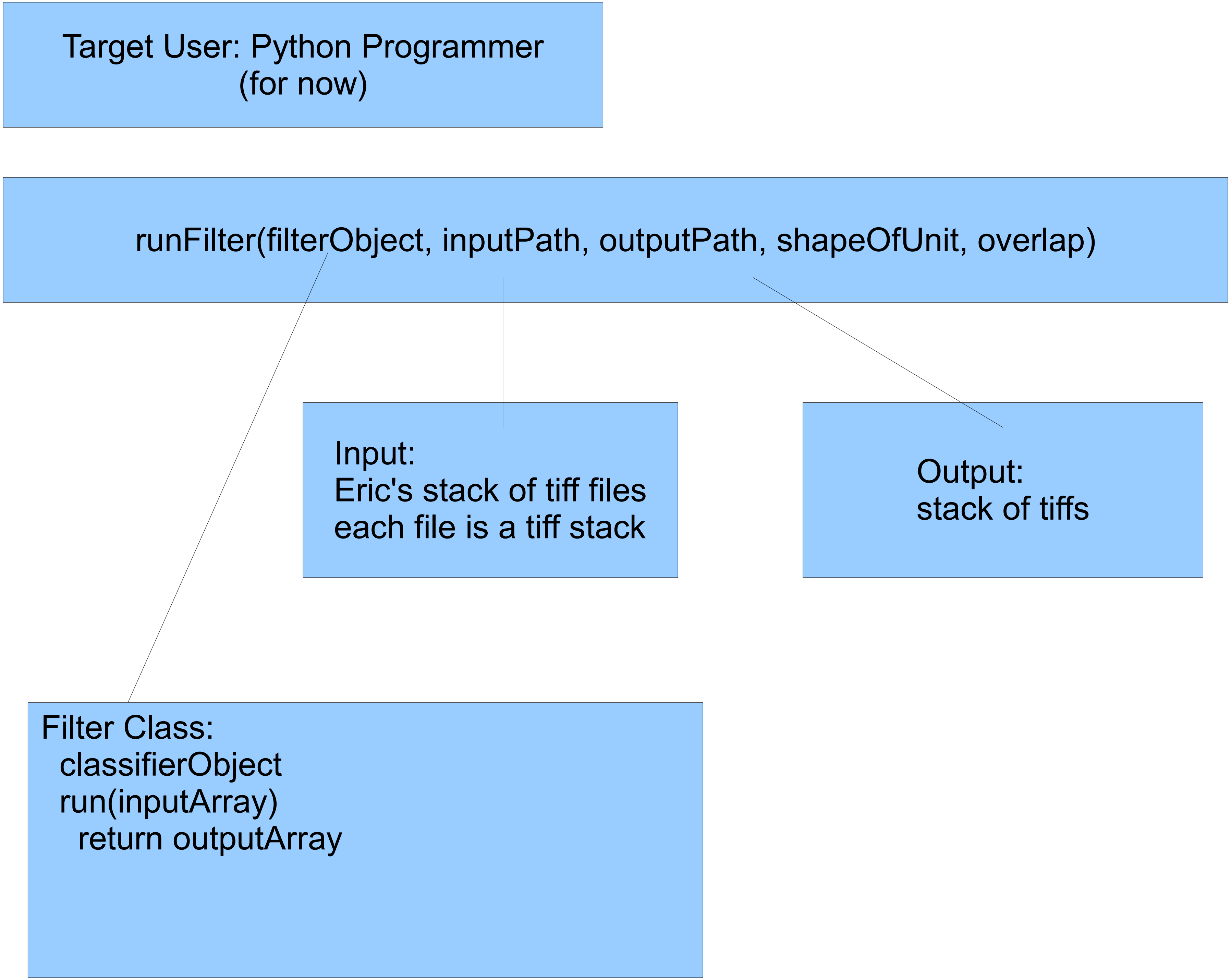


Target User: Python Programmer
(for now)

