



Cairo University



PATTERNS PROJECT

[Document subtitle]



Delivered to:

Saifeleslam Abdelrahman Mahmoud	1180515
Ahmed Yasser Ali	1180518
Hazem Tarek Abdelhakam	1180509
Arwa Ibrahim Shamardal	1180519

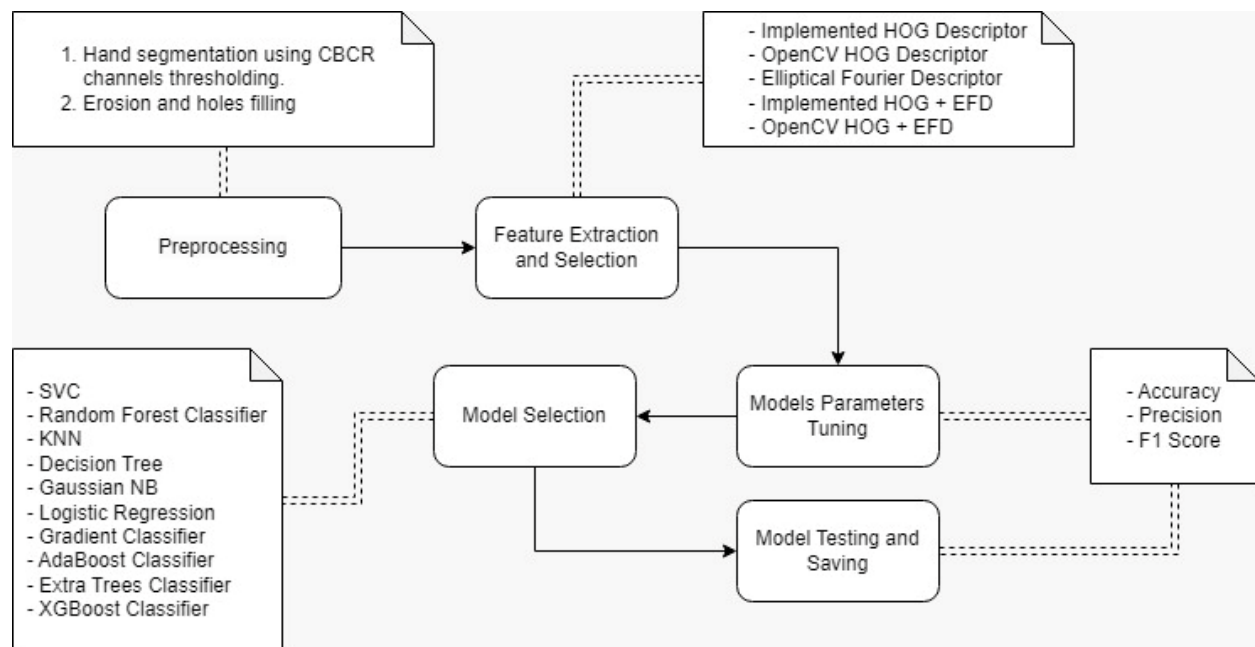
Table of Contents

Project Pipeline	2
Preprocessing Module.	2
a. Hand segmentation using CBCR.....	2
b. Erosion and holes filling	4
c. Image cropping	5
Feature Extraction/Selection Module.....	6
a. Implemented HOG Descriptor	7
1. Definition	7
2. Implementation	7
b. OpenCV HOG Descriptor	8
1. Definition	8
2. Implementation	8
c. Elliptical Fourier Descriptor.....	8
1. Definition	8
2. Implementation	8
d. Implemented HOG+EFD.....	9
1. Definition and Implementation	9
e. OpenCV HOG+EFD.....	9
1. Definition and Implementation	9
f. Results comparison	9
Model Selection	10
1. Models chosen for test	10
.11 Tuning parameters.....	11
Training Module.....	12
a. Dataset split	12
b. Training	12
c. Testing.....	12
Performance Analysis Module.	13
Enhancements and Future work.....	14

Project Pipeline

In order to classify a set of hand gesture signs with the best accuracy, we have gone through several steps.

In a nutshell, we started by preprocessing the input images using binary image processing and cropping technique. Then, we performed feature extraction using HOG and EFD feature extraction methods. After that, we chose a set of models to train the data on and we fit the data on all of them to tune and find the best parameters. Finally, we fit the data on all the models to get the best accuracy and save its model. As illustrated in the below chart, these were the borderline steps of the process and further details are discussed in below sections.



Preprocessing Module.

Preprocessing is an essential step in machine learning projects, and we used three techniques to preprocess our hand gesture dataset, namely hand segmentation using CBCR, erosion and hole filling, and image cropping. We performed these techniques in order to transform the raw data into a format that could be used for training our machine learning models. Following is a deeper look at each of the approaches we used.

