UCCC 3074 Image Processing and Pattern Recognition

Assignment

Course	**Computer Science**
Group Member (Student ID)	**Joshua Chan Mun Wei** — **1303528** **Lee Wen Dick** — **1401837**
Lecturer	**Prof Kar Hang Leung**

Introduction

There are different types of fishes found in a river. Fisherman deliver these fishes to a fish processing factory every day. Different processes will be applied to different types of fishes. The first step of processing is to sort the incoming fishes. We are assigned to design and implement an automatic fish sorting system using a video camera. In this system, we train a Support Vector Machine (SVM) for fish types classification with a set of 20 training samples for each fish type. We recognize 8 types of fishes which are Black Crappie, Channel Catfish, Longear Sunfish, Sturgeon, Tuna, Walleye, White Bass and Yellow Perch.

Problem Statement

Our system should be acceptably accurate and robust to be used in a controlled environment such as an industrial factory. Therefore, the object recognition system should be able to handle minor variations of the captured image. Ideally, it should be translation invariant, scale invariant and rotation invariant.

Input: An image of a fish captured side-view with a lightened background (white).
Output: The classification of the fish.

System Design

Image Input and Preprocessing(Image collection by Joshua Chan and Lee Wen Dick)

Preprocessing prepares and improves the quality of our input image, making it more suitable for higher level processes.

Our system requires the input image to have a foreground fish object of side view angle and a plain white background. Our system supports image file format of PNG or JPEG. Upon input of an image, we perform Gaussian Blur to smoothen the image to reduce unwanted noises.

Segmentation(Lee Wen Dick)

The first important task is to segment the fish from the input image. We use the Canny edge detection algorithm to obtain the edge image. Then, we perform 3 iterations of dilation on the image to expand the pixel; this helps in generalizing the input image and boost the connectivity of the image.

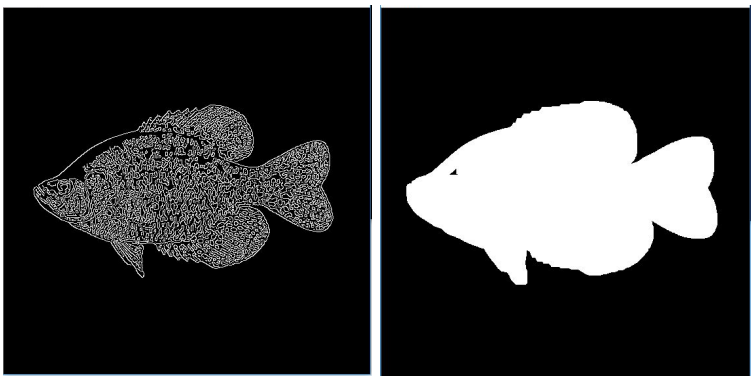

Figure 1 Canny edge image (left) and segmented image (right)

We want to find a binary mask that separates the fish from the background as show as the right image in Figure 1. Notice that the boundary of the fish in the canny image is white and distinct from its black background. Unfortunately, even though the boundary has been nicely extracted, the interior of the fish has intensities similar to the background. Therefore, the thresholding operation cannot distinguish it from the background. To fill all the pixel inside the boundary of fish with white, our system uses the Flood Fill algorithm.

The algorithm is set to start filling the background from seed pixel at (0, 0). After the filling is completed, we invert the flood filled image and combine it with the initial canny image using bitwise OR operation to obtain the final foreground mask with holes filled in.

Rotate(Joshua Chan Mun Wei)

With the segmented image obtained, our system computes the contour with the largest area for the given image. Principal component analysis (PCA) is done using the points of the contour to find out the eigenvalues and eigenvectors of the image, which is then used to calculate the orientation of the image. The image is rotated to its principal axis, this is to ensure that our feature points are rotation-invariant.

