```
classdef CascadeObjectDetector < matlab.System
    %CascadeObjectDetector Detect objects using the Viola-Jones algorithm
    %   DETECTOR = vision.CascadeObjectDetector creates a System object
    %   that detects objects using the Viola-Jones algorithm. The DETECTOR
    %   is capable of detecting a variety of objects, including faces and a
    %   person's upper body. The type of object to detect is controlled by
    %   the ClassificationModel property. By default, the DETECTOR is
    %   configured to detect faces.
    %
    %   DETECTOR = vision.CascadeObjectDetector(MODEL) creates a System
    %   object, DETECTOR, configured to detect objects defined by MODEL.
    %   MODEL is a string describing the type of object to detect. There
    %   are several valid MODEL strings. Examples include
    %   'FrontalFaceCART', 'UpperBody', and 'ProfileFace'.
    %
    %   <a href="matlab:helpview(fullfile(docroot,'toolbox','vision','vision
map'),'vision.CascadeObjectDetector.ClassificationModel')">A list of all available
models is shown in the documentation.</a>
    %
    %   DETECTOR = vision.CascadeObjectDetector(XMLFILE) creates a System
    %   object, DETECTOR, and configures it to use the custom classification
    %   model specified with the XMLFILE input. XMLFILE can be created using
    %   the trainCascadeObjectDetector function or OpenCV training
    %   functionality. You must specify a full or relative path to the
    %   XMLFILE, if it is not on the MATLAB path.
    %
    %   DETECTOR = vision.CascadeObjectDetector(...,Name,Value) configures
    %   the System object properties, specified as one or more name-value
    %   pair arguments. Unspecified properties have default values.
    %
    %   BBOXES = step(DETECTOR, I) performs multi-scale object detection on
    %   the input image, I, and returns, BBOXES, an M-by-4 matrix defining
    %   M bounding boxes containing the detected objects. Each row in
    %   BBOXES is a four-element vector, [x y width height], that specifies
    %   the upper left corner and size of a bounding box in pixels. When no
    %   objects are detected, BBOXES is empty. I must be a grayscale or
    %   truecolor (RGB) image.
    %
    %   [...] = step(DETECTOR, I, ROI) detects objects within the
    %   rectangular search region specified by ROI. ROI must be a 4-element
    %   vector, [x y width height], that defines a rectangular region of
    %   interest within image I. The 'UseROI' property must be true to use
    %   this syntax.
    %
    %   CascadeObjectDetector methods:
    %
    %   step     - See above description for use of this method
    %   release  - Allow property value and input characteristics changes
    %   clone    - Create cascade object detector object with same property
    %              values
    %   isLocked - Locked status (logical)
    %
    %   CascadeObjectDetector properties:
    %
    %   ClassificationModel   - Name of the classification model
```

```
    %   MinSize               - Size of the smallest object to detect
    %   MaxSize               - Size of the biggest object to detect
    %   ScaleFactor           - Scaling for multi-scale object
    %                           detection
    %   MergeThreshold        - Threshold for merging colocated detections
    %   UseROI                - Detect objects within a region of interest
    %
    %   % Example 1:  Face detection
    %   % ----------------------------
    %   faceDetector = vision.CascadeObjectDetector; % Default: finds faces
    %
    %   I = imread('visionteam.jpg');
    %   bboxes = step(faceDetector, I); % Detect faces
    %
    %   % Annotate detected faces
    %   IFaces = insertObjectAnnotation(I, 'rectangle', bboxes, 'Face');
    %   figure, imshow(IFaces), title('Detected faces');
    %
    %   % Example 2: Upper body detection
    %   % ---------------------------------------
    %   bodyDetector = vision.CascadeObjectDetector('UpperBody');
    %   bodyDetector.MinSize = [60 60];
    %   bodyDetector.MergeThreshold = 10;
    %   bodyDetector.UseROI = true;
    %
    %   I2 = imread('visionteam.jpg');
    %
    %   % Search for objects in the top half of the image.
