Need to go to cascadedetect.cpp, add the following code:

struct DTNode\_

{

int featureIdx;

int left;

int right;

};

struct ObjectModel{

int sizeX, sizeY;

int nStages;

int maxCatCount;

int maxDepth;

int nFeatures;

std::vector<float>stageThreshold;

std::vector<int>nStageWeekClassifierCount;

std::vector<DTNode\_>node;

std::vector<int>cascadeSubsets;

std::vector<float>cascadeLeaves;

std::vector<Rect>featureRect;

};

CascadeClassifierImpl::Data::Data()

{

stageType = featureType = ncategories = maxNodesPerTree = 0;

}

#if 0

bool CascadeClassifierImpl::Data::read(const FileNode &root)

{

static const float THRESHOLD\_EPS = 1e-5f;

Model\_ fdModel;

// load stage params

String stageTypeStr = (String)root[CC\_STAGE\_TYPE];

if( stageTypeStr == CC\_BOOST )

stageType = BOOST;

else

return false;

String featureTypeStr = (String)root[CC\_FEATURE\_TYPE];

if( featureTypeStr == CC\_HAAR )

featureType = FeatureEvaluator::HAAR;

else if( featureTypeStr == CC\_LBP )

featureType = FeatureEvaluator::LBP;

else if( featureTypeStr == CC\_HOG )

{

featureType = FeatureEvaluator::HOG;

CV\_Error(Error::StsNotImplemented, "HOG cascade is not supported in 3.0");

}

else

return false;

origWinSize.width = (int)root[CC\_WIDTH];

origWinSize.height = (int)root[CC\_HEIGHT];

CV\_Assert( origWinSize.height > 0 && origWinSize.width > 0 );

// load feature params

FileNode fn = root[CC\_FEATURE\_PARAMS];

if( fn.empty() )

return false;

ncategories = fn[CC\_MAX\_CAT\_COUNT];

int subsetSize = (ncategories + 31)/32,

nodeStep = 3 + ( ncategories>0 ? subsetSize : 1 );

// load stages

fn = root[CC\_STAGES];

if( fn.empty() )

return false;

stages.reserve(fn.size());

classifiers.clear();

nodes.clear();

stumps.clear();

FileNodeIterator it = fn.begin(), it\_end = fn.end();

minNodesPerTree = INT\_MAX;

maxNodesPerTree = 0;

for( int si = 0; it != it\_end; si++, ++it )

{

FileNode fns = \*it;

Stage stage;

stage.threshold = (float)fns[CC\_STAGE\_THRESHOLD] - THRESHOLD\_EPS;

fns = fns[CC\_WEAK\_CLASSIFIERS];

if(fns.empty())

return false;

stage.ntrees = (int)fns.size();

stage.first = (int)classifiers.size();

stages.push\_back(stage);

classifiers.reserve(stages[si].first + stages[si].ntrees);

FileNodeIterator it1 = fns.begin(), it1\_end = fns.end();

for( ; it1 != it1\_end; ++it1 ) // weak trees

{

FileNode fnw = \*it1;

FileNode internalNodes = fnw[CC\_INTERNAL\_NODES];

FileNode leafValues = fnw[CC\_LEAF\_VALUES];

if( internalNodes.empty() || leafValues.empty() )

return false;

DTree tree;

tree.nodeCount = (int)internalNodes.size()/nodeStep;

minNodesPerTree = std::min(minNodesPerTree, tree.nodeCount);

maxNodesPerTree = std::max(maxNodesPerTree, tree.nodeCount);

classifiers.push\_back(tree);

nodes.reserve(nodes.size() + tree.nodeCount);

leaves.reserve(leaves.size() + leafValues.size());

if( subsetSize > 0 )

subsets.reserve(subsets.size() + tree.nodeCount\*subsetSize);

FileNodeIterator internalNodesIter = internalNodes.begin(), internalNodesEnd = internalNodes.end();

for( ; internalNodesIter != internalNodesEnd; ) // nodes

{

DTreeNode node;

node.left = (int)\*internalNodesIter; ++internalNodesIter;

node.right = (int)\*internalNodesIter; ++internalNodesIter;

node.featureIdx = (int)\*internalNodesIter; ++internalNodesIter;

if( subsetSize > 0 )

{

for( int j = 0; j < subsetSize; j++, ++internalNodesIter )

subsets.push\_back((int)\*internalNodesIter);

node.threshold = 0.f;

}

else

{

node.threshold = (float)\*internalNodesIter; ++internalNodesIter;

}

nodes.push\_back(node);

}

internalNodesIter = leafValues.begin(), internalNodesEnd = leafValues.end();

for( ; internalNodesIter != internalNodesEnd; ++internalNodesIter ) // leaves

leaves.push\_back((float)\*internalNodesIter);

}

}

if( maxNodesPerTree == 1 )

