MINISTRY OF EDUCATION AND TRAINING

**CAN THO UNIVERSITY**

**DEPARTMENT OF INFORMATION &**

**COMMUNICATION TECHNOLOGY**

****

**THESIS**

**INFORMATION TECHNOLOGY MAJORS**

**TRAINING A SHAPE PREDICTOR MODEL**

**WITH DLIB LIBRARY**

**Advisor: Student:**

**Dr. Pham The Phi Mai Phuoc Vinh**

**Student ID: B1805835**

**Course: 44**

Can Tho, May 2021

**ACKNOWLEDGMENT**

*This project would not have been possible without the support of many people. I would like to express my gratitude to my advisor Dr. Pham The Phi for his support and supervision in this project. His guidance helped me in all the time of research and writing this thesis.*

*I would also like to thanks to express my sincere thanks to the Lectures of Can Tho University, especially the teachers at the Department of Information & Communication Technology, who conveyed valuable knowledge in recent years.*

*Finally, I wish to express my special thanks to my family for their great encouragement throughout my project and thesis, as well as my friends for their continual support and encouragement.*

*Although there are many efforts in the process of writing the thesis, it cannot be avoided from errors. I look forward to receiving your valuable comments to make the thesis more complete.*

Can Tho, May 2021

Student

Mai Phuoc Vinh

**TABLE OF** **CONTENTS**

[INTRODUCTION 6](#_Toc73732829)

[1. Project purposes 6](#_Toc73732830)

[2. Project methodology and approach 6](#_Toc73732831)

[3. Project content 6](#_Toc73732832)

[CONTENTS 7](#_Toc73732833)

[CHAPTER 1: An overview of the definition of terms and libraries used 7](#_Toc73732834)

[1.1. Definition of terms in this project 7](#_Toc73732835)

[1.1.1. Machine Learning 7](#_Toc73732836)

[1.1.2. Computer Vision 7](#_Toc73732837)

[1.1.3. Object detection and Shape predictors 7](#_Toc73732838)

[1.1.4. Decision Tree 8](#_Toc73732839)

[1.1.5. Ensemble Learning 9](#_Toc73732840)

[1.1.6. Loss Function 9](#_Toc73732841)

[1.1.7. Boosting 10](#_Toc73732842)

[1.1.8. Gradient Descent 10](#_Toc73732843)

[1.1.9. Gradient Boosting Decision Tree (GBDT) 11](#_Toc73732844)

[1.2. Libraries 12](#_Toc73732845)

[1.2.1. Dlib library 12](#_Toc73732846)

[1.2.2. OpenCV library 12](#_Toc73732847)

[CHAPTER 2: Model design and implement 13](#_Toc73732848)

[2.1. Shape/landmark predictors training algorithm 13](#_Toc73732849)

[2.2. Prepare dataset for training 13](#_Toc73732850)

[2.2.1. iBUG 300-W dataset 13](#_Toc73732851)

[2.2.2. Dataset preprocessing 14](#_Toc73732852)

[2.3. Implement custom dlib shape predictor training script 16](#_Toc73732853)

[2.3.1. Set up training script 16](#_Toc73732854)

[2.3.2. Training result and Evaluation 19](#_Toc73732855)

[CHAPTER 3: Testing and reviewing 20](#_Toc73732856)

[3.1. Implement the shape predictor inference script 20](#_Toc73732857)

[3.2. Benchmark 21](#_Toc73732858)

[CONCLUSION 22](#_Toc73732859)

[1. Summary of Results 22](#_Toc73732860)

[2. Future Work 22](#_Toc73732861)

[REFERENCES 23](#_Toc73732862)

**LIST OF FIGURES**

[Figure 1: Example for object detection 8](#_Toc73732863)

[Figure 2: Facial landmark 8](#_Toc73732864)

[Figure 3: Outlook decision tree 9](#_Toc73732865)

[Figure 4: Ensemble learning model 9](#_Toc73732866)

[Figure 5: How boosting model work 10](#_Toc73732867)

[Figure 6: Visualize gradient descent 11](#_Toc73732868)

[Figure 7: Schematic diagram of a boosted ensemble of decision tree 11](#_Toc73732869)

[Figure 8: Dlib library 12](#_Toc73732870)

[Figure 9: OpenCV with Python 12](#_Toc73732871)

[Figure 10: Ensemble of regression trees 13](#_Toc73732872)

[Figure 11: iBUG 300-W face landmark dataset 13](#_Toc73732873)

[Figure 12: Various type of faces 14](#_Toc73732874)