    %   [height, width, ~] = size(I2);
    %   roi = [1 1 width height/2];
    %   bboxBody = step(bodyDetector, I2, roi); % Detect upper bodies
    %
    %   % Annotate detected upper bodies
    %   IBody = insertObjectAnnotation(I2, 'rectangle', ...
    %                                  bboxBody, 'Upper Body');
    %   figure, imshow(IBody), title('Detected upper bodies');
    %
    %   See also trainCascadeObjectDetector, vision.PeopleDetector

    %   Copyright 2011-2015 The MathWorks, Inc.

    %   References:
    %   -----------
    %  [1] Paul Viola and Michael J. Jones "Rapid Object Detection using a
    %      Boosted Cascade of Simple Features" IEEE CVPR, 2001
    %
    %  [2] Rainer Lienhart, Alexander Kuranov, Vadim Pisarevsky
    %      "Empirical Analysis of Detection Cascades of Boosted Classifiers
    %      for Rapid Object Detection", DAGM Symposium for Pattern
    %      Recognition, pp. 297-304, 2003

%#codegen
%#ok<*EMCLS>
%#ok<*EMCA>
    properties(Nontunable)
```

```
        %ClassificationModel A trained cascade classification model
        %   Specify the name of the model as a string. The value specified
        %   for this property may be one of the valid MODEL strings listed
        %   <a href="matlab:helpview(fullfile(docroot,'toolbox','vision','vision.
map'),'vision.CascadeObjectDetector.ClassificationModel')">here</a> or an OpenCV XML
file containing custom classification
        %   model data. When an XML file is specified, a full or relative
        %   path is required if the file is not on the MATLAB path.
        %
        %   Default: 'FrontalFaceCART'
        %
        %   See also <a href="matlab:helpview(fullfile
(docroot,'toolbox','vision','vision.map'),'vision.CascadeObjectDetector.
ClassificationModel')">Available models</a>
        ClassificationModel = 'FrontalFaceCART';
    end
    properties
        %MinSize Size of the smallest object to detect
        %   Specify the size of the smallest object to detect, in pixels,
        %   as a two-element vector, [height width]. Use this property to
        %   reduce computation time when the minimum object size is known
        %   prior to processing the image. When this property is not
        %   specified, the minimum detectable object size is the image size
        %   used to train the classification model. This property is
        %   tunable.
        %
        %   Default: []
        MinSize = [];
        %MaxSize Size of the biggest object to detect
        %   Specify the size of the biggest object to detect, in pixels, as
        %   a two-element vector, [height width]. Use this property to
        %   reduce computation time when the maximum object size is known
        %   prior to processing the image. When this property is not
        %   specified, the maximum detectable object size is SIZE(I). When
        %   'UseROI' is true, the maximum detectable object size is the
        %   defined by the height and width of the ROI. This property is
        %   tunable.
        %
        %   Default: []
        MaxSize = [];
        %ScaleFactor Scaling for multi-scale object detection
        %   Specify the factor used to incrementally scale the detection
        %   scale between MinSize and MaxSize. The ScaleFactor must be
        %   greater than or equal to 1.0001. At each increment, N, the
        %   detection scale is
        %
        %     round(TrainingSize*(ScaleFactor^N))
        %
        %   where TrainingSize is the image size used to train the
        %   classification model. The training size used for each
        %   classification model is shown <a href="matlab:helpview(fullfile
(docroot,'toolbox','vision','vision.map'),'vision.CascadeObjectDetector.
ClassificationModel')">here</a>. This property is tunable.
        %
        %   Default: 1.1
```

```
        ScaleFactor = 1.1;
        %MergeThreshold Threshold for merging colocated detections
        %   Specify a threshold value as a scalar integer. This property
        %   defines the minimum number of colocated detections needed to
        %   declare a final detection. Groups of colocated detections that
        %   meet the threshold are merged to produce one bounding box
        %   around the target object. Increasing this threshold can help
        %   suppress false detections by requiring that the target object
        %   be detected multiple times during the multi-resolution
        %   detection phase. By setting this property to 0, all detections
        %   are returned without merging. This property is tunable.