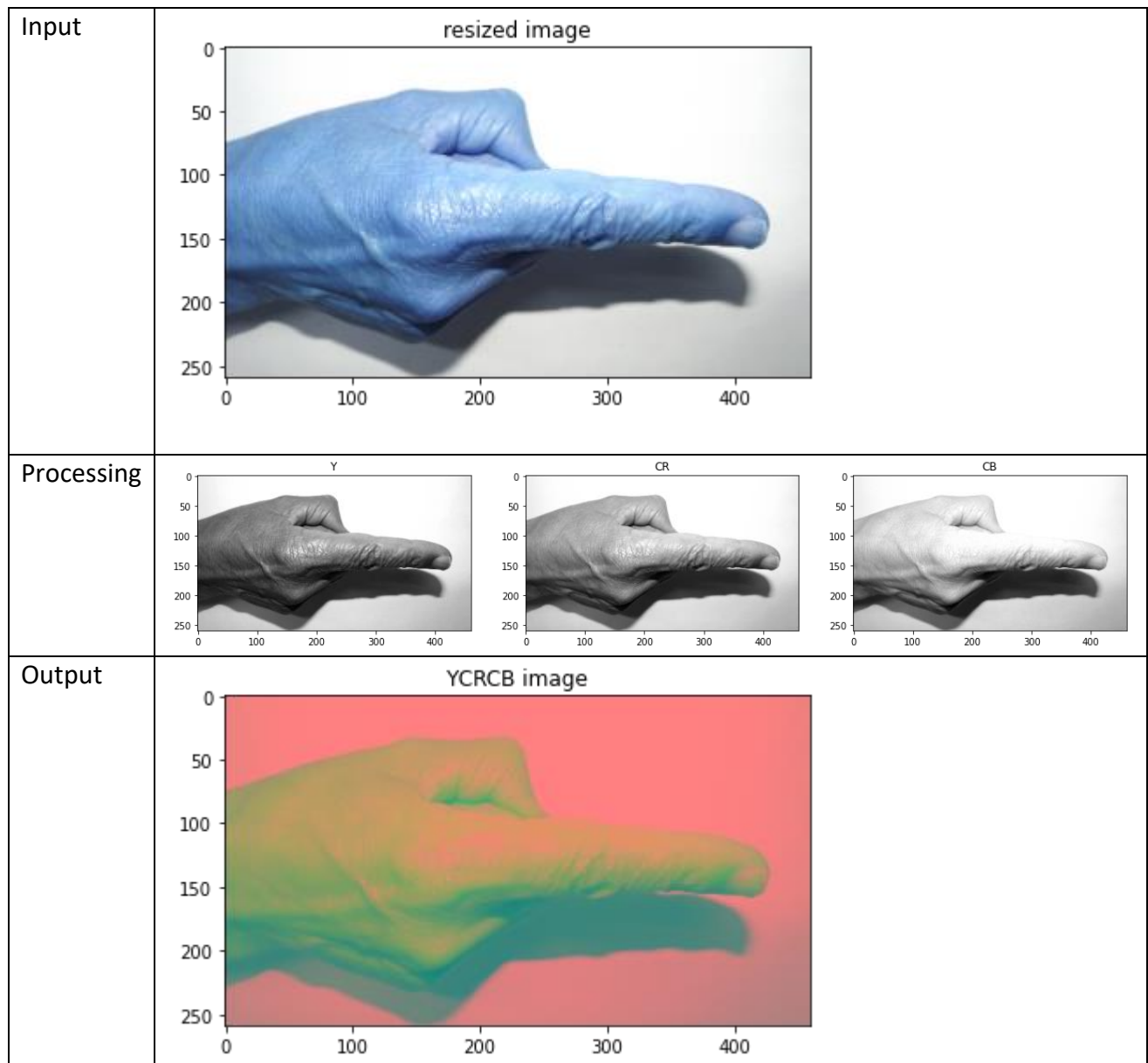
a. Hand segmentation using CBCR

The CBCR color space is an effective way to segment the hand region from the background in hand gesture recognition systems. The steps involved in this process are as follows:

- Convert the image from the RGB color space to the YCBCR color space, which separates the luminance and chrominance components of the image.

- Extract the CBCR channels from the YCBCR image, which represent the chrominance information.
- Apply a skin color model to the CBCR channels to create a binary mask that highlights the skin pixels. A common method for creating a skin color model is by computing the mean and standard deviation of the skin pixels in a training set.
- Apply morphological operations such as dilation and erosion to the binary mask to remove noise and smooth the edges.
- Multiply the binary mask with the original image to extract the hand region.

An illustration of the input, the processing steps and the output of the CBCR stage is shown below. This output is to be passed to the next stage of preprocessing.

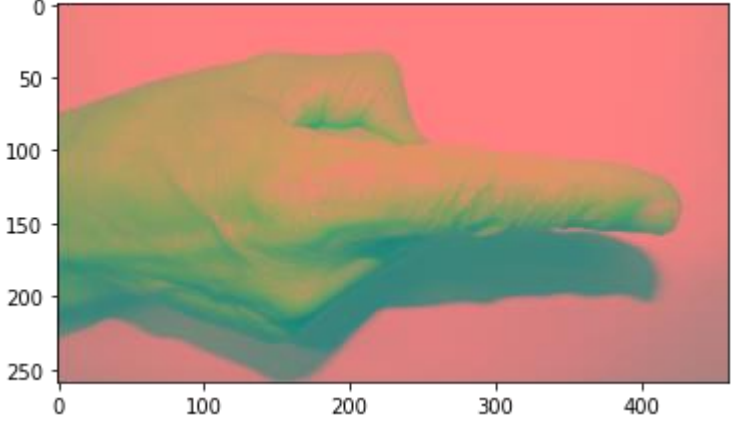
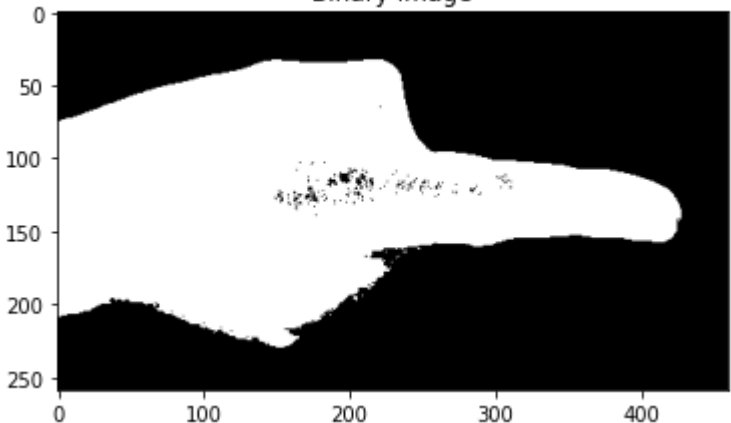