[Figure 13: Visualizing the 68 facial landmark coordinates 14](#_Toc73732875)

[Figure 14: Dataset directory structure 15](#_Toc73732876)

[Figure 15: Testing dataset xml structure 15](#_Toc73732877)

[Figure 16: Dataset structure after process 16](#_Toc73732878)

[Figure 17: Training verbose 19](#_Toc73732879)

[Figure 18: Testing in good light 20](#_Toc73732880)

**LIST OF TABLES**

[Table 1: Commom loss function 10](#_Toc73732881)

[Table 2: Benchmark\* with other model 21](#_Toc73732882)

**ABSTRACT**

In recent years, information technology in the world is growing greatly, especially in the study of machine learning because of there wide applications in many fields of life. Along with machine learning is a branch of computer vision that helps computers learn to visualize objects in an image or detect objects in images that are similar to humans brain. In this project, we will give a specific example for training a model to predict object shape from a input data set that has been processed using dlib libraries, resulting in a model that can balance speed, accuracy, and model size. Specifically in this project we will train a shape predictors used to localize specific locations on the human face (i.e., facial landmarks) and then put our shape predictor to the test and apply it to set of input images/video streams, demonstrating that our shape predictor is capable of running in real-time.

Keywords: machine learning, computer vision, object detection and facial landmark.

**TÓM TẮT**

Trong những năm gần đây, công nghệ thông tin trên thế giới có sự phát triển rất vượt bậc đặc biệt là các vấn đề liên quan đến máy học bởi vì những ứng dụng rộng rãi của nó trong cuộc sống. Đi cùng với máy học là thị giác máy tính, phân nhánh này giúp cho máy tính có thể hình dung được các vật thể trong một bức ảnh hoặc nhận diện một bức ảnh theo góc nhìn của khoa học máy tính. Trong niên luận lần này, em sẽ nghiên cứu một ví dụ về huấn luyện mô hình nhận diện vật thể từ một tập dữ liệu cho trước bằng cách sử dụng các thư viện có sẵn của dlib, kết quả sẽ là một mô hình dự đoán với khả năng dự đoán nhanh, chính xác và độ lớn của mô hình được đảm bảo. Nói một cách chi tiết hơn thì trong niên luận, em sẽ huấn luyện một mô hình nhận diện các bộ phận trên mặt người với đầu vào là một hình ảnh hoặc một video trong thời gian thực.

Các từ khoá: máy học, thị giác máy tính, nhận dạng vật thể và nhận diện bộ phận trên gương mặt.

INTRODUCTION

1. Project purposes

The main purpose of this project is to build and train a machine learning model that helps identify facial features. Learn how to custom the model training parameters and learn the algorithms applied in the dlib libraries in use. From that training model, we can implement into some applications based on its.

1. Project methodology and approach

The method of applying the dlib library for face recognition and marking points around the parts of the face. Build the markup model based on reusing the dlib library and customizing some parameters to suit actual usage needs.

1. Project content

- **Introduction**: An overview of the thesis: an introduction to the topic, research methods and layout of the thesis.

- **Content part**: The content of the thesis is divided into 3 chapters

+ Chapter 1: An overview of the definition of terms and libraries used

+ Chapter 2: Model design and implement

+ Chapter 3: Test and review

- **Conclusion**: Present the results achieved and the development direction of the system

CONTENTS

1. An overview of the definition of terms and libraries used

1.1. Definition of terms in this project

1.1.1. Machine Learning

Machine learning is a branch of computer science that broadly aims to enable computers to “learn” without being directly programmed. It has origins in the artificial intelligence movement of the 1950s and emphasizes practical objectives and applications, particularly prediction and optimization. Computers “learn” in machine learning by improving their performance at tasks through “experience”. In practice, “experience” usually means fitting to data; hence, there is not a clear boundary between machine learning and statistical approaches. The accuracy of the machine learning model depends a lot on the quality of the training data set and the training algorithm. However, choosing a certain algorithm that is suitable for the problem is a problem that requires a lot of research and comparison to be able to choose which algorithm is reasonable for the problem.

1.1.2. Computer Vision

Computer vision is an area of computer science that helps computers "see and understand" the content of digital images such as images and movies. Computer vision performs simulations like human vision with successive stages: eye simulation (information acquisition), digital image processing, data analysis and extraction of the Real image converted to digitization of image information. Image recognition can be viewed as solving the problem of information symbols from image data using models built with the help of theoretical, statistical, and physical disciplines.