        %
        %   Default: 4
        MergeThreshold = 4;
    end

    properties(Nontunable, Logical)
        % UseROI Detect objects within a ROI
        %    Set to true to detect objects within a rectangular region of
        %    interest within I.
        %
        % Default: false
        UseROI = false;
    end

    properties (Transient,Access = private)
        pCascadeClassifier; % OpenCV pCascadeClassifier
    end

    properties(Hidden,Dependent,SetAccess = private)
        %TrainingSize Image size used to train classification model
        %   This is the smallest object that the classification model is
        %   trained to detect. It is the smallest object size that the
        %   model can detect.
        TrainingSize;
    end
    methods
        %--------------------------------------------------------------------
        % Constructor
        %--------------------------------------------------------------------
        function obj = CascadeObjectDetector(varargin)
            if (isSimMode())
                obj.pCascadeClassifier = vision.internal.CascadeClassifier;
            else
                obj.pCascadeClassifier =...
                    vision.internal.buildable.cascadeClassifierBuildable↙
cascadeClassifier_construct();
            end
            setProperties(obj,nargin,varargin{:},'ClassificationModel');
            % we need to load XML before calling getClassifierInfo (called
            % from obj.TrainingSize in validatePropertiesImpl
            loadXMLFromClassModel(obj);
            validatePropertiesImpl(obj);
        end
```

```
        %--------------------------------------------------------------------
        % ClassificationModel set method
        %--------------------------------------------------------------------
        function set.ClassificationModel(obj,value)
            coder.extrinsic('exist');

            validateattributes(value,{'char'},{'nonempty','row'},↙
'CascadeObjectDetector');

            % for exist: file must be in path or an absolute path must
            % be provided
            file_exists = coder.internal.const(exist(value,'file')) == 2;
            coder.internal.errorIf(~isSupportedModel(value) && ~file_exists,...
                'vision:ObjectDetector:modelNotFound',value);

            obj.ClassificationModel = value;
            loadXMLFromClassModel(obj);
        end

        %--------------------------------------------------------------------
        % UseROI set method
        %--------------------------------------------------------------------
        function set.UseROI(obj, value)
            obj.UseROI = logical(value);
        end

        %--------------------------------------------------------------------
        % ScaleFactor set method
        %--------------------------------------------------------------------
        function set.ScaleFactor(obj,value)
            validateattributes( value,{'numeric'},...
                {'scalar', '>=',1.0001,'real', 'nonempty','nonsparse','finite'},...
                '','ScaleFactor');
            obj.ScaleFactor = value;
        end

        %--------------------------------------------------------------------
        % MinSize set method
        %--------------------------------------------------------------------
        function set.MinSize(obj,value)
            validateSize('MinSize',value);
            obj.MinSize = value;
        end

        %--------------------------------------------------------------------
        % MaxSize set method
        %--------------------------------------------------------------------
        function set.MaxSize(obj,value)
            validateSize('MaxSize',value);
            obj.MaxSize = value;
        end

        %--------------------------------------------------------------------
        % MergeThreshold set method
        %--------------------------------------------------------------------
```

```
        function set.MergeThreshold(obj,value)
            validateattributes( value,...
                {'numeric'}, {'scalar','>=' 0, 'real','integer',...
                'nonempty','nonsparse','finite'},...
                '','MergeThreshold');
            obj.MergeThreshold = value;
        end

        %--------------------------------------------------------------------
        % TrainingSize get method
        %--------------------------------------------------------------------
        function value = get.TrainingSize(obj)
            if (isSimMode())
                info = obj.pCascadeClassifier.getClassifierInfo();
                value = info.originalWindowSize;
            else
                [originalWindowSize, ~]=vision.internal.buildable↙
cascadeClassifierBuildable.cascadeClassifier_getClassifierInfo(..
                    obj.pCascadeClassifier);
                value = originalWindowSize;
            end
        end
    end

    methods(Access = protected)
        %--------------------------------------------------------------------
        % Cross validate properties
        %--------------------------------------------------------------------
        function validatePropertiesImpl(obj)

            % validate that MinSize is greater than or equal to the minimum
            % object size used to train the classification model

            if ~isempty(obj.MinSize)
                 % obj.TrainingSize calls getClassifierInfo in get.TrainingSize
                coder.internal.errorIf(any(obj.MinSize < obj.TrainingSize),...