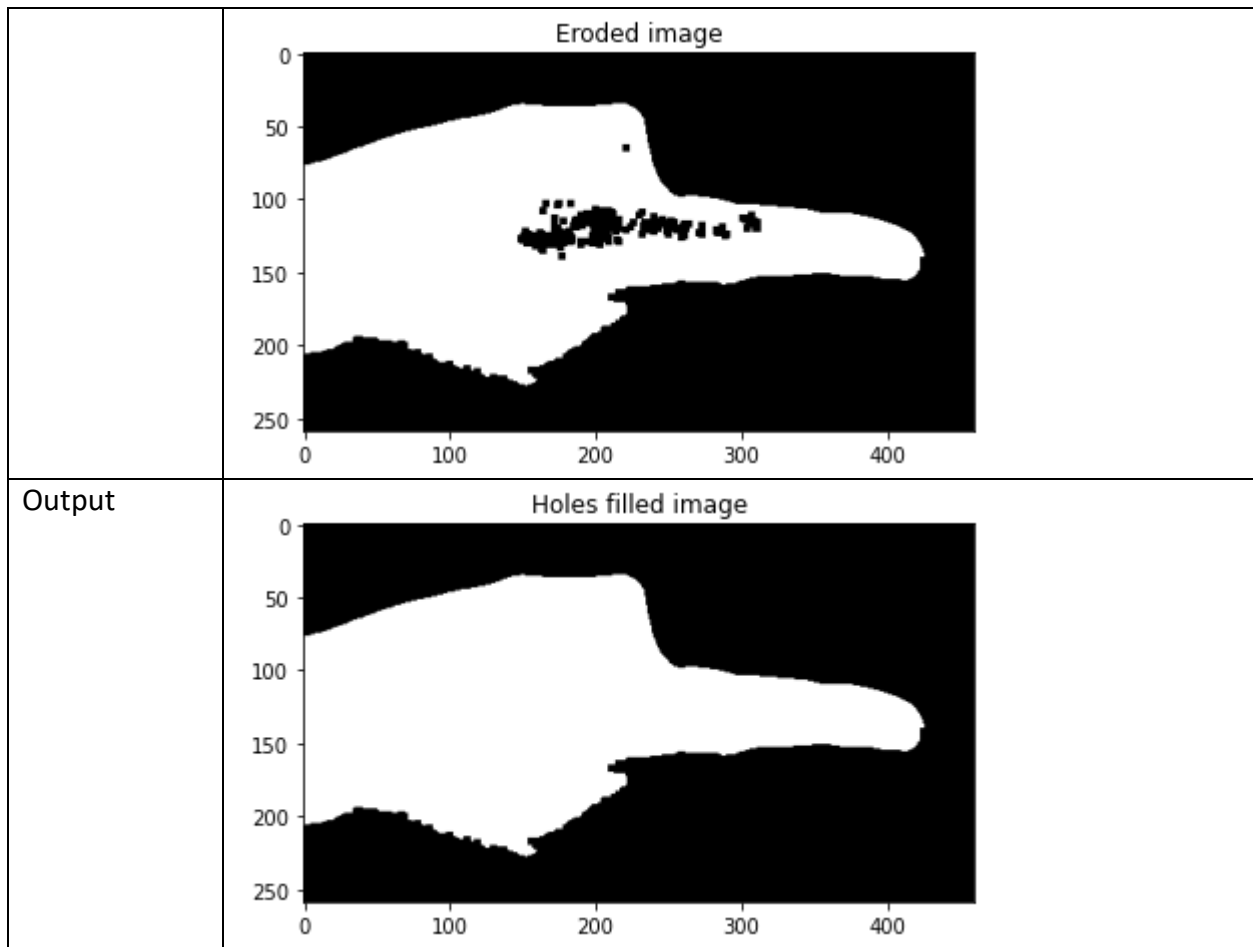
b. Erosion and holes filling

Erosion is a morphological operation that removes small objects and reduces the size of larger objects in an image. After showing the images and analyzing them, we ended up with performing the following steps:

- Convert the hand region to a binary image.
- Apply erosion with a small kernel to remove small objects and artifacts. We have chosen a kernel size of 5x5 square.
- Apply hole filling to fill any holes that may have been created by the erosion operation.

The below table shows the process followed to yield a binary image with no holes. As shown that the input is the YCRCB images produced in the previous stage and the output is a binary image with the hand in white (1,1,1) and the surrounding in black (0,0,00). This is stage is to help us determine the target object or area and discard irrelevant information in the image.

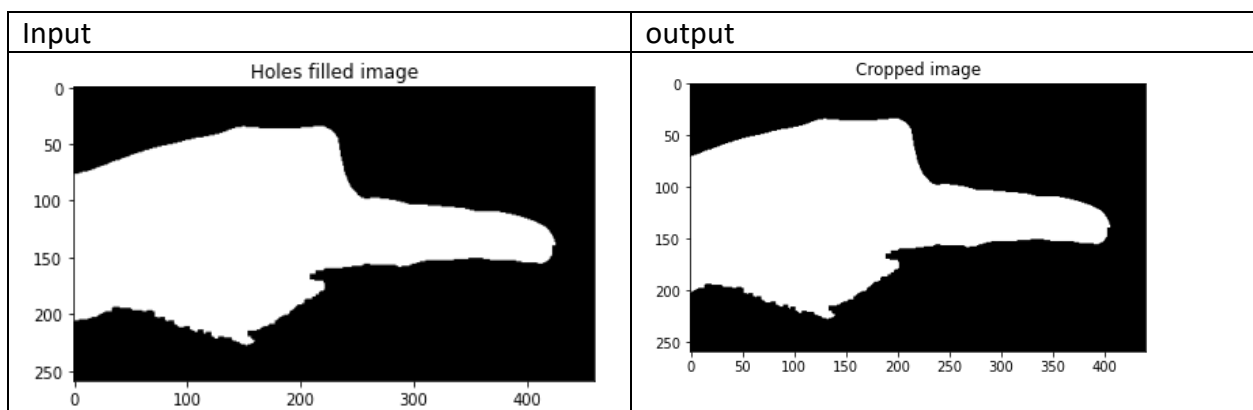
Input	 <p>A YCRCB image showing a hand. The hand is in the center, with a greenish-yellow color. The background is a reddish-pink color. The image is labeled 'YCRCB image' at the top. The x-axis ranges from 0 to 400, and the y-axis ranges from 0 to 250.</p>
Processing	 <p>A binary image showing the hand as a white shape on a black background. The image is labeled 'Binary image' at the top. The x-axis ranges from 0 to 400, and the y-axis ranges from 0 to 250.</p>