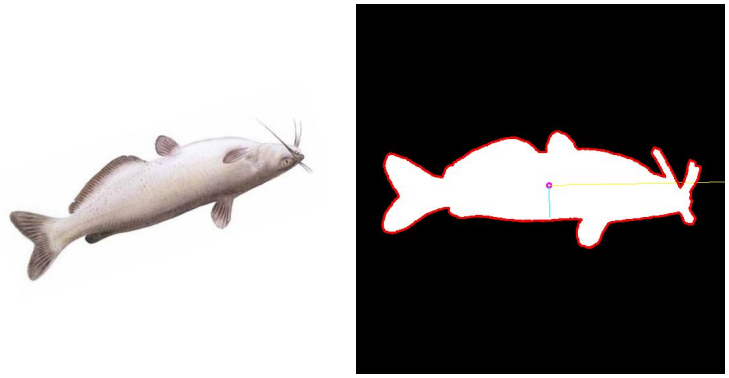

Figure 2 Input image is aligned to the principal axis

Scale(Joshua Chan Mun Wei)

After alignment of the segmented image, the contours are recalculated and used to form a bounding box; enclosing the subject. The image is then cropped about the bounding box and scaled to a size of *500x500* pixels. This helps to ensure that the canvas size would not affect the features of the image (scale-invariant).

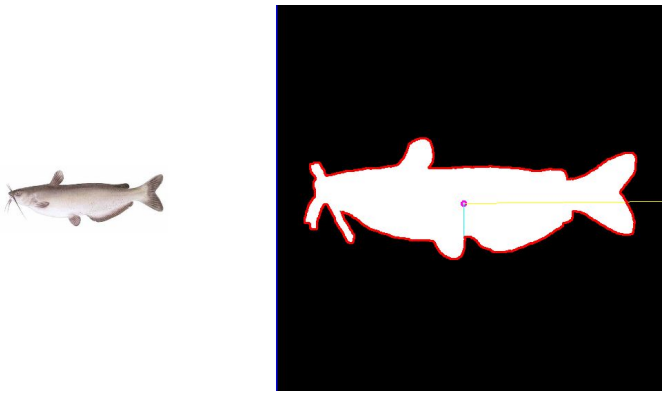

Figure 3 Small fish having large canvas size is normalized to *500x500* pixels

Feature Extraction(Hu Moments by Lee Wen Dick, Contour perimeter and area by Joshua)

Feature extraction is one of the most important part in object recognition. The uniqueness and reproducibility of the features ultimately determines the power to distinguish different types of models.

For our system, we utilize Hu moments of the input images as feature points. It is observed that calculating the Hu moments from the entire segmented image or largest calculated contour yields good results. Nevertheless, we see even better results when calculating Hu moments from the polygonal approximation of the segmented image.

The improvements of using the polygonal approximation over the calculated image contour could be attributed to overfitting. Since polygonal approximations contains less contour points, it gives a more general contour on the specific fish species; thereby giving flexibility to identify the fish image that vary on a case by case basis.

Besides that, we also include the contour perimeter and area as features to train our SVM. Since the processed image is translation, scale and rotation invariant, these feature points give a fairly distinctive representation of the fish species.

These features are calculated for every sample in the training dataset, the results are shown in Table 1. We find that the *Coefficient of Variation (CV)* which is the ratio of standard deviation to the mean is relatively high in Hu moments 3 - 7. Typically, we want low variability in the features describing a model since it means the feature is seen in most of the samples. Hence, we only used the first and second Hu moments, contour perimeter and area as features to train the SVM.