                    'vision:ObjectDetector:minSizeLTTrainingSize',...
                    obj.TrainingSize(1),obj.TrainingSize(2));
            end

            % validate the MaxSize is greater than the
            % pModel.TrainingSize when MinSize is not specified
            if isempty(obj.MinSize) && ~isempty(obj.MaxSize)
                % obj.TrainingSize calls getClassifierInfo in get.TrainingSize
                coder.internal.errorIf(any(obj.TrainingSize >= obj.MaxSize),...
                    'vision:ObjectDetector:modelMinSizeGTMaxSize',...
                    obj.TrainingSize(1),obj.TrainingSize(2));
            end

            % validate that MinSize < MaxSize
            if ~isempty(obj.MaxSize) && ~isempty(obj.MinSize)
                 coder.internal.errorIf(any(obj.MinSize >= obj.MaxSize),...
                    'vision:ObjectDetector:minSizeGTMaxSize');
            end
```

```
    end

    %--------------------------------------------------------------------
    % Validate inputs to STEP method
    %--------------------------------------------------------------------
    function validateInputsImpl(obj,I,varargin)
        validateattributes(I,...
            {'uint8','uint16','double','single','int16'},...
            {'real','nonsparse'},...
            '','',2);
        coder.internal.errorIf(~any(ndims(I)==[2 3]),...
            'vision:dims:imageNot2DorRGB');

        sz = size(I);
        coder.internal.errorIf(ndims(I)==3 && sz(3) ~= 3,...
            'vision:dims:imageNot2DorRGB');

        if obj.UseROI
            vision.internal.detector.checkROI(varargin{1},size(I));
        end
    end

    %--------------------------------------------------------------------
    % STEP method implementation
    %--------------------------------------------------------------------
    function bboxes = stepImpl(obj,I,varargin)

        if obj.UseROI
            roi = varargin{1};
        else
            roi = zeros(0,4);
        end

        Iroi = vision.internal.detector.cropImageIfRequested(I, roi, obj.UseROI);

        Iu8  = im2uint8(Iroi);

        grayImage  = convertToGrayscale(obj, Iu8);

        if (isSimMode())

            bboxes = double(obj.pCascadeClassifier.detectMultiScale(grayImage,...
                double(obj.ScaleFactor),...
                uint32(obj.MergeThreshold),...
                int32(obj.MinSize),...
                int32(obj.MaxSize)));
        else

            % OpenCV library works on transposed data
            Iu8_grayT = grayImage';

            if isempty(obj.MinSize)
                obj_MinSize = [0 0];
            else
                obj_MinSize = obj.MinSize;
```

```
                end
                if isempty(obj.MaxSize)
                    obj_MaxSize = [0 0];
                else
                    obj_MaxSize = obj.MaxSize;
                end
                bboxes = vision.internal.buildable.cascadeClassifierBuildable.
cascadeClassifier_detectMultiScale(obj.pCascadeClassifier ,..
                    Iu8_grayT, ...
                    double(obj.ScaleFactor),...
                    uint32(obj.MergeThreshold),...
                    int32(obj_MinSize),...
                    int32(obj_MaxSize));
            end

            bboxes(:,1:2) = vision.internal.detector.addOffsetForROI(bboxes(:,1:2),
roi, obj.UseROI);

        end

        %--------------------------------------------------------------------
        % Release method implementation
        %--------------------------------------------------------------------
        function releaseImpl(obj)
            if ~isSimMode()
                % delete cascadeClassifier object
                vision.internal.buildable.cascadeClassifierBuildable.
cascadeClassifier_deleteObj(obj.pCascadeClassifier);
            end
        end

        %--------------------------------------------------------------------
        % Custom save/load method
        %--------------------------------------------------------------------
        function s = saveObjectImpl(obj)
            s = saveObjectImpl@matlab.System(obj);
        end

        function loadObjectImpl(obj,s, ~)

            coder.extrinsic('exist');

            obj.ScaleFactor = s.ScaleFactor;
            obj.MinSize = s.MinSize;
            obj.MaxSize = s.MaxSize;
            obj.MergeThreshold = s.MergeThreshold;

            invalidModel = false;
            if isfield(s,'ClassificationModel') && ischar(s.ClassificationModel)
                if (~isSupportedModel(s.ClassificationModel))

                    file_exists = coder.internal.const(exist(s.