c. Image cropping

A further processing that we thought about was to crop the image so that we reduce the irrelevant information or area in the image. As this might be an unwanted step in some models, we performed this step, testing it on the models and then decided whether to keep it or not.

The following illustration shows that the input to the cropping stage is an image with size (259,460) and the output is an image with size (249,440)



As we mentioned that, we needed to test the model before and after cropping the images to see how would that affect the accuracy. Please note that, an explanation of each of the following models and their approaches are explained in below sections and these results are just to support our evidence for now.

The following graph shows the accuracy of the models after performing two feature extraction methods; customized HOG and HOG+EFD. It is obvious that the accuracies of the models have dropped when using the cropped images. So, we decided to exclude this stage as it is not supportive with the feature extraction and models we selected.

	Before					After				
HOG		Accuracy	Precision	Recall	F1-score		Accuracy	Precision	Recall	F1-score
	SVC	0.742466	0.744294	0.742466	0.741560	SVC	0.720548	0.717885	0.720548	0.718738
	RandomForestClassifier	0.484932	0.497067	0.484932	0.485925	RandomForestClassifier	0.402740	0.410316	0.402740	0.404146
	KNeighborsClassifier	0.657534	0.643243	0.657534	0.647723	KNeighborsClassifier	0.701370	0.698422	0.701370	0.696403
	DecisionTreeClassifier	0.326027	0.308028	0.326027	0.270413	DecisionTreeClassifier	0.408219	0.412439	0.408219	0.379269
	GaussianNB	0.613699	0.627070	0.613699	0.618532	GaussianNB	0.608219	0.613884	0.608219	0.608003
	LogisticRegression	0.756164	0.759463	0.756164	0.756626	LogisticRegression	0.712329	0.712516	0.712329	0.711387
	GradientBoostingClassifier	0.676712	0.689317	0.676712	0.680584	GradientBoostingClassifier	0.630137	0.652730	0.630137	0.639238
	AdaBoostClassifier	0.567123	0.618068	0.567123	0.582957	AdaBoostClassifier	0.479452	0.517740	0.479452	0.491194
	ExtraTreesClassifier	0.468493	0.466713	0.468493	0.464698	ExtraTreesClassifier	0.493151	0.498649	0.493151	0.494673
	XGBClassifier	0.704110	0.713435	0.704110	0.705786	XGBClassifier	0.679452	0.693006	0.679452	0.685109
HOG +EFD		Accuracy	Precision	Recall	F1-score		Accuracy	Precision	Recall	F1-score
	SVC	0.739726	0.742912	0.739726	0.739474	SVC	0.734247	0.731913	0.734247	0.732457
	RandomForestClassifier	0.517808	0.518990	0.517808	0.517272	RandomForestClassifier	0.430137	0.437212	0.430137	0.430222
	KNeighborsClassifier	0.652055	0.637043	0.652055	0.642112	KNeighborsClassifier	0.701370	0.699390	0.701370	0.696421
	DecisionTreeClassifier	0.454795	0.466065	0.454795	0.419833	DecisionTreeClassifier	0.504110	0.562918	0.504110	0.510682
	GaussianNB	0.610959	0.624981	0.610959	0.615926	GaussianNB	0.608219	0.613884	0.608219	0.608003
	LogisticRegression	0.747945	0.750627	0.747945	0.747983	LogisticRegression	0.717808	0.715877	0.717808	0.715925
	GradientBoostingClassifier	0.750685	0.759338	0.750685	0.753736	GradientBoostingClassifier	0.720548	0.723128	0.720548	0.721318
	AdaBoostClassifier	0.526027	0.560175	0.526027	0.537212	AdaBoostClassifier	0.526027	0.574538	0.526027	0.542731
	ExtraTreesClassifier	0.517808	0.514448	0.517808	0.513447	ExtraTreesClassifier	0.438356	0.446436	0.438356	0.440120
	XGBClassifier	0.797260	0.799781	0.797260	0.797539	XGBClassifier	0.739726	0.734463	0.739726	0.736749

Feature Extraction/Selection Module.

The Feature Extraction/Selection module provides various options for extracting and selecting features from images for object detection and recognition tasks. The HOG and EFD descriptors can be used separately or combined for more robust feature representations, and the OpenCV implementations provide optimizations for faster computation and training of classifiers.

We tried every option of the following ones, trained them and tested them to get the performance of the best model. Following, we would describe the 5 approaches we tried to select the best feature extractor.

a. Implemented HOG Descriptor

1. Definition

The Histogram of Oriented Gradients (HOG) is a feature descriptor used in computer vision and image processing for object detection. It works by calculating the distribution of gradients in an image. The HOG Descriptor implemented in this module computes the gradient orientation and magnitude for each pixel in the input image, divides the image into small blocks, and computes the histogram of the gradient orientations within each block. The histograms are then normalized to reduce the effect of illumination variations.