Table 1: Calculated Features For 20 *Black Crappie* Training Samples

	I_1	I_2	I_3	I_4	I_5	I_6	I_7	$c_{Perimeter}$	c_{Area}
1	0.202399	0.012612	2.86E-05	3.47E-06	3.45E-11	1.26E-07	-1.85E-12	1458.63	84350.5
2	0.222811	0.018397	9.90E-05	0.000111	1.13E-08	1.47E-05	3.08E-09	1485.51	76212.5
3	0.20954	0.014016	5.69E-05	1.83E-05	5.45E-10	1.70E-06	2.27E-10	1456.83	77760
4	0.225652	0.021006	9.14E-05	6.38E-05	4.51E-09	8.24E-06	1.85E-09	1511.39	73390.5
5	0.219717	0.01832	3.60E-05	3.84E-05	1.42E-09	5.17E-06	-1.19E-10	1520.25	75188
6	0.214128	0.016666	3.51E-06	1.85E-05	1.31E-10	2.38E-06	7.07E-11	1421.87	76986
7	0.21976	0.019924	0.000114	4.03E-05	2.72E-09	4.06E-06	-2.21E-10	1395.66	73923.5
8	0.219824	0.019739	1.21E-05	4.64E-06	3.08E-11	3.42E-07	-1.58E-11	1420.59	73885.5
9	0.215372	0.017887	4.99E-05	2.61E-05	9.02E-10	3.05E-06	2.78E-10	1310.84	74336.5
10	0.209559	0.015006	4.35E-05	1.32E-05	2.72E-10	1.35E-06	1.61E-10	1361.71	81419
11	0.217697	0.016846	6.11E-05	4.50E-05	1.78E-09	5.68E-06	1.55E-09	1402.17	78656
12	0.21672	0.01858	4.70E-05	2.99E-05	1.11E-09	3.50E-06	1.77E-10	1398.11	73713
13	0.202734	0.012753	4.81E-05	3.09E-06	3.01E-11	-3.64E-08	2.25E-11	1411.02	83462
14	0.212094	0.015541	3.15E-05	2.66E-05	7.27E-10	3.11E-06	2.54E-10	1388.21	75703
15	0.208833	0.014906	1.13E-05	2.50E-05	4.13E-10	3.05E-06	8.12E-11	1480.02	81804.5
16	0.202917	0.012735	3.65E-05	5.39E-06	7.48E-11	2.98E-07	1.04E-11	1463.93	84130.5
17	0.213511	0.016987	3.10E-07	5.44E-06	3.15E-12	7.09E-07	6.32E-12	1383.16	74906
18	0.206723	0.013617	8.34E-05	2.18E-05	8.65E-10	1.93E-06	3.37E-10	1450.65	77475
19	0.20125	0.011569	6.60E-05	9.94E-06	2.43E-10	5.97E-07	7.52E-11	1443.6	81550.5
20	0.203162	0.011744	0.000289	0.000124	2.31E-08	1.27E-05	-3.26E-09	1508.31	80686
mean, μ	0.21222	0.015942	6.04E-05	3.17E-05	2.51E-09	3.63E-06	2.28E-10	1433.62	77976.9
SD, σ	0.007281	0.002831	6.05E-05	3.26E-05	5.35E-09	3.95E-06	1.13E-09	52.8264	3660.99
σ / μ	0.03431	0.177577	1.001726	1.027727	2.131255	1.088044	4.93338	0.036848	0.04695

Recognition and Interpretation(SVM by Lee Wen Dick, Normalisation by Joshua Chan)

After we identify the features that are unique to represent the fish species, we are ready to feed the data obtained into the system to train the SVM. To obtain a more accurate prediction, we perform normalisation on the data by computing the mean and standard deviation. Then, the normalised data is calculated with formula below:

$$X_{new} = \frac{X - \mu}{\sigma}$$

Originally, SVM is a technique for building an optimal binary classifier. The technique was extended to regression and clustering problems. SVM is a partial case of kernel-based method. It maps feature vector into a higher-dimensional space using a kernel function and builds an optimal linear discriminating function in this space or an optimal hyper-plane that fits into the training data. Our SVM is defined using a kernel type of Radial basis function (RBF), which is a good choice in most cases. The linear kernel type offers the fastest option but offers lower accuracy in prediction in return. Since we have 4 features to train the SVM, we use the C-Support Vector Classification, which is a n-class classification that allows imperfect separation of classes with penalty multiplier c for outliers.