{

int nodeOfs = 0, leafOfs = 0;

size\_t nstages = stages.size();

for( size\_t stageIdx = 0; stageIdx < nstages; stageIdx++ )

{

const Stage& stage = stages[stageIdx];

int ntrees = stage.ntrees;

for( int i = 0; i < ntrees; i++, nodeOfs++, leafOfs+= 2 )

{

const DTreeNode& node = nodes[nodeOfs];

stumps.push\_back(Stump(node.featureIdx, node.threshold,

leaves[leafOfs], leaves[leafOfs+1]));

}

}

}

return true;

}

#else

using namespace std;

bool CascadeClassifierImpl::Data::read(const FileNode &root)

{

static const float THRESHOLD\_EPS = 1e-5f;

ObjectModel objModel;

// load stage params

string stageTypeStr = (string)root[CC\_STAGE\_TYPE];

if( stageTypeStr == CC\_BOOST )

stageType = BOOST;

else

return false;

string featureTypeStr = (string)root[CC\_FEATURE\_TYPE];

if( featureTypeStr == CC\_HAAR )

featureType = FeatureEvaluator::HAAR;

else if( featureTypeStr == CC\_LBP )

featureType = FeatureEvaluator::LBP;

else if( featureTypeStr == CC\_HOG )

featureType = FeatureEvaluator::HOG;

else

return false;

origWinSize.width = (int)root[CC\_WIDTH];

origWinSize.height = (int)root[CC\_HEIGHT];

CV\_Assert( origWinSize.height > 0 && origWinSize.width > 0 );

objModel.sizeX = origWinSize.width;

objModel.sizeY = origWinSize.height;

//isStumpBased = (int)(root[CC\_STAGE\_PARAMS][CC\_MAX\_DEPTH]) == 1 ? true : false;

objModel.maxDepth = (int)(root[CC\_STAGE\_PARAMS][CC\_MAX\_DEPTH]);

// load feature params

FileNode fn = root[CC\_FEATURE\_PARAMS];

if( fn.empty() )

return false;

ncategories = fn[CC\_MAX\_CAT\_COUNT];

int subsetSize = (ncategories + 31)/32,

nodeStep = 3 + ( ncategories>0 ? subsetSize : 1 );

objModel.maxCatCount = ncategories;

// load stages

fn = root[CC\_STAGES];

if( fn.empty() )

return false;

objModel.nStages = fn.size();

stages.reserve(fn.size());

classifiers.clear();

nodes.clear();

FileNodeIterator it = fn.begin(), it\_end = fn.end();

for( int si = 0; it != it\_end; si++, ++it )

{

FileNode fns = \*it;

Stage stage;

stage.threshold = (float)fns[CC\_STAGE\_THRESHOLD] - THRESHOLD\_EPS;

fns = fns[CC\_WEAK\_CLASSIFIERS];

if(fns.empty())

return false;

stage.ntrees = (int)fns.size();

stage.first = (int)classifiers.size();

stages.push\_back(stage);

classifiers.reserve(stages[si].first + stages[si].ntrees);

objModel.stageThreshold.push\_back( stage.threshold);

objModel.nStageWeekClassifierCount.push\_back( stage.ntrees);

FileNodeIterator it1 = fns.begin(), it1\_end = fns.end();

for( ; it1 != it1\_end; ++it1 ) // weak trees

{

FileNode fnw = \*it1;

FileNode internalNodes = fnw[CC\_INTERNAL\_NODES];

FileNode leafValues = fnw[CC\_LEAF\_VALUES];

if( internalNodes.empty() || leafValues.empty() )

return false;

DTree tree;

tree.nodeCount = (int)internalNodes.size()/nodeStep;

classifiers.push\_back(tree);

nodes.reserve(nodes.size() + tree.nodeCount);

leaves.reserve(leaves.size() + leafValues.size());

if( subsetSize > 0 )

subsets.reserve(subsets.size() + tree.nodeCount\*subsetSize);

FileNodeIterator internalNodesIter = internalNodes.begin(), internalNodesEnd = internalNodes.end();

for( ; internalNodesIter != internalNodesEnd; ) // nodes

{

DTreeNode node;

node.left = (int)\*internalNodesIter; ++internalNodesIter;

node.right = (int)\*internalNodesIter; ++internalNodesIter;

node.featureIdx = (int)\*internalNodesIter; ++internalNodesIter;

DTNode\_ node\_;

node\_.left = node.left;

node\_.right = node.right;

node\_.featureIdx = node.featureIdx;

objModel.node.push\_back( node\_);

if( subsetSize > 0 )

{

for( int j = 0; j < subsetSize; j++, ++internalNodesIter ){

subsets.push\_back((int)\*internalNodesIter);

objModel.cascadeSubsets.push\_back( (int)\*internalNodesIter);

}

node.threshold = 0.f;

}

else

{

node.threshold = (float)\*internalNodesIter; ++internalNodesIter;