2. Implementation

The parameters used for this implementation are the same as the default of the openCV implementation:

HOG_WIDHT = 128	HOG_HEIGHT = 64	HOG_CELL_SIZE = 8
HOG_BLOCK_STRIDE = 8	HOG_BLOCK_SIZE = 16	HOG_BIN_COUNT = 9

Step 1: Preprocess the Data:

- 1.1- Grayscale: image is converted to grayscale in order to reduce the complexity of the image
- 1.2- Resize: image is resized to 128x64 pixels as hog must work on a fixed size image, it has a ration of 2:1 or 1:2 (recommended ratio), and this size is the most common size used (used in the openCV implementation)

Step 2: Calculate the Gradient:

- 2.1- Calculate the gradient in x and y directions using Sobel operator
- 2.2- Calculate the magnitude and direction of the gradient using the x and y gradients by changing the cartesian coordinates to polar coordinates

Step 3: Calculate the Histogram of the Gradient:

- 3.1- Divide the image into blocks of 8x8 pixels (cells)
- 3.2- Calculate the histogram for each cell

in this step, each pixel's angle coordinates were in the range of 0 to 2pi originally, but it's better to have them in the range of 0 to pi, so we divide them by 2 (to give better results by decreasing the step to 20 degrees instead since 180 degrees would be treated the same as 0 degrees)

- 3.3- Put the pixel result into one histogram (angle gives us the index, the magnitude is value put in the histogram)

Step 4: Normalize the n (4) cells of Histogram block:

- 4.1- Divide the image into blocks of 16x16 pixels (blocks)
- 4.2- Concatenate the histograms of the blocks into one histogram
- 4.3- Normalize the histogram of the block by dividing it by the sum of the histograms of the 4 cells of the block
- 4.4- Concatenate the histograms of the blocks into one histogram

finally we have a histogram of 3780 values for each image. That is because with these default settings mentioned above: for a 64x128 image, the HOG descriptor would divide the image into 16x32 cells, resulting in $7 \times 15 = 105$ blocks. Each block contains 4 cells, so there would be a total of $105 \times 4 = 420$ cells. The histogram in each cell contains 9 bins, so the final feature vector would be $420 \times 9 = 3780$ features.

b. OpenCV HOG Descriptor

1. Definition

OpenCV is an open-source computer vision library that provides various algorithms for image and video processing. The HOG Descriptor implementation in OpenCV uses the same basic idea as the implemented HOG Descriptor, but with some optimizations such as the use of integral images for faster computation. OpenCV also provides a function for training a Support Vector Machine (SVM) classifier on the extracted HOG features for object detection.

2. Implementation

The idea of the descriptor is explained above in details.

An object of class HOGDescriptor was created, images were passed to it one by one, and the 3780 features were computed.

c. Elliptical Fourier Descriptor

1. Definition

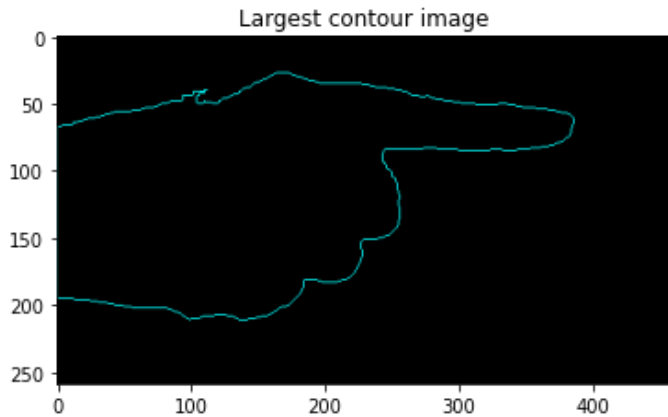
The Elliptical Fourier Descriptor (EFD) is a shape descriptor used to represent closed curves such as contours or boundaries of objects. It works by decomposing the contour into a set of ellipses and then extracting a set of coefficients that represent the shape of the contour. The EFD implemented in this module computes the Fourier descriptors for each point on the contour, and then scales and rotates the descriptors to be invariant to translation, rotation, and scale.

2. Implementation

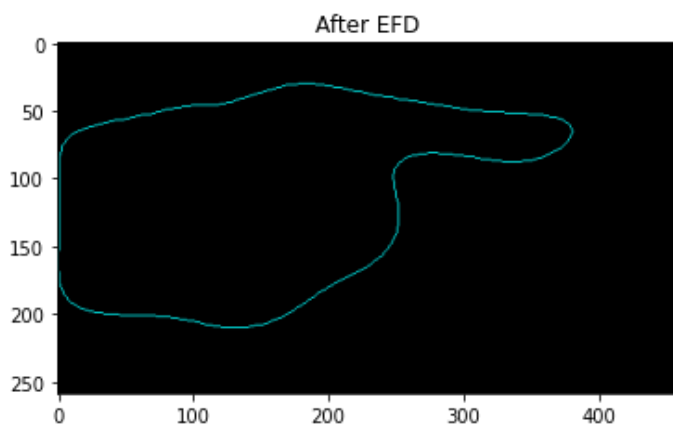
Hyperparameter used: *Order (number of coefficients) = 10*

To use Elliptical Fourier Descriptor,

1. The preprocessed binary image that contains the masked hand is used as an input to the function.
2. Find all contours of the image, and then get the contour that has maximum area, which is most probably be the hand itself.



3. Then, an EFD algorithm is run on the contoured image with different orders (coefficients).



4. Normalize these coefficients.
5. Return these coefficients as features for an image.

d. Implemented HOG+EFD

1. Definition and Implementation

The HOG and EFD descriptors can be combined to provide a more robust feature representation for object detection. The combined descriptor implemented in this module extracts both the HOG features and the EFD coefficients from an input image, concatenates them into a single feature vector, and normalizes the vector to reduce the effect of illumination variations.