Methodology and Tools

Block Diagram

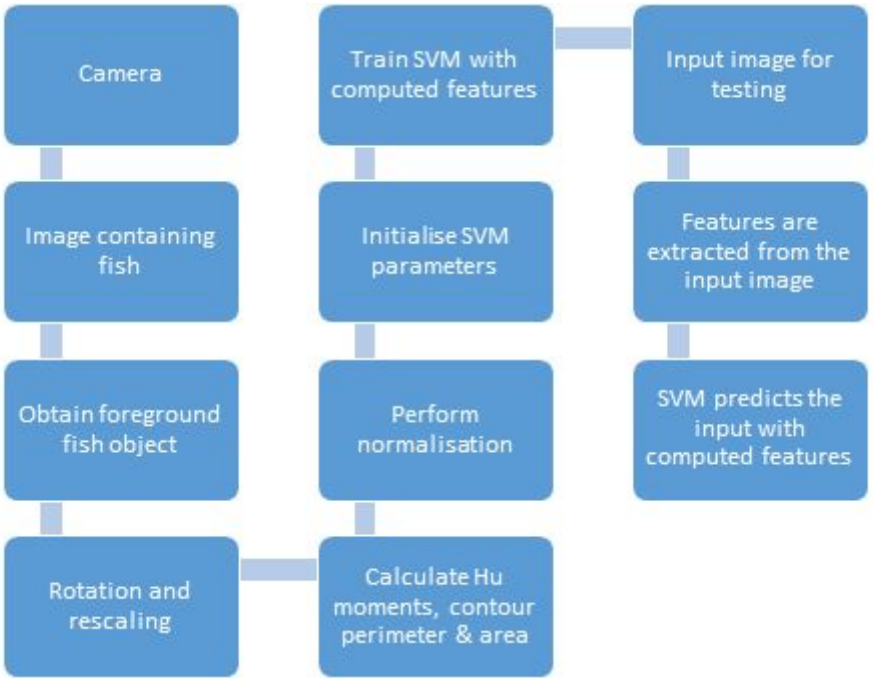

Figure 3 Block Diagram of fish recognition system

Tools

Language: C++ with OpenCV 3.0

IDE: Visual Studio 2013

Source file required: “Fishes” folder attached

Directory: Inputs\Fishes

Steps:

1. Copy the “Fishes” folder into your project solution.
2. Copy the source code attached into source file.
3. To input testing image, navigate to Fishes\Testing\[Fish Species] and paste the image there.
4. Run the program. Output will be shown in the console.

Testing and Results

After our system trained the SVM, we prepared about 90 testing fishes to identify the accuracy of our prediction by SVM:

Fish Type	Result
Black Crappie	10/10 (100%)
Channel Catfish	18/19 (94.73%)
Tuna	11/11 (100%)
Longear Sunfish	11/11 (100%)
Sturgeon	10/10 (100%)
Walleye	9/10 (90%)
White Bass	8/10 (80%)
Yellow Perch	8/9 (88.89%)

Overall accuracy: 85/90, 94.44% accuracy

There are few wrong predictions by SVM on few fish species such as White Bass and Yellow Perch. The fish species are rather similar and more training data can be introduced to achieve better prediction. Besides, if we are provided sufficient image samples of the real fish, then we can further identify more unique features such as texture information and color information. Our system is able to predict fish samples on different scales and rotation as well. Furthermore, our system supports multiple image format such as JPG and PNG but not GIF. However, our system fails when the input image contains noisy and complicated background and different skew angles of fish from front view, bottom view and top view.

Conclusion

We present an automated fish recognition system that utilises an SVM trained using 4 calculated features from captured side-view images of fishes:

1. Hu Moment 1
2. Hu Moment 2
3. Contour perimeter
4. Contour area.

The system can be used in a controlled environment such as an industrial factory. Therefore, our fish recognition system is capable of handling minor variations of the captured image and is invariant under translation, rotation and scaling. Results are optimistic, the system is able to detect at least 8 fish species and identify 85 out of 90 test samples (94.44%). To achieve even better accuracy, we suggest detecting features such as texture information and color information from real fish images.