1.1.3. Object detection and Shape predictors

Object detection is a general term to describe a set of computer vision tasks that are related to the identification of objects in digital images. It involves classifying an image into different categories or naming features, there are now quite a few models and data available for classification that have been highly effective in practice. Although there are many challenges for image classification, there have been many studies and solutions to improve. Along with classification, finding the position of an object in an image (shape prediction), which is widely used, can solve many useful things in life. For example, machine intelligent cropping (knowing to crop an image based on the location of an object) can even perform normal extraction to increase the accuracy of other processing techniques.

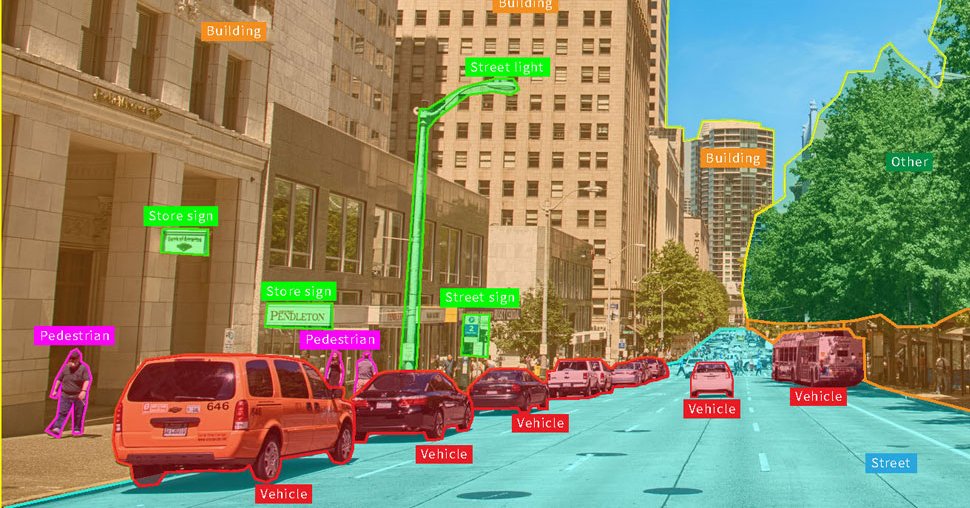


Figure 1: Example for object detection

Shape/landmark predictors are used to localize specific (x, y)-coordinates on an input “shape”. The term “shape” is arbitrary, but it’s assumed that the shape is structural in nature. Examples of structural shapes include: faces, hands, fingers, toes,...

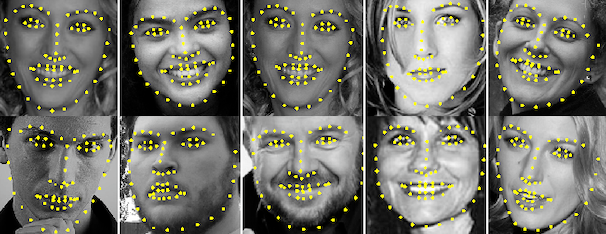


Figure 2: Facial landmark

1.1.4. Decision Tree

Decision Tree is a supervised machine learning algorithm. Decision Trees are those type of trees which groups attributes by sorting them based on their values. Decision tree is used mainly for classification purpose. Each tree consists of nodes and branches. Each nodes represents attributes in a group that is to be classified and each branch represents a value that the node can take. Decision Trees are applied to both types of problems: classification and regression.