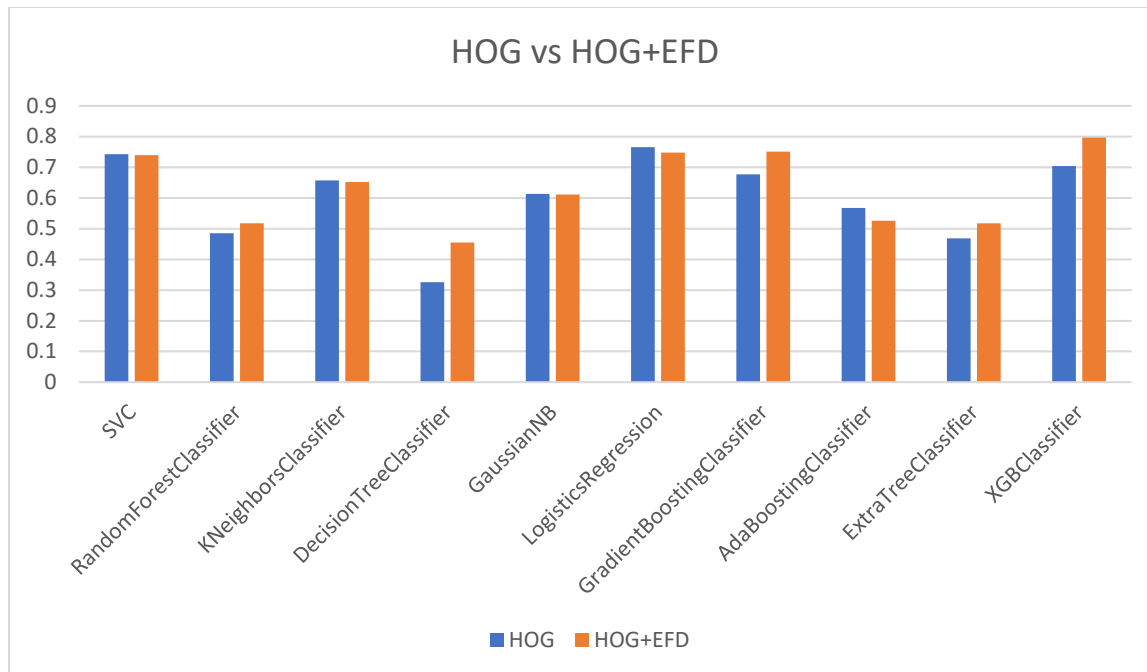
e. OpenCV HOG+EFD

1. Definition and Implementation

The OpenCV implementation of the HOG+EFD descriptor is similar to the implemented version, but with the added optimizations of the OpenCV HOG Descriptor implementation. OpenCV also provides a function for training an SVM classifier on the combined HOG+EFD features for object detection

f. Results comparison

After performing the feature selection by both methods and feeding them to our models, we got the following accuracies for HOG and HOG+EFD implemented. It is clear that HOG+EFD results in a better accuracy and thus, this would be the final feature extraction method chosen.



Model Selection

At the stage of model selection, we need to evaluate the performance of different models on the hand gesture classification dataset and choose the best performing model. To do this, we need to split the dataset into training and testing sets, fit each model on the training set, and evaluate its performance on the testing set using appropriate evaluation metrics such as accuracy, precision, recall, and F1 score.

1. Models chosen for test

To choose the best performing model, we need to compare the performance of all the models we have chosen and select the one that achieves the highest performance on the testing set.

Each model has its own strengths and weaknesses, and some models may be more suitable for certain types of data and tasks than others. Therefore, it is important to try different models to see which ones perform best on the hand gesture classification dataset.

1. **SVC:** Support Vector Machines are often used in classification tasks, and they work well when there is a clear margin of separation between classes. They can also work well on small datasets.
2. **Random Forest Classifier:** This is an ensemble method that combines multiple decision trees to improve performance and reduce overfitting. They work well on both small and large datasets and can handle both categorical and numerical data.
3. **K Neighbors Classifier:** This is a simple algorithm that works well for small datasets and can handle both classification and regression tasks. It is a non-parametric method that does not make any assumptions about the distribution of the data.

4. **Decision Tree Classifier:** This is a simple and interpretable model that works well on both small and large datasets. However, decision trees can suffer from overfitting and may not generalize well to new data.
5. **Gaussian NB:** This is a probabilistic model that works well on high-dimensional datasets with many features. It assumes that the features are independent, which may not be true in some cases.
6. **Logistic Regression:** This is a widely used model for binary classification tasks. It works well when the data is linearly separable and can handle both categorical and numerical data.
7. **Gradient Boosting Classifier:** This is another ensemble method that combines multiple weak learners to improve performance. It works well on both small and large datasets and can handle both categorical and numerical data.
8. **Ada Boosting Classifier:** This is also an ensemble method that combines multiple weak learners, but it focuses on misclassified samples and adjusts the weights of the training instances accordingly. It works well on both small and large datasets and can handle both categorical and numerical data.
9. **Extra Tree Classifier:** This is another ensemble method that combines multiple decision trees to improve performance. It works well on both small and large datasets and can handle both categorical and numerical data.
10. **XGB Classifier:** This is a powerful gradient boosting algorithm that uses decision trees as weak learners. It is known for its high performance and efficiency and works well on both small and large datasets.

11. Tuning parameters

In this stage, we tune the models in order to get the best performance with the models. The tuned parameters are shown in the below table and would be used in the training process.