APPENDICES

Source code

```
#include	<iostream>
#include	<opencv2/opencv.hpp>
#include	<opencv2/highgui/highgui.hpp>
#include	"Supp.h"
#include	<string>

using namespace std;
using namespace cv;
using namespace ml;

// Parameters
const int POLY_APPROX_EPSILON = 11;
const vector<string> folderName = { "Black Crappie", "Channel Catfish", "Tuna", "Longear Sunfish", "Sturgeon", "Walleye", "White Bass", "Yellow Perch" };

// Globals
int const	noOfImagePerCol = 1, noOfImagePerRow = 2; // create window partition
Mat		largeWin,	win[noOfImagePerRow	*	noOfImagePerCol], legend[noOfImagePerRow * noOfImagePerCol];

void getLargestContour(Mat &img, vector<vector<Point> > &largestContour) {
	// Extract contours of the foreground image
	vector<vector<Point> > contoursH;
	vector<Vec4i> hierarchyH;
	findContours(img, contoursH, hierarchyH, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE);
	// Extract contour with largest contour
	double largest_area = 0;
	int largest_contour_index;
	for (int i = 0; i < contoursH.size(); i++)
	{
		double area = contourArea(contoursH[i]);
		if (area > largest_area) { // If the area of contour > current largest contour
			largest_area = area;
			largest_contour_index = i; // Store the index of largest contour
		}
	}
	largestContour[0] = contoursH[largest_contour_index];
}

Mat rotateMat(Mat &img, double angle) {
	double offsetX, offsetY;
	double width = img.size().width;
	double height = img.size().height;
	Point2d center = Point2d(width / 2, height / 2);
	RotatedRect rotatedBounds = RotatedRect(center, img.size(), angle);
	Rect bounds = rotatedBounds.boundingRect();
	Mat resized = Mat::zeros(bounds.size(), img.type());
	offsetX = (bounds.width - width) / 2;
	offsetY = (bounds.height - height) / 2;
	Rect roi = Rect(offsetX, offsetY, width, height);
	img.copyTo(resized(roi));
	center += Point2d(offsetX, offsetY);
	Mat M = getRotationMatrix2D(center, angle, 1.0);
	warpAffine(resized, resized, M, resized.size());

	return resized;
}
```

```
double getOrientation(vector<Point> &pts, Mat &img, bool draw = true)
{
	//Construct a buffer used by the pca analysis
	Mat data_pts = Mat(pts.size(), 2, CV_64FC1);
	for (int i = 0; i < data_pts.rows; ++i)
	{
		data_pts.at<double>(i, 0) = pts[i].x;
		data_pts.at<double>(i, 1) = pts[i].y;
	}
	//Perform PCA analysis
	PCA pca_analysis(data_pts, Mat(), CV_PCA_DATA_AS_ROW);
	//Store the position of the object
	Point	pos	=	Point(pca_analysis.mean.at<double>(0,	0),
pca_analysis.mean.at<double>(0, 1));
	//Store the eigenvalues and eigenvectors
	vector<Point2d> eigen_vecs(2);
	vector<double> eigen_val(2);
	for (int i = 0; i < 2; ++i)
	{
		eigen_vecs[i] = Point2d(pca_analysis.eigenvectors.at<double>(i, 0),
			pca_analysis.eigenvectors.at<double>(i, 1));
		eigen_val[i] = pca_analysis.eigenvalues.at<double>(i, 0);
	}
	// Draw the principal components
	if (draw) {
		circle(img, pos, 3, CV_RGB(255, 0, 255), 2);
		line(img,	pos,	pos	+	0.02	*	Point(eigen_vecs[0].x	*	eigen_val[0],
eigen_vecs[0].y * eigen_val[0]), CV_RGB(255, 255, 0));
		line(img,	pos,	pos	+	0.02	*	Point(eigen_vecs[1].x	*	eigen_val[1],
eigen_vecs[1].y * eigen_val[1]), CV_RGB(0, 255, 255));
	}

	return atan2(eigen_vecs[0].y, eigen_vecs[0].x) * (180.0 / CV_PI);
}

/*!
* \brief Enlarge an ROI rectangle by a specific amount if possible
* \param from The image the ROI will be set on
* \param boundingBox The current boundingBox
* \param padding The amount of padding around the boundingbox
* \return The enlarged ROI as far as possible
*/
Rect enlargeROI(Mat frm, Rect boundingBox, int padding) {
	Rect	returnRect	=	Rect(boundingBox.x	-	padding,	boundingBox.y	-	padding,
boundingBox.width + (padding * 2), boundingBox.height + (padding * 2));