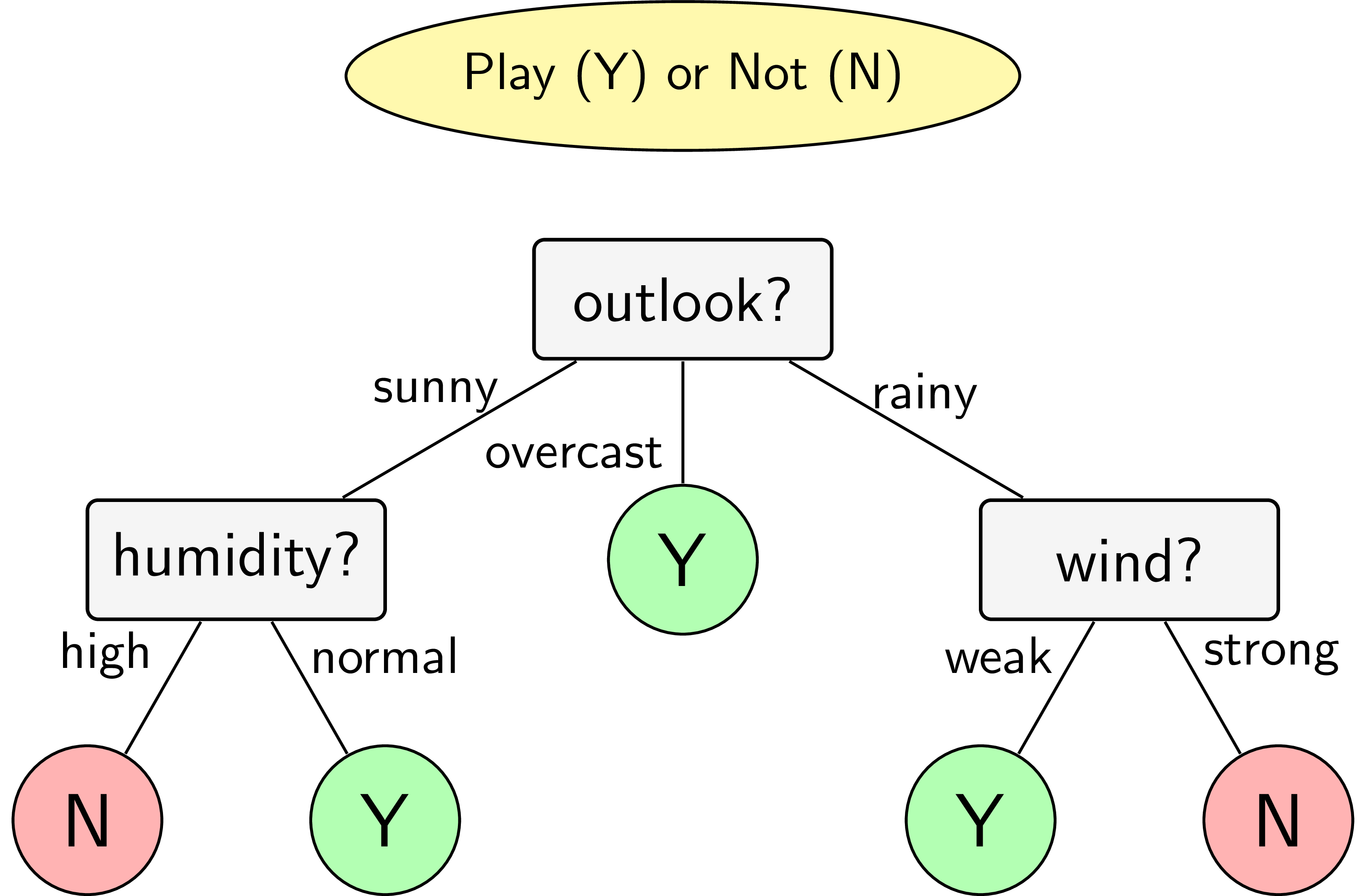


Figure 3: Outlook decision tree

1.1.5. Ensemble Learning

Ensemble learning is the process by which multiple models, such as classifiers or experts, are strategically generated and combined to solve a particular computational intelligence problem. Ensemble learning is primarily used to improve the (classification, prediction, function approximation, etc.) performance of a model, or reduce the likelihood of an unfortunate selection of a poor one. Other applications of ensemble learning include assigning a confidence to the decision made by the model, selecting optimal (or near optimal) features, data fusion, incremental learning, non stationary learning and error-correcting. This article focuses on classification related applications of ensemble learning, however, all principle ideas described below can be easily generalized to function approximation or prediction type problems as well.

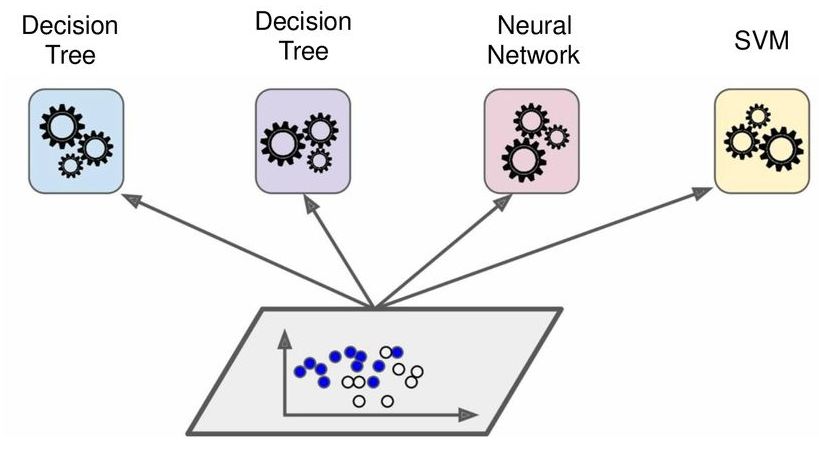


Figure 4: Ensemble learning model

1.1.6. Loss Function

The loss function is a function used to measure the loss and error of the predictive model, the function is responsible for reflecting the accuracy of the model. There are many loss functions in use today. The following table shows the most commonly used functions.