}

nodes.push\_back(node);

}

internalNodesIter = leafValues.begin(), internalNodesEnd = leafValues.end();

for( ; internalNodesIter != internalNodesEnd; ++internalNodesIter ){ // leaves

leaves.push\_back((float)\*internalNodesIter);

objModel.cascadeLeaves.push\_back((float)\*internalNodesIter);

}

}

}

bool writeModel = true;

if( writeModel){

FileNode fn1 = root[CC\_FEATURES];

FileNodeIterator it1 = fn1.begin(), it\_end2 = fn1.end();

for(int i = 0; it1 != it\_end2; ++it1, i++)

{

LBPEvaluator::Feature feat;

feat.read( \*it1);

objModel.featureRect.push\_back( feat.rect);

}

std::ofstream o1("C:/Jianfeng/OpenCV/CVRD/Researchs/bb/FaceProject/objectDetection/data/stageThreshold.txt");

o1<<objModel.stageThreshold.size()<<"\n";

for(int i=0;i<objModel.stageThreshold.size();i++){

//o1<<fdModel.stageThreshold[i]<<"f, ";

o1<<cvRound( objModel.stageThreshold[i] \* 10000)<<", ";

}

o1.close();

std::ofstream o2("C:/Jianfeng/OpenCV/CVRD/Researchs/bb/FaceProject/objectDetection/data/nStageWeekClassifierCount.txt");

o2<<objModel.nStageWeekClassifierCount.size()<<"\n";

for(int i=0;i<objModel.nStageWeekClassifierCount.size();i++)

o2<<objModel.nStageWeekClassifierCount[i]<<", ";

o2.close();

std::ofstream o3("C:/Jianfeng/OpenCV/CVRD/Researchs/bb/FaceProject/objectDetection/data/cascadeInternalNodeValues.txt");

o3<<objModel.cascadeSubsets.size()<<"\n";

for(int i=0;i<objModel.cascadeSubsets.size();i++){

o3<<objModel.cascadeSubsets[i]<<", ";

}

o3.close();

std::ofstream o4("C:/Jianfeng/OpenCV/CVRD/Researchs/bb/FaceProject/objectDetection/data/cascadeLeaves.txt");

o4<<objModel.cascadeLeaves.size()<<"\n";

for(int i=0;i<objModel.cascadeLeaves.size();i++){

//o4<<fdModel.cascadeLeaves[i]<<"f, ";

o4<<cvRound( objModel.cascadeLeaves[i] \* 10000)<<", ";

}

o4.close();

std::ofstream o5("C:/Jianfeng/OpenCV/CVRD/Researchs/bb/FaceProject/objectDetection/data/nodeFeatureIndex.txt");

o5<<objModel.node.size()<<"\n";

for(int i=0;i<objModel.node.size();i++)

o5<<objModel.node[i].featureIdx<<", ";

o5.close();

std::ofstream o6("C:/Jianfeng/OpenCV/CVRD/Researchs/bb/FaceProject/objectDetection/data/nodeLeft.txt");

o6<<objModel.node.size()<<"\n";

for(int i=0;i<objModel.node.size();i++)

o6<<objModel.node[i].left<<", ";

o6.close();

std::ofstream o7("C:/Jianfeng/OpenCV/CVRD/Researchs/bb/FaceProject/objectDetection/data/nodeRight.txt");

o7<<objModel.node.size()<<"\n";

for(int i=0;i<objModel.node.size();i++)

o7<<objModel.node[i].right<<", ";

o7.close();

std::ofstream o8("C:/Jianfeng/OpenCV/CVRD/Researchs/bb/FaceProject/objectDetection/data/featureRectX.txt");

o8<<objModel.featureRect.size()<<"\n";

for(int i=0;i<objModel.featureRect.size();i++)

o8<<objModel.featureRect[i].x<<", ";

o8.close();

std::ofstream o9("C:/Jianfeng/OpenCV/CVRD/Researchs/bb/FaceProject/objectDetection/data/featureRectY.txt");

o9<<objModel.featureRect.size()<<"\n";

for(int i=0;i<objModel.featureRect.size();i++)

o9<<objModel.featureRect[i].y<<", ";

o9.close();

std::ofstream o10("C:/Jianfeng/OpenCV/CVRD/Researchs/bb/FaceProject/objectDetection/data/featureRectW.txt");

o10<<objModel.featureRect.size()<<"\n";

for(int i=0;i<objModel.featureRect.size();i++)

o10<<objModel.featureRect[i].width<<", ";

o10.close();

std::ofstream o11("C:/Jianfeng/OpenCV/CVRD/Researchs/bb/FaceProject/objectDetection/data/featureRectH.txt");

o11<<objModel.featureRect.size()<<"\n";

for(int i=0;i<objModel.featureRect.size();i++)

o11<<objModel.featureRect[i].height<<", ";

o11.close();

}

return true;

}

#endif