Model	Parameters
SVC	Best parameters: {'C': 0.1, 'gamma': 0.1, 'kernel': 'poly'} Best score: 0.76279141577891 Validation accuracy: 1.0 Test accuracy: 0.791970802919708
Random Forest Classifier	Best parameters: {'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100} Best score: 0.7384529874942103 Validation accuracy: 1.0 Test accuracy: 0.7372262773722628
K Neighbors Classifier	Best parameters: {'n_neighbors': 1} Best score: 0.6716720704029644 Validation accuracy: 1.0 Test accuracy: 0.718978102189781
Decision Tree Classifier	Best parameters: {'criterion': 'gini', 'max_depth': 9, 'min_samples_leaf': 1}

	Best score: 0.49491122433225254 Validation accuracy: 0.6605839416058394 Test accuracy: 0.5182481751824818
Gaussian NB	Validation accuracy: 0.8613138686131386 Test accuracy: 0.5985401459854015
Logistics Regression	Best parameters: {'C': 0.01, 'max_iter': 100, 'penalty': 'l2'} Best score: 0.7282322062683342 Validation accuracy: 1.0 Test accuracy: 0.7153284671532847
Gradient Boosting Classifier	NOT TUNED as it took too long to process. n_estimators=100, learning_rate=.1, max_depth=3, random_state=0
Ada Boosting Classifier	Best parameters: {'learning_rate': 0.01, 'n_estimators': 1000} Best score: 0.5240296433534043 Validation accuracy: 0.5620437956204379 Test accuracy: 0.4927007299270073
Extra Tree Classifier	Best parameters: {'criterion': 'entropy', 'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200} Best score: 0.766709896557048 Validation accuracy: 1.0 Test accuracy: 0.781021897810219
XGB Classifier	NOT TUNED as it took too long to process. n_estimators=1000, learning_rate=0.008, max_depth=3, random_state=0

Training Module

a. Dataset split

The first step is to split the dataset into training, validation, and testing sets. In this case, we have split the dataset into 70% for training, 15% for validation, and 15% for testing. This split helps to evaluate the performance of the model on unseen data and prevents overfitting.

b. Training

The next step is to train the 10 models on the training set. The training process involves feeding the model with the input features and corresponding labels and optimizing the model's parameters to minimize the loss function.

We repeated the process of training and testing once on the models without tuning parameters and another time after we tune the parameters.

c. Testing

The final step is to evaluate the performance of the trained models on the testing set. This is done by feeding the testing set into the trained model and comparing the predicted labels with the actual labels. The evaluation metrics used can include accuracy, precision, recall, and F1 score.

The scores and results of each of the modules are stated in the below section.

Performance Analysis Module.

After we decided to choose exclude the cropping stage from preprocessing as it shows lower performance, we tested the models again tuned parameters and the default parameters and the results was as following.

	Not Tuned					Tuned				
HOG(Built-in)		Accuracy	Precision	Recall	F1-score		Accuracy	Precision	Recall	F1-score
	SVC	0.784672	0.782047	0.784672	0.781080	SVC	0.739726	0.742912	0.739726	0.739474
	RandomForestClassifier	0.722628	0.718793	0.722628	0.719071	RandomForestClassifier	0.517808	0.518990	0.517808	0.517272
	KNeighborsClassifier	0.718978	0.719038	0.718978	0.716247	KNeighborsClassifier	0.652055	0.637043	0.652055	0.642112
	DecisionTreeClassifier	0.547445	0.567308	0.547445	0.543353	DecisionTreeClassifier	0.454795	0.466065	0.454795	0.419833
	GaussianNB	0.616788	0.621706	0.616788	0.613733	GaussianNB	0.610959	0.624981	0.610959	0.615926
	LogisticRegression	0.693431	0.689867	0.693431	0.685897	LogisticRegression	0.747945	0.750627	0.747945	0.747983
	GradientBoostingClassifier	0.686131	0.691098	0.686131	0.687708	GradientBoostingClassifier	0.750685	0.759338	0.750685	0.753736
	AdaBoostClassifier	0.503650	0.557024	0.503650	0.514149	AdaBoostClassifier	0.526027	0.560175	0.526027	0.537212
	ExtraTreesClassifier	0.755474	0.752905	0.755474	0.752412	ExtraTreesClassifier	0.517808	0.514448	0.517808	0.513447
	XGBClassifier	0.697080	0.695876	0.697080	0.694780	XGBClassifier	0.797260	0.799781	0.797260	0.797539
HOG(Custom)		Accuracy	Precision	Recall	F1-score		Accuracy	Precision	Recall	F1-score
	SVC	0.755474	0.750939	0.755474	0.752425	SVC	0.742466	0.744294	0.742466	0.741560
	RandomForestClassifier	0.737226	0.738754	0.737226	0.735802	RandomForestClassifier	0.484932	0.497067	0.484932	0.485925
	KNeighborsClassifier	0.682482	0.679584	0.682482	0.679781	KNeighborsClassifier	0.657534	0.643243	0.657534	0.647723
	DecisionTreeClassifier	0.434307	0.537486	0.434307	0.443593	DecisionTreeClassifier	0.326027	0.308028	0.326027	0.270413
	GaussianNB	0.554745	0.568706	0.554745	0.558418	GaussianNB	0.613699	0.627070	0.613699	0.618532
	LogisticRegression	0.686131	0.682492	0.686131	0.681399	LogisticRegression	0.756164	0.759463	0.756164	0.756626
	GradientBoostingClassifier	0.689781	0.690204	0.689781	0.689499	GradientBoostingClassifier	0.676712	0.689317	0.676712	0.680584
	AdaBoostClassifier	0.500000	0.582660	0.500000	0.508989	AdaBoostClassifier	0.567123	0.618068	0.567123	0.582957
	ExtraTreesClassifier	0.744526	0.739189	0.744526	0.740100	ExtraTreesClassifier	0.468493	0.466713	0.468493	0.464698
	XGBClassifier	0.682482	0.686724	0.682482	0.683499	XGBClassifier	0.704110	0.713435	0.704110	0.705786
EFD		Accuracy	Precision	Recall	F1-score		Accuracy	Precision	Recall	F1-score
	SVC	0.509589	0.502797	0.509589	0.490276	SVC	0.153285	0.023496	0.153285	0.040747
	RandomForestClassifier	0.515068	0.531022	0.515068	0.520426	RandomForestClassifier	0.755474	0.761806	0.755474	0.755400
	KNeighborsClassifier	0.698630	0.707069	0.698630	0.696845	KNeighborsClassifier	0.762774	0.764888	0.762774	0.761756
	DecisionTreeClassifier	0.454795	0.492417	0.454795	0.443449	DecisionTreeClassifier	0.576642	0.588046	0.576642	0.576561
	GaussianNB	0.567123	0.544293	0.567123	0.550859	GaussianNB	0.551095	0.538436	0.551095	0.542082
	LogisticRegression	0.512329	0.488240	0.512329	0.489266	LogisticRegression	0.259124	0.100127	0.259124	0.138522
	GradientBoostingClassifier	0.723288	0.723513	0.723288	0.723188	GradientBoostingClassifier	0.733577	0.738611	0.733577	0.735032
	AdaBoostClassifier	0.556164	0.598774	0.556164	0.569093	AdaBoostClassifier	0.525547	0.547131	0.525547	0.530196
	ExtraTreesClassifier	0.495890	0.498294	0.495890	0.496624	ExtraTreesClassifier	0.781022	0.781002	0.781022	0.780016
	XGBClassifier	0.728767	0.729509	0.728767	0.728298	XGBClassifier	0.722628	0.725245	0.722628	0.722561