ClassificationModel,'file')) == 2;
                    if (~file_exists)
                        invalidModel = true;
```

```
                        % error while setting the ClassificationModel
                        % throw a warning and leave ClassificationModel set to default
                        warning(...
                            message('vision:ObjectDetector:modelNotFoundOnLoad',...
                            s.ClassificationModel,'FrontalFaceCART'));
                    end
                end
            end

            if (~invalidModel)
                obj.ClassificationModel = s.ClassificationModel;
            end

            if isfield(s, 'UseROI') % UseROI added in R2015a
                obj.UseROI = s.UseROI;
            else
                obj.UseROI = false;
            end

        end
        %-------------------------------------------------------------------
        % Initialize classification model
        %-------------------------------------------------------------------
        function loadXMLFromClassModel(obj)
            if (isSimMode())
                obj.pCascadeClassifier.load(obj.getModelPath(obj↙
ClassificationModel));
            else
                % append char(0) to get a Null-terminated string.
                ClassificationModelPath = coder.internal.const([obj.getModelPath(obj↙
ClassificationModel) char(0)]);
                vision.internal.buildable.cascadeClassifierBuildable↙
cascadeClassifier_load(...
                    obj.pCascadeClassifier, ClassificationModelPath);
                % for packNGo, we need to put the model file in the zip
                % folder. 'load' function first looks for the model file in
                % the original path, if it is not found there, it searches
                % for the file in the current folder.
                coder.updateBuildInfo('addNonBuildFiles',↙
ClassificationModelPath,'','BlockModules');
            end
        end

        %-------------------------------------------------------------------
        % Return the number of inputs
        %-------------------------------------------------------------------
        function num_inputs = getNumInputsImpl(obj)
            if obj.UseROI
                num_inputs = 2;
            else
                num_inputs = 1;
            end
        end

        %-------------------------------------------------------------------
```

```
        % Return the number of outputs
        %-------------------------------------------------------------------
        function num_outputs = getNumOutputsImpl(~)
            num_outputs = 1;
        end
    end

    methods(Access = private, Hidden)
        %-------------------------------------------------------------------
        % Converts RGB to grayscale if needed.
        %-------------------------------------------------------------------
        function grayImage = convertToGrayscale(~, Iu8)
            if ndims(Iu8) == 3
                if isempty(Iu8)
                    % color space converter does not allow empty.
                    grayImage = Iu8(:,:,1);
                else
                    grayImage = rgb2gray(Iu8);
                end
            else
                grayImage = Iu8;
            end
        end
    end

    methods(Static, Hidden)
        %-------------------------------------------------------------------
        % getModelPath returns full path to model data file
        %-------------------------------------------------------------------
        function file_path = getModelPath(name)
            coder.extrinsic('ctfroot');
            coder.extrinsic('matlabroot');
            coder.extrinsic('fullfile');
            coder.extrinsic('which');

            if isdeployed && isSimMode()
                rootDirectory = coder.internal.const(ctfroot);
            else
                rootDirectory = coder.internal.const(matlabroot);
            end

            dataDirectory = coder.internal.const(fullfil↙
(rootDirectory,'toolbox','vision',...