|  |  |
| --- | --- |
| **Loss function** |  |
| Squared Error Loss |  |
| Absolute Error Loss |  |
| Huber Loss |  |
| Hinge Loss |  |
| 0-1 Loss Funtion |  |

Table 1: Commom loss function

1.1.7. Boosting

Boosting is a technique in ensemble learning which is used to decrease bias and variance. Boosting creates a collection of weak learners and convert them to one strong learner. A weak learner is a classifier which is barely correlated with true classification. On the other hand, a strong learner is a type of classifier which is strongly correlated with true classification.

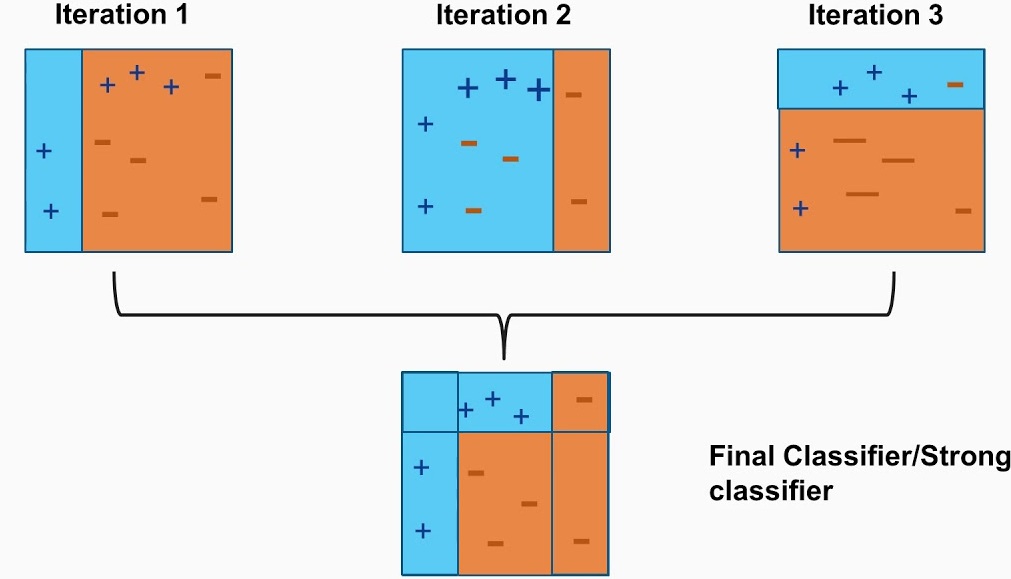


Figure 5: How boosting model work

1.1.8. Gradient Descent

In machine learning problems, we often have to find the minimum or maximum value for the error function. In general, finding the minimum point on a function can be found by solving the zero derivative equation, but for some very complex error functions (multidimensional data, many data points), the derivative to find the solution is an impossible thing. So one will try to iterate through the values to find the minimum point on the function, and to some extent we consider that to be the solution of the problem to be solved. The most common approach is to start from a point that we consider close to the solution of the problem, and then use an iterative operation to work toward the desired point, i.e., until the derivative is close to zero. In other words gradient descent is an optimization method based on the gradient of the function. This algorithm will update the parameters by going against the gradient until convergence.