HOG+EFD built-in		Accuracy	Precision	Recall	F1-score			Accuracy	Precision	Recall	F1-score	
	SVC	0.720548	0.715483	0.720548	0.716700			SVC	0.813869	0.810260	0.813869	0.810785
	RandomForestClassifier	0.515068	0.516330	0.515068	0.515427			RandomForestClassifier	0.755474	0.747835	0.755474	0.749610
	KNeighborsClassifier	0.731507	0.720299	0.731507	0.721951			KNeighborsClassifier	0.722628	0.717597	0.722628	0.717603
	DecisionTreeClassifier	0.452055	0.462006	0.452055	0.423772			DecisionTreeClassifier	0.562044	0.560762	0.562044	0.558824
	GaussianNB	0.665753	0.675819	0.665753	0.663840			GaussianNB	0.678832	0.683719	0.678832	0.677485
	LogisticRegression	0.720548	0.723662	0.720548	0.720468			LogisticRegression	0.711679	0.703869	0.711679	0.705673
	GradientBoostingClassifier	0.750685	0.753114	0.750685	0.751579			GradientBoostingClassifier	0.740876	0.741928	0.740876	0.740831
	AdaBoostClassifier	0.536986	0.591654	0.536986	0.557001			AdaBoostClassifier	0.620438	0.660968	0.620438	0.634075
	ExtraTreesClassifier	0.545205	0.538941	0.545205	0.539878			ExtraTreesClassifier	0.770073	0.764379	0.770073	0.766370
	XGBClassifier	0.767123	0.769509	0.767123	0.768023			XGBClassifier	0.766423	0.768930	0.766423	0.767490
HOG+EFD custom		Accuracy	Precision	Recall	F1-score			Accuracy	Precision	Recall	F1-score	
	SVC	0.739726	0.742912	0.739726	0.739474			SVC	0.759124	0.754595	0.759124	0.756002
	RandomForestClassifier	0.517808	0.518990	0.517808	0.517272			RandomForestClassifier	0.755474	0.754871	0.755474	0.754210
	KNeighborsClassifier	0.652055	0.637043	0.652055	0.642112			KNeighborsClassifier	0.693431	0.688517	0.693431	0.689296
	DecisionTreeClassifier	0.454795	0.466065	0.454795	0.419833			DecisionTreeClassifier	0.525547	0.559419	0.525547	0.531193
	GaussianNB	0.610959	0.624981	0.610959	0.615926			GaussianNB	0.551095	0.566311	0.551095	0.555330
	LogisticRegression	0.747945	0.750627	0.747945	0.747983			LogisticRegression	0.686131	0.685550	0.686131	0.682179
	GradientBoostingClassifier	0.750685	0.759338	0.750685	0.753736			GradientBoostingClassifier	0.722628	0.733762	0.722628	0.726711
	AdaBoostClassifier	0.526027	0.560175	0.526027	0.537212			AdaBoostClassifier	0.594891	0.639165	0.594891	0.608472
	ExtraTreesClassifier	0.517808	0.514448	0.517808	0.513447			ExtraTreesClassifier	0.766423	0.762654	0.766423	0.762570
	XGBClassifier	0.797260	0.799781	0.797260	0.797539			XGBClassifier	0.751825	0.759254	0.751825	0.754098

Depending on the above table, we choose the SVC model with the HOF+EFD feature extraction. This model showed the best accuracy and runtime on the test dataset.

Enhancements and Future work.

- Dataset expansion as the dataset was partially small. The number of the total records was about 2000 and this is not a great amount for machine learning. So, adding more poses for the hand would eventually increase the accuracy of the models.
- Tuning the parameters of XGB and Gradient classifiers on faster machine. It took us more than a day of run and it didn't finish. Therefore, figuring out the tuned parameters for these two classifiers might result in a better accuracy.
- Testing against more models would also be a good approach.
- We wanted to implement deep learning algorithm to perform this task, yet this is beyond the scope of this course. So, trying it as a future work might result in better trained models.