            'visionutilities','classifierdata','cascade'));

            % Get the path to the model data file so that the data can be
            % loaded using OpenCV

            [isSupportedMdl, filename] = getFileName(name);
            if isSupportedMdl
                %filename = getFileName(name);
                if strncmp(filename,'lbp',3)
                    feature_type ='lbp';
                else
                    feature_type ='haar';
```

```
                end
                % construct full path to file
                file_path = coder.internal.const(fullfile(dataDirectory,feature_type,
filename));
            else
                % custom file supplied; 3 options:
                % just file name which is on the path: use which to get full path
                % file name with relative or absolute path: just use user's input

                % determine if it is on the path
                file_fullPath = coder.internal.const(which(name));
                if isempty(file_fullPath) % not on the path nor in current directory
                    file_path = coder.internal.const(name);% must be full/relative
path
                else
                    file_path = file_fullPath;
                end

            end
        end
    end
end % of classdef

%--------------------------------------------------------------------------
% Validation for MinSize and MaxSize
%--------------------------------------------------------------------------
function validateSize(prop,value)

% By default MaxSize/MinSize is [], and it can be set to empty too.
validateattributes( value,...
    {'numeric'}, {'real','nonsparse','finite','2d','integer', '>=',0},...
    '',prop);
% Using 'vector',2 in validateattributes fails for [] so the
% following check makes sure that MaxSize has 2-elements
coder.internal.errorIf(~isempty(value) && (numel(value) ~= 2),...
                    'vision:ObjectDetector:invalidSize',prop);
end

%==========================================================================
function flag = isSupportedModel(name)
    if (strcmpi(name, 'FrontalFaceCART') ...
     || strcmpi(name, 'FrontalFaceLBP') ...
     || strcmpi(name, 'ProfileFace') ...
     || strcmpi(name, 'Mouth') ...
     || strcmpi(name, 'Nose') ...
     || strcmpi(name, 'EyePairBig') ...
     || strcmpi(name, 'EyePairSmall') ...
     || strcmpi(name, 'RightEye') ...
     || strcmpi(name, 'LeftEye') ...
     || strcmpi(name, 'RightEyeCART') ...
     || strcmpi(name, 'LeftEyeCART') ...
     || strcmpi(name, 'UpperBody'))
        flag = true;
    else
        flag = false;
```

```
    end
end

%==========================================================================
function [isSupportedMdl, filename] = getFileName(name)

    if strcmpi(name, 'FrontalFaceCART')
        isSupportedMdl = true;
        filename = 'haarcascade_frontalface_alt2.xml';
    elseif strcmpi(name, 'FrontalFaceLBP')
        isSupportedMdl = true;
        filename = 'lbpcascade_frontalface.xml';
    elseif strcmpi(name, 'ProfileFace')
        isSupportedMdl = true;
        filename = 'haarcascade_profileface.xml';
    elseif strcmpi(name, 'Mouth')
        isSupportedMdl = true;
        filename = 'haarcascade_mcs_mouth.xml';
    elseif strcmpi(name, 'Nose')
        isSupportedMdl = true;
        filename = 'haarcascade_mcs_nose.xml';
    elseif strcmpi(name, 'EyePairBig')
        isSupportedMdl = true;
        filename = 'haarcascade_mcs_eyepair_big.xml';
    elseif strcmpi(name, 'EyePairSmall')
        isSupportedMdl = true;
        filename = 'haarcascade_mcs_eyepair_small.xml';
    elseif strcmpi(name, 'RightEye')
        isSupportedMdl = true;
        filename = 'haarcascade_mcs_righteye.xml';
    elseif strcmpi(name, 'LeftEye')
        isSupportedMdl = true;
        filename = 'haarcascade_mcs_lefteye.xml';
    elseif strcmpi(name, 'RightEyeCART')
        isSupportedMdl = true;
        filename = 'haarcascade_righteye_2splits.xml';
    elseif strcmpi(name, 'LeftEyeCART')
        isSupportedMdl = true;
        filename = 'haarcascade_lefteye_2splits.xml';
    elseif strcmpi(name, 'UpperBody')
        isSupportedMdl = true;
        filename = 'haarcascade_mcs_upperbody.xml';
    else
        isSupportedMdl = false;
        filename = '';
    end
end

%==========================================================================
function flag = isSimMode()
    flag = isempty(coder.target);
end
```