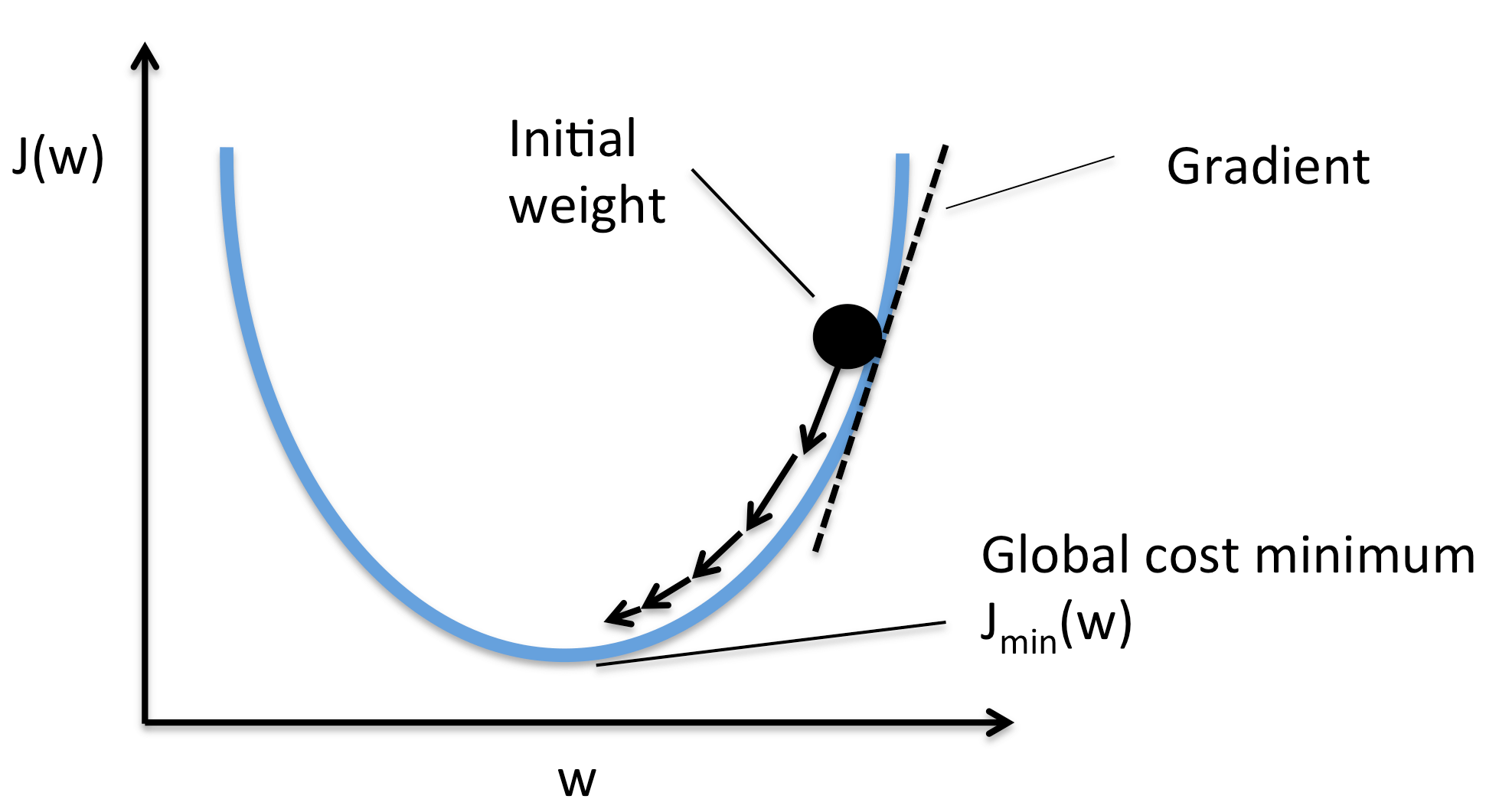


Figure 6: Visualize gradient descent

1.1.9. Gradient Boosting Decision Tree (GBDT)

Gradient boosting is an important method in ensemble learning. This method is based on the fact that we will use boosting to calculate gradient values on a series of elementary models then all the basic models in this learning series will combine by some math to give output the final result.

The gradient boosting decision tree is similar to the gradient boosting but the base model of its entire algorithm will be the decision tree.

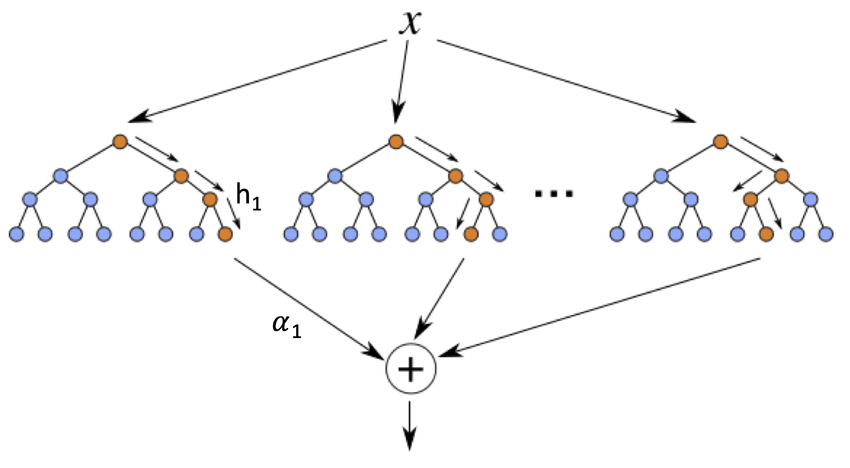


Figure 7: Schematic diagram of a boosted ensemble of decision tree

1.2. Libraries

1.2.1. Dlib library

Dlib is a cross platform open source software library written in the C++ programming language. Its design is heavily influenced by ideas from design by contract and component-based software engineering. This means it is first and foremost a collection of independent software components,each accompanied by extensive documentation and thorough debugging modes. Moreover, the library is intended to be useful in both research and real world commercial projects and has been carefully designed to make it easy to integrate into a user’s C++ application.

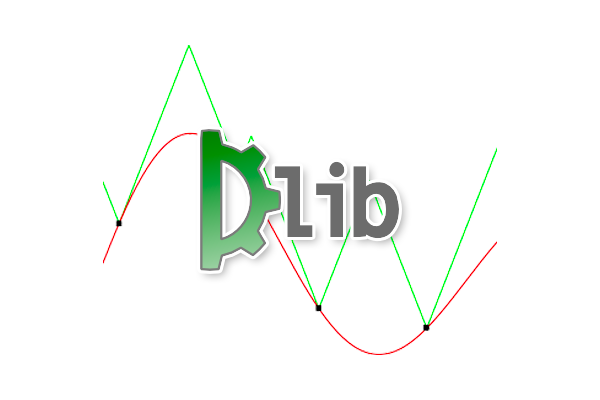


Figure 8: Dlib library

1.2.2. OpenCV library

OpenCV is an open source computer vision library. The library is written in C and C++ and runs under Linux, Window and Mac OS X. There is active development on interfaces for Python, Ruby, Matlab, and other languages. Open CV was designed for computational efficiency and with a strong focus on real-time applications. OpenCV is written in optimized C and can take advantage of multicore processors.

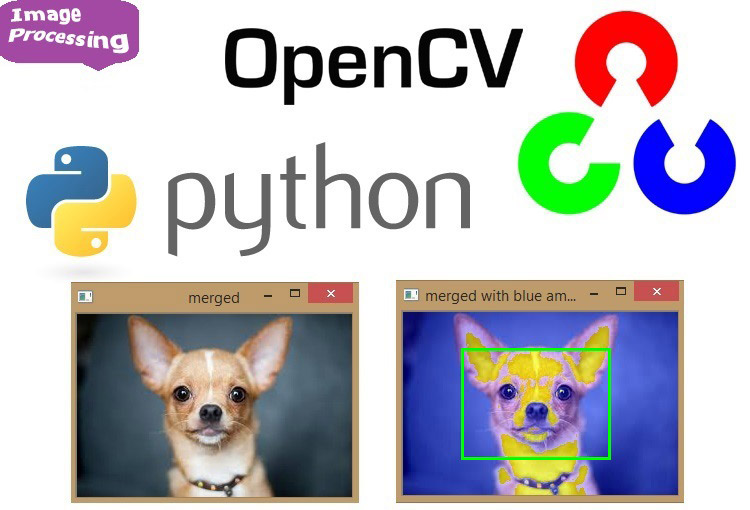


Figure 9: OpenCV with Python

1. Model design and implement

2.1. Shape/landmark predictors training algorithm

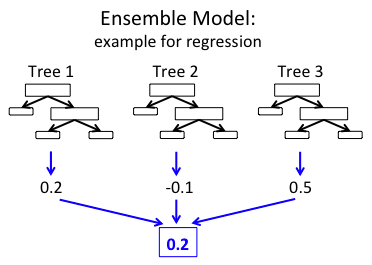


Figure 10: Ensemble of regression trees

The shape predictor algorithm implemented in the dlib library comes from Kazemi and Sullivan’s 2014 CVPR paper.

To estimate the landmark locations, the algorithm:

* Examines a sparse set of input pixel intensities (i.e., the “features” to the input model)
* Passes the features into an Ensemble of Regression Trees (ERT)
* Refines the predicted locations to improve accuracy through a cascade of regressors

2.2. Prepare dataset for training

2.2.1. iBUG 300-W dataset

To train our custom dlib shape predictor, we’ll be utilizing the [iBUG 300-W dataset](https://ibug.doc.ic.ac.uk/resources/300-W/).



Figure 11: iBUG 300-W face landmark dataset

The goal of iBUG-300W is to train a shape predictor capable of localizing each individual facial structure, including the eyes, eyebrows, nose, mouth, and jawline. It consists 68 pairs of integer values - these values are the (x, y) - coordinates of the facial structures on each training image.



Figure 12: Various type of faces

To create the iBUG-300W dataset, researchers manually and painstakingly annotated and labeled each of the 68 coordinates on a total of 7,764 images. A model trained on iBUG-300W can predict the location of each of these 68 (x, y)-coordinate pairs and can, therefore, localize each of the locations on the face.

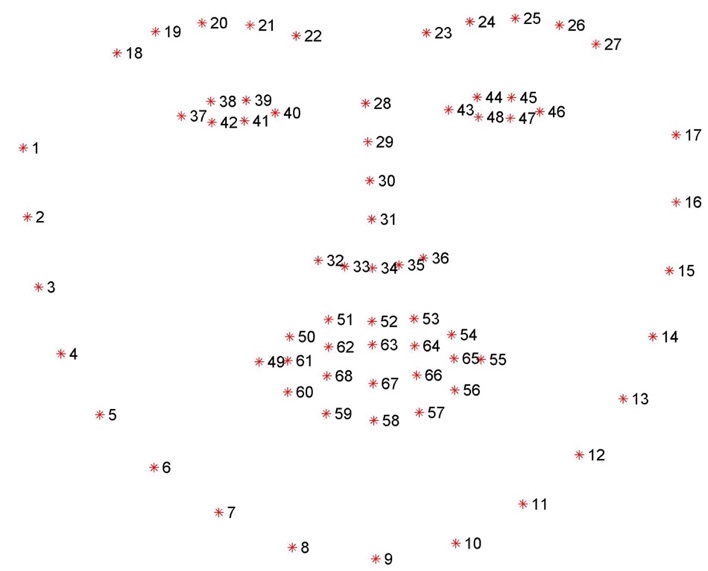


Figure 13: Visualizing the 68 facial landmark coordinates

2.2.2. Dataset preprocessing

In this project, we train a shape predictor capable of localizing eyes and mouth. So we need filter out some coordinate pairs to fit our model (just need eyes and mouth coordinate pairs). All we need is some basic file parsing to create a new training file that includes just the eyes and mouth coordinates.

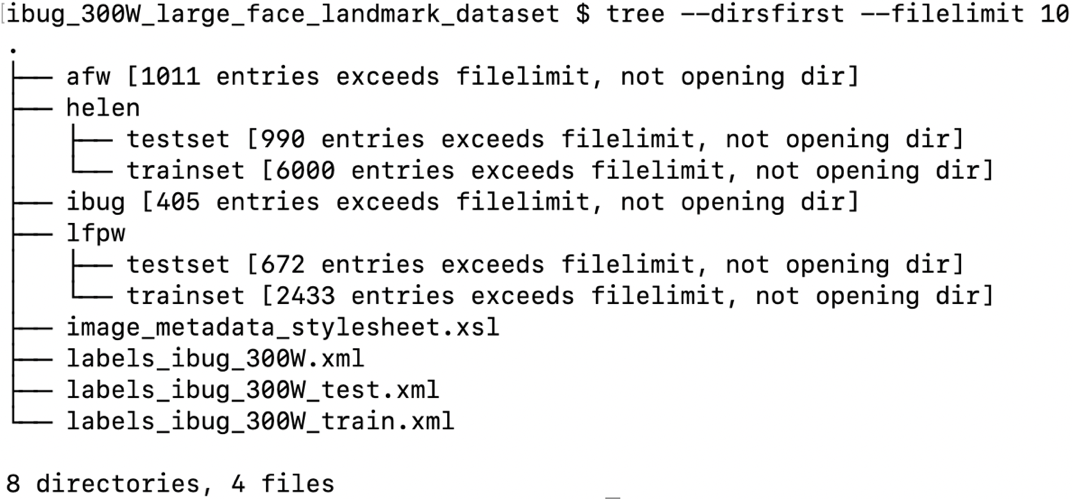


Figure 14: Dataset directory structure

All training data in the iBUG-300W dataset is represented by a structured XML file. Each image has an image tag. Inside the image tag is a file attribute that points to where the example image file resides on disk. Each image has a box element contains top (the starting y-coordinate of the bounding box), left (the starting x-coordinate of the bounding