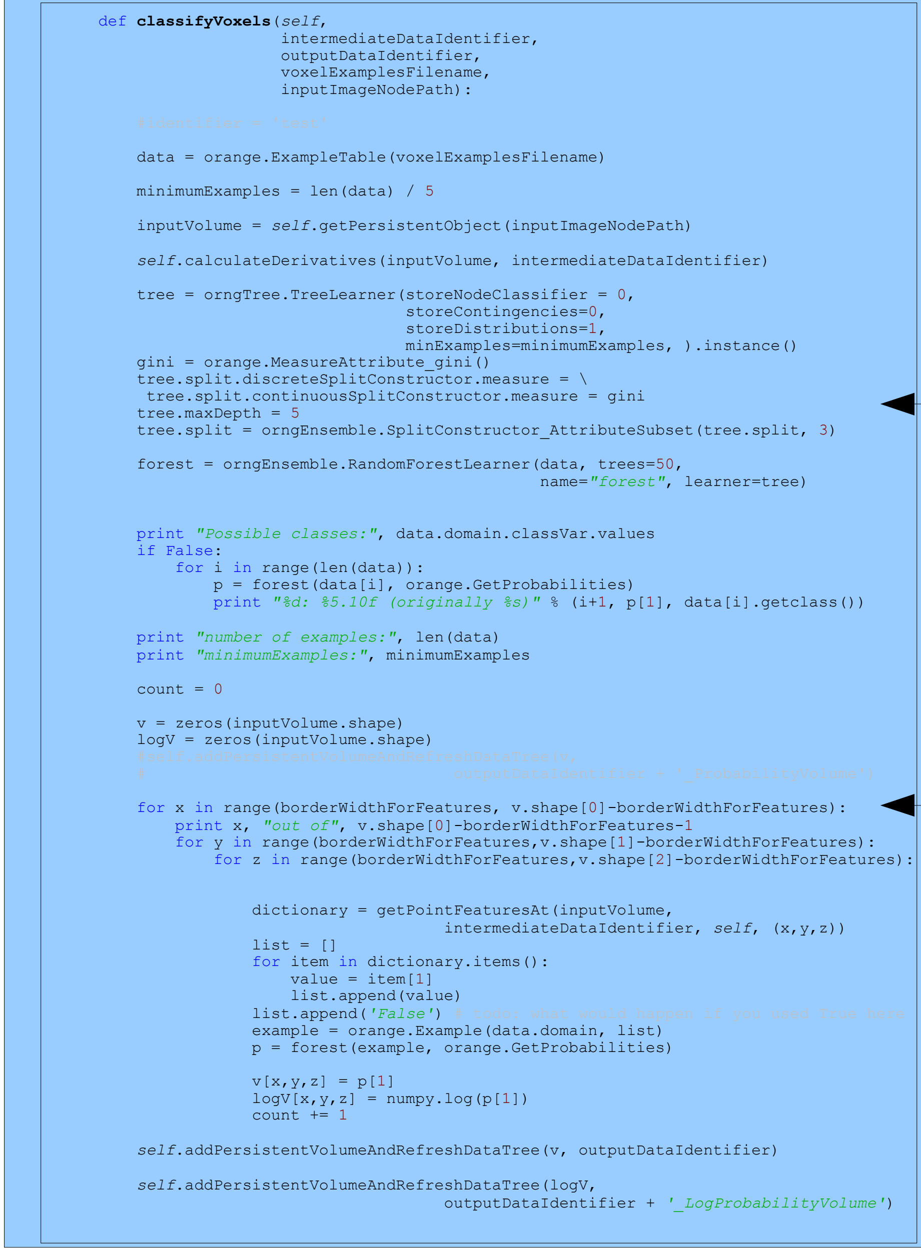
runFilter(filterObject, inputPath, outputPath, shapeOfUnit, overlap)

Input:
Eric's stack of tiff files
each file is a tiff stack

Output:
stack of tiffs

```
Filter Class:
  classifierObject
  run(inputArray)
    return outputArray
```

Current Code:



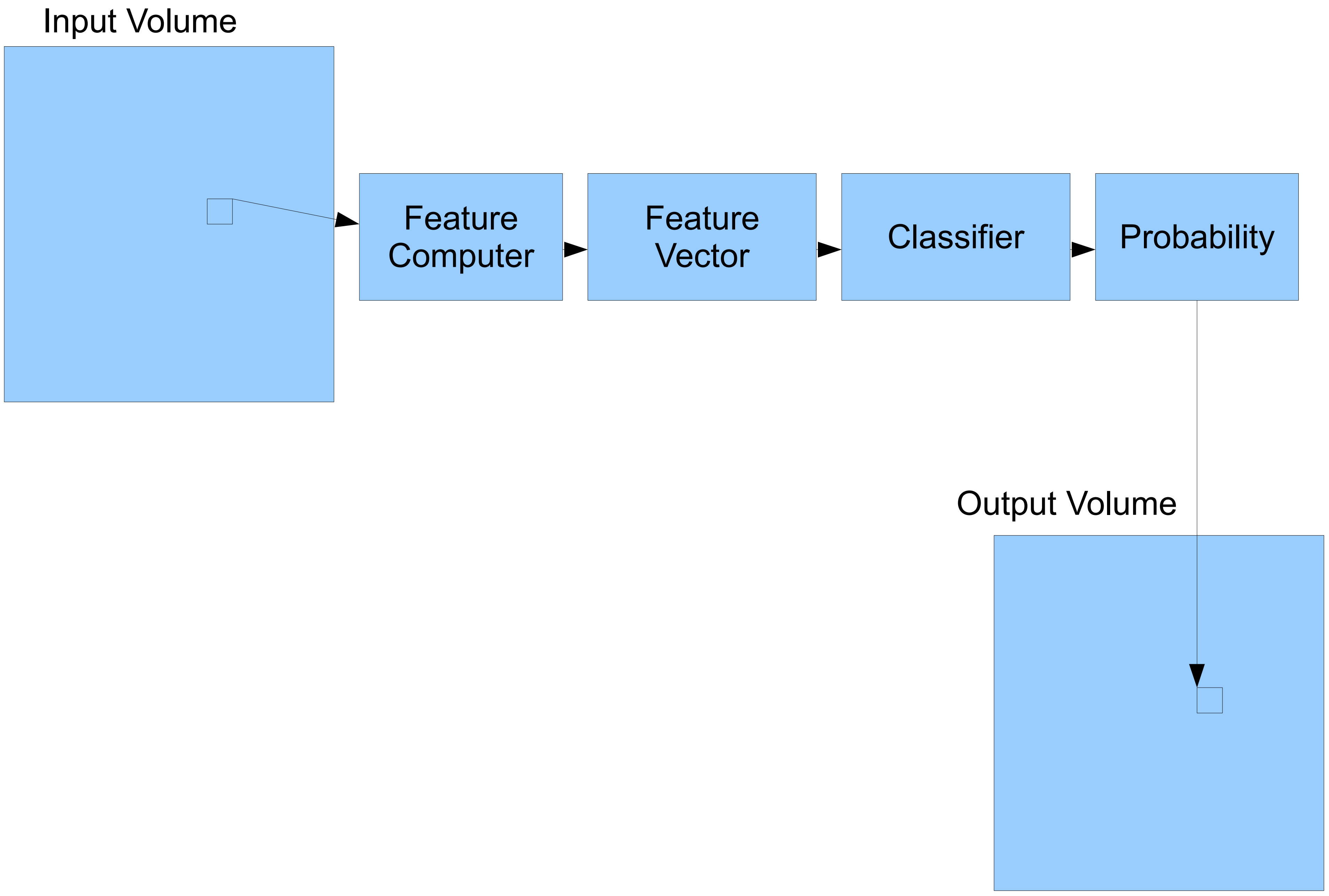
```python
    def classifyVoxels(self,
                       intermediateDataIdentifier,
                       outputDataIdentifier,
                       voxelExamplesFilename,
                       inputImageNodePath):

        #identifier = 'test'

        data = orange.ExampleTable(voxelExamplesFilename)

        minimumExamples = len(data) / 5

        inputVolume = self.getPersistentObject(inputImageNodePath)

        self.calculateDerivatives(inputVolume, intermediateDataIdentifier)

        tree = orngTree.TreeLearner(storeNodeClassifier = 0,
                                    storeContingencies=0,
                                    storeDistributions=1,
                                    minExamples=minimumExamples, ).instance()
        gini = orange.MeasureAttribute_gini()
        tree.split.discreteSplitConstructor.measure = \
         tree.split.continuousSplitConstructor.measure = gini
        tree.maxDepth = 5
        tree.split = orngEnsemble.SplitConstructor_AttributeSubset(tree.split, 3)

        forest = orngEnsemble.RandomForestLearner(data, trees=50,
                                                  name="forest", learner=tree)


        print "Possible classes:", data.domain.classVar.values
        if False:
            for i in range(len(data)):
                p = forest(data[i], orange.GetProbabilities)
                print "%d: %5.10f (originally %s)" % (i+1, p[1], data[i].getclass())

        print "number of examples:", len(data)
        print "minimumExamples:", minimumExamples

        count = 0

        v = zeros(inputVolume.shape)
        logV = zeros(inputVolume.shape)
        #self.addPersistentVolumeAndRefreshDataTree(v,
        #                                outputDataIdentifier + '_ProbabilityVolume')

        for x in range(borderWidthForFeatures, v.shape[0]-borderWidthForFeatures):
            print x, "out of", v.shape[0]-borderWidthForFeatures-1
            for y in range(borderWidthForFeatures,v.shape[1]-borderWidthForFeatures):
                for z in range(borderWidthForFeatures,v.shape[2]-borderWidthForFeatures):


                    dictionary = getPointFeaturesAt(inputVolume,
                                        intermediateDataIdentifier, self, (x,y,z))
                    list = []
                    for item in dictionary.items():
                        value = item[1]
                        list.append(value)
                    list.append('False') # todo: what would happen if you used True here
                    example = orange.Example(data.domain, list)
                    p = forest(example, orange.GetProbabilities)

                    v[x,y,z] = p[1]
                    logV[x,y,z] = numpy.log(p[1])
                    count += 1

        self.addPersistentVolumeAndRefreshDataTree(v, outputDataIdentifier)

        self.addPersistentVolumeAndRefreshDataTree(logV,
                                        outputDataIdentifier + '_LogProbabilityVolume')
```

