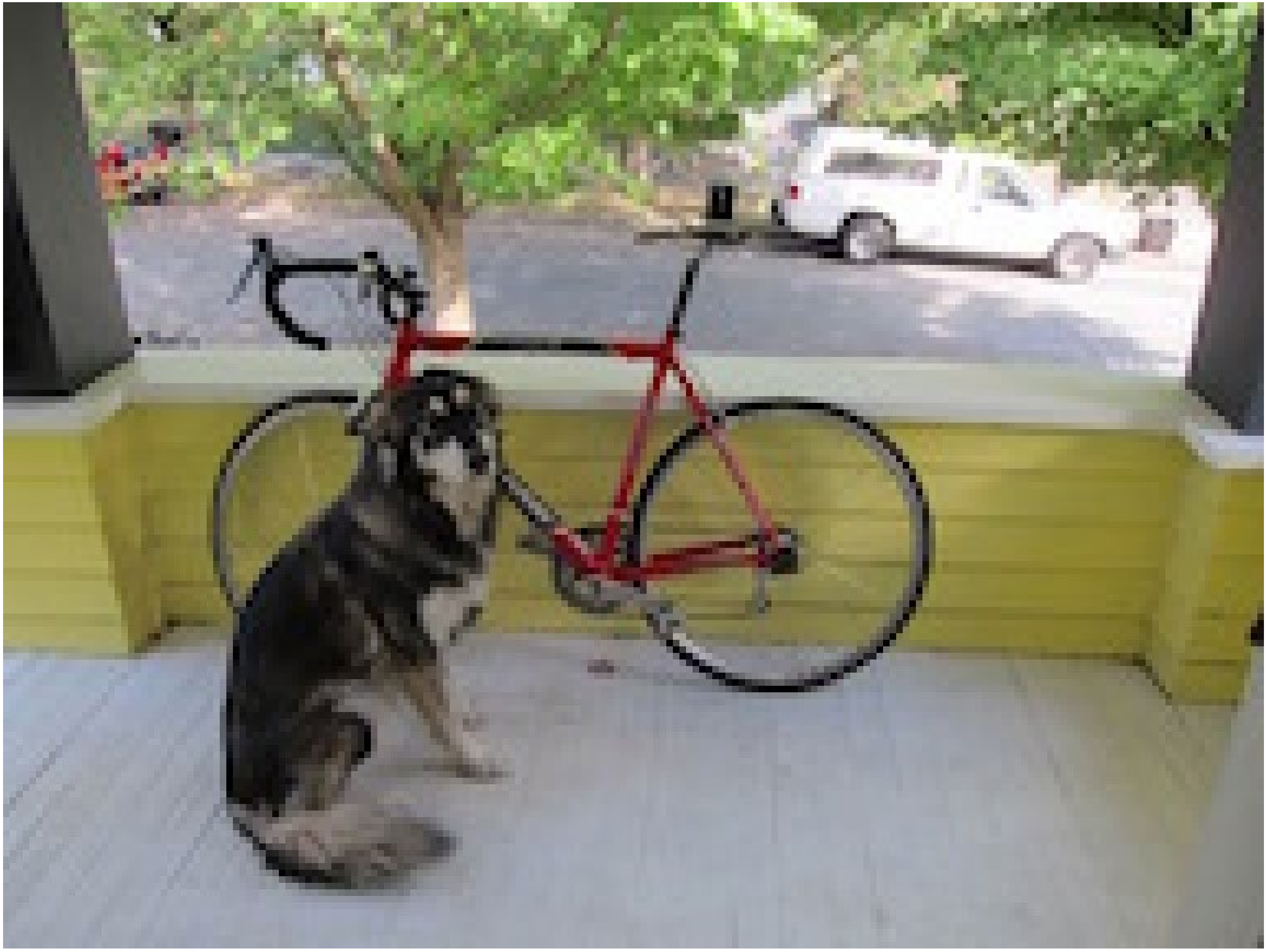
Original Image:



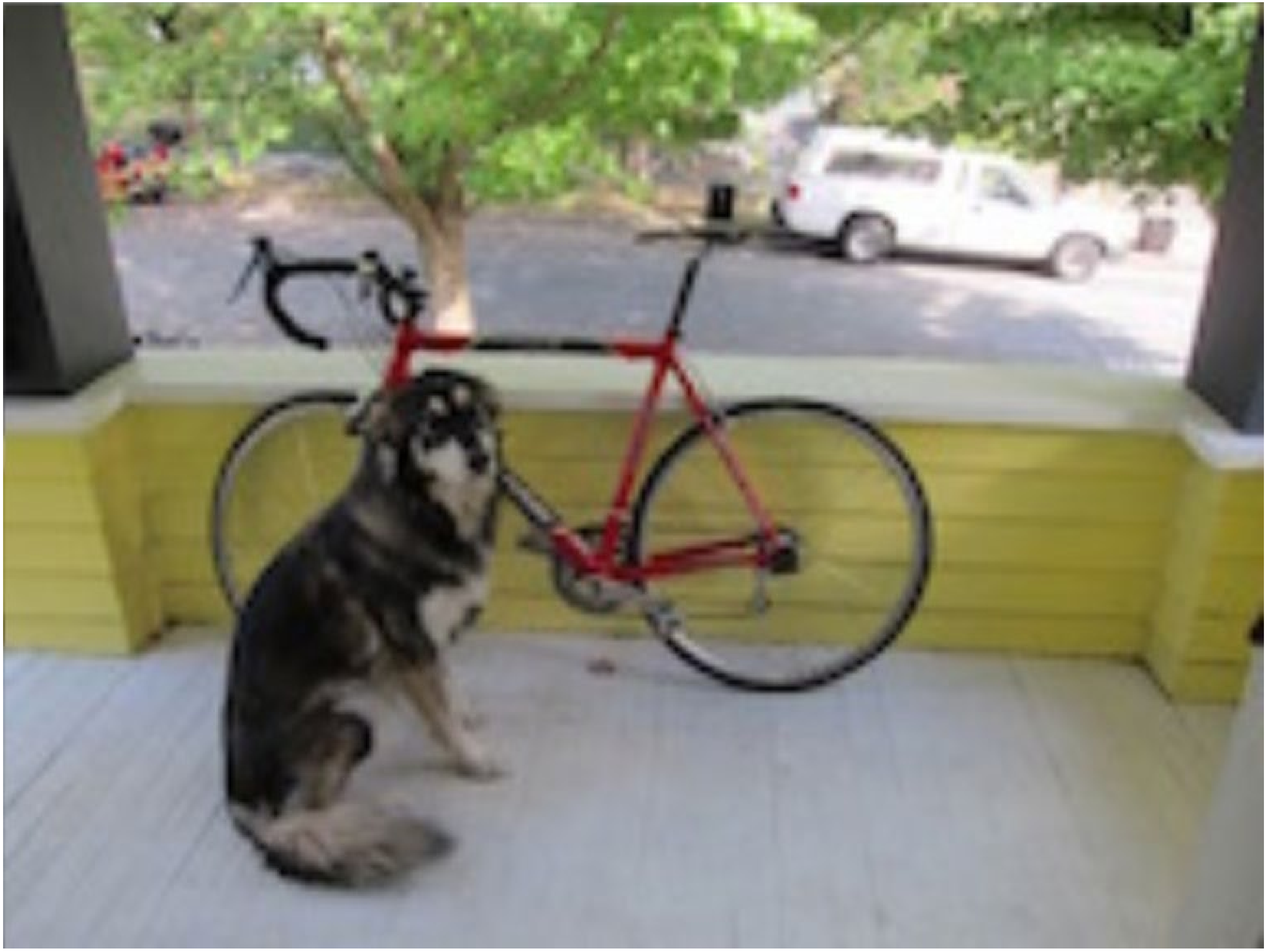
Nearest Neighbor Interpolation Result:



Nearest Neighbor Code:

```
def nn_resize(im, dim):
    # Create new image with size dim and im number of channels
    newimage = np.ones([dim[1], dim[0], im.shape[2]], dtype = "uint8")
    ratiox = dim[1] / im.shape[0]
    ratioy = dim[0] / im.shape[1]
    offsetx = (im.shape[0] - dim[1]) / (2 * dim[1] + 1)
    offsety = (im.shape[1] - dim[0]) / (2 * dim[0] + 1)
    # Loop over pixels and map back to old coordinates
    for x in range(0, dim[1]):
        for y in range(0, dim[0]):
            # Use nearest-neighbor interpolate to fill in the image
            srcx = min(int(round(x / ratiox + offsetx)), im.shape[0] - 1)
            srcy = min(int(round(y / ratioy + offsety)), im.shape[1] - 1)
            newimage[x][y] = im[srcx][srcy]
    return newimage
```

Bilinear Interpolation Result:



Bilinear Code:

```
def bilinear_resize(im, dim):
    # Create new image with size dim and im number of channels
    newimage = np.ones([dim[1], dim[0], im.shape[2]], dtype = "uint8")
    ratiox = dim[1] / im.shape[0]
    ratioy = dim[0] / im.shape[1]
    width = im.shape[0] - 1
    height = im.shape[1] - 1
    offsetx = (im.shape[0] - dim[1]) / (2 * dim[1] + 1)
    offsety = (im.shape[1] - dim[0]) / (2 * dim[0] + 1)
    print(offsetx, offsety)
    # Loop over pixels and map back to old coordinates
    for x in range(0, dim[1]):
        for y in range(0, dim[0]):
            for c in range(0, im.shape[2]):
                # Use bilinear interpolate to fill in the image
                d1 = x/ratiox + offsetx - math.floor(x/ratiox + offsetx)
                d2 = 1 - d1
                d3 = y/ratioy + offsety - math.floor(y/ratioy + offsety)
                d4 = 1 - d3
                v1 = im[math.floor(x/ratiox + offsetx)][math.floor(y/ratioy + 
offsety)][c]
```

```
                v2 = im[min(math.floor(x/ratiox + offsetx) + 1, 
width)][math.floor(y/ratioy + offsety)][c]
                v3 = im[math.floor(x/ratiox + offsetx)][min(math.floor(y/ratioy + 
offsety) + 1, height)][c]
                v4 = im[min(math.floor(x/ratiox + offsetx) + 1, 
width)][min(math.floor(y/ratioy + offsety) + 1, height)][c]
                a4 = d1 * d3 / ((d1 + d2) * (d3 + d4))
                a3 = d2 * d3 / ((d1 + d2) * (d3 + d4))
                a2 = d1 * d4 / ((d1 + d2) * (d3 + d4))
                a1 = d2 * d4 / ((d1 + d2) * (d3 + d4))
                q = v1 * a1 + v2 * a2 + v3 * a3 + v4 * a4
                newimage[x][y][c] = q
    return newimage
```