	if (returnRect.x < 0)returnRect.x = 0;
	if (returnRect.y < 0)returnRect.y = 0;
	if	(returnRect.x	+	returnRect.width	>=	frm.cols)returnRect.width	=	frm.cols	-
returnRect.x;
	if	(returnRect.y	+	returnRect.height	>=	frm.rows)returnRect.height	=	frm.rows	-
returnRect.y;
	return returnRect;
}

Mat getSquareImage(const Mat& img, int target_width = 500) {
	int width = img.cols,
		height = img.rows;

	Mat square = Mat(target_width, target_width, img.type(), img.at<Vec3b>(0, 0));
	int max_dim = (width >= height) ? width : height;
	float scale = ((float)target_width) / max_dim;
	Rect roi;
	if (width >= height)
	{
		roi.width = target_width;
		roi.x = 0;
```

```
                roi.height = height * scale;
                roi.y = (target_width - roi.height) / 2;
        }
        else
        {
                roi.y = 0;
                roi.height = target_width;
                roi.width = width * scale;
                roi.x = (target_width - roi.width) / 2;
        }

        resize(img, square(roi), roi.size());
        copyMakeBorder(square, square, 30, 30, 30, 30, BORDER_CONSTANT, img.at<Vec3b>(0,
0));

        return square;
}

void getFeaturePoints(String image, vector<float> &featurePoints, bool training) {
        Moments mom;
        double hu[7];
        Mat im_src = imread(image);
        Mat im_srcDisplay = getSquareImage(im_src);

        createWindowPartition(im_srcDisplay, largeWin, win, legend, noOfImagePerCol,
noOfImagePerRow);
        im_srcDisplay.copyTo(win[0]);
        putText(legend[0], "Source Image", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

        copyMakeBorder(im_src, im_src, 30, 30, 30, 30, BORDER_CONSTANT, im_src.at<Vec3b>(0,
0));
        // get Canny edge
        Mat im_gray;
        cvtColor(im_src, im_gray, CV_BGR2GRAY);

        Mat im_canny;
        GaussianBlur(im_gray, im_canny, Size(3, 3), 0, 0);
        Canny(im_canny, im_canny, 0, 150);
        dilate(im_canny, im_canny, Mat(), Point(-1, -1), 3);

        // Floodfill from point (0, 0)
        Mat im_floodfill = im_canny.clone();
        floodFill(im_floodfill, Point(0, 0), Scalar(255));

        // Invert floodfilled image
        Mat im_floodfill_inv;
        bitwise_not(im_floodfill, im_floodfill_inv);

        // Combine the two images to get the foreground.
        Mat im_out = (im_canny | im_floodfill_inv);

        // Extract contour with largest contour
        vector<vector<Point> > largestContour(1);
        getLargestContour(im_out, largestContour);

        // Get bounding rect of largest contour,
        // this is the region containing the fish
        Rect boundRect;
        boundRect = boundingRect(largestContour[0]);
        boundRect = enlargeROI(im_out, boundRect, 10);

        // Crop the fish and store into ROI,
        // rotate the image to principal axis
        Mat ROI = Mat(im_out, boundRect);
        double orientation = getOrientation(largestContour[0], im_out, false);
        im_out = rotateMat(ROI, orientation);
```

```
	getLargestContour(im_out, largestContour);

	// Get bounding rect of largest contour,
	// this is the region containing the fish
	boundRect = boundingRect(largestContour[0]);
	boundRect = enlargeROI(im_out, boundRect, 10);

	// Crop the fish and store into ROI,
	// normalize image size to 500x500
	ROI = Mat(im_out, boundRect);
	ROI = getSquareImage(ROI);

	getLargestContour(ROI, largestContour);
	// Get polygonal approximation of contour
	vector<Point> contoursApprox;
	approxPolyDP(largestContour[0], contoursApprox, POLY_APPROX_EPSILON, true);

	// Convert ROI to color image for display purposes
	cvtColor(ROI, ROI, CV_GRAY2BGR);
	getOrientation(largestContour[0], ROI);

	// Draw the contours to for display purposes