Build classifier from examples file (this does not need to be done in parallel)

For each voxel in the input, compute a feature vector and use the classifier to get a probability. Set output voxel to that probability.

Compute a feature vector for a voxel

```
def getPointFeaturesAt(volume, derivativeVolumesIdentifier, gui, point):
    # f is dictionary of features


    if not(isInsideVolumeWithBorder(volume, point, borderWidthForFeatures)):
        raise Exception, 'The point %s is not inside the volume enough. In needs to be away from the border by
%d pixels.' % (point, borderWidthForFeatures)

    f = odict()




    #sizeIdentifiers = ('3x3x3', '5x5x5', '7x7x7')
    sizeIdentifiers = ('(3)', '(5)', '(7)')
    v = [None, None, None]

    #for i in range(3):
    for i in range(1):
        size = i+1
        v = volume[point[0]-size:point[0]+size,point[1]-size:point[1]+size,point[2]-size:point[2]+size]


        #if isInsideVolumeWithBorder(volume, point, border):
        #    # 3 by 3 by 3 chunk of volume
        #    v3 = volume[point[0]-1:point[0]+1,point[1]-1:point[1]+1,point[2]-1:point[2]+1]
        #else:
        #    v3 = zeros([3,3,3])

        #(CUBE_3X3, CUBE_5X5, CUBE_7X7) = (0, 1, 2)

        #i = 0
        #todo: note that getVolume may be a slow operation
        xG = at(gui.getVolume('%s_0Gradient_blur%d' % (derivativeVolumesIdentifier, i)), point)
        yG = at(gui.getVolume('%s_1Gradient_blur%d' % (derivativeVolumesIdentifier, i)), point)
        zG = at(gui.getVolume('%s_2Gradient_blur%d' % (derivativeVolumesIdentifier, i)), point)

        if i == 0:
            f['grayValue'] = at(volume, point)
            #'differenceOfGaussian'
            f['gradientMagnitude'] = sqrt(pow(xG,2) + pow(yG,2) + pow(zG,2))


        stAtSelectedPoint = structureTensor(xG,yG,zG)

        sortedEigAtSelectedPoint = numpy.linalg.eigvals(stAtSelectedPoint)
        sortedEigAtSelectedPoint.sort()

        prefix = sizeIdentifiers[i] + '_'

        f[prefix + 'eig0'] = sortedEigAtSelectedPoint[0]
        f[prefix + 'eig1'] = sortedEigAtSelectedPoint[1]
        f[prefix + 'eig2'] = sortedEigAtSelectedPoint[2]

        values = v.flatten(1)
        #print "i", i, "values", values

        moments = statistics.moments(values)
        f[prefix + 'mean'] = moments[0]
        f[prefix + 'standardDeviation'] = moments[1]
        f[prefix + 'thirdMoment'] = moments[2]
        f[prefix + 'fourthMoment'] = moments[3]

        quantiles = statistics.sortAndReturnQuantiles(values)
        f[prefix + 'minimum'] = quantiles[0]
        f[prefix + '0.25-quantile'] = quantiles[1]
        f[prefix + 'median'] = quantiles[2]
        f[prefix + '0.75-quantile'] = quantiles[3]
        f[prefix + 'maximum'] = quantiles[4]






        # Distribution of the gradient magnitude
        #'gmStandardDeviation'
        #'gmThirdMoment'
        #'gmFourthMoment'
        #'gmMinimum'
        #'gm0.25-quantile'
        #'gmMedian'
        #'gm0.75-quantile'
        #'gmtMaximum'

    return f
```



Input Volume

Feature Computer

Feature Vector

Classifier

Probability

Output Volume