	drawContours(ROI, largestContour, 0, Scalar(0, 0, 255), 2); // Draw the largest
contour using previously stored index.
	if (training) {
		ROI.copyTo(win[1]);
		putText(legend[1], "Training", Point(5, 11), 1, 1, Scalar(250, 250, 250),
1);
		imshow("Training", largeWin);
	}
	else {
		cout << image << endl;
		ROI.copyTo(win[1]);
		putText(legend[1], "Testing", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);
		imshow("Testing", largeWin);
	}

	// Calculate features:
	// hu moment 1, 2
	// contour perimeter
	// contour area
	mom = moments(contoursApprox, true);
	HuMoments(mom, hu);
	for (int i = 0; i < 2; i++) {
		cout << (float)hu[i] << ", ";
		featurePoints.push_back((float)hu[i]);
	}
	float cLength = (float)arcLength(contoursApprox, true);
	float cArea = (float)contourArea(largestContour[0]);
	cout << cLength << ", " << cArea << endl;
	featurePoints.push_back(cLength);
	featurePoints.push_back(cArea);
}

int main() {
	vector<vector<float> > trainingData;
	vector<int> labels;
	Ptr<SVM> svm = SVM::create();
	svm->setType(SVM::C_SVC);
	svm->setKernel(SVM::RBF);
	svm->setTermCriteria(TermCriteria(TermCriteria::MAX_ITER, 100, 1e-6));
	for (int folder = 0; folder < folderName.size(); folder++) {
		cout << "Features for " << folderName[folder] << ": " << endl;
		vector<String> images;
		glob("Inputs\\Fishes\\Training\\" + folderName[folder] + "\\*", images);
```

```
			for (int i = 0; i < images.size(); i++) {
				vector<float> featurePoints;
				getFeaturePoints(images[i], featurePoints, true);
				trainingData.push_back(featurePoints);
				labels.push_back(folder);
				waitKey();
			}
			cout << endl;
		}
		destroyWindow("Training");
		// Copy data from vector into Mat
		Mat trainingDataMat(trainingData.size(), trainingData[0].size(), CV_32F);
		for (size_t i = 0; i < trainingData.size(); i++)
			for (size_t j = 0; j < trainingData[i].size(); j++)
				trainingDataMat.at<float>(i, j) = trainingData[i][j];
		Mat labelsMat(labels);

		// Normalization of the data
		// Xnew = (X - mean) / sigma
		vector<double> mean, sigma;
		for (int i = 0; i < trainingDataMat.cols; i++) {  //take each of the features in
vector
			Scalar meanOut, sigmaOut;
			meanStdDev(trainingDataMat.col(i), meanOut, sigmaOut);  //get mean and std
deviation
			mean.push_back(meanOut[0]);
			sigma.push_back(sigmaOut[0]);
		}
		for (size_t i = 0; i < trainingData.size(); i++)
			for (size_t j = 0; j < trainingData[i].size(); j++)
				trainingDataMat.at<float>(i, j) = (trainingData[i][j] - mean[j]) /
sigma[j];

		cout << trainingDataMat << endl << endl;
		svm->train(trainingDataMat, ROW_SAMPLE, labelsMat);

		// Predict
		int correct = 0, total_tested = 0;
		for (int folder = 0; folder < folderName.size(); folder++) {
			cout << "Testing for " << folderName[folder] << ": " << endl;
			vector<String> images;
			glob("Inputs\\Fishes\\Testing\\" + folderName[folder] + "\\*", images);
			for (int i = 0; i < images.size(); i++) {
				vector<float> featurePoints;
				getFeaturePoints(images[i], featurePoints, false);
				Mat sampleMat(1, featurePoints.size(), CV_32F);
				for (int i = 0; i < featurePoints.size(); i++)
					sampleMat.at<float>(0, i) = (featurePoints[i] - mean[i]) /
sigma[i]; //normalization

				int response = svm->predict(sampleMat);
				total_tested++;
				if (folder == response) {
					correct++;
				}
				cout << "EXPECTING: " << folderName[folder] << ", GET: " <<
folderName[response] << ((folder == response) ? " CORRECT" : " WRONG") << endl << endl;
				waitKey();
			}
		}
		cout << "GOT " << correct << " OUT OF " << total_tested << " CORRECT. (" << 100 *
(correct / (total_tested*1.0)) << "%)" << endl;
		system("pause");
		return 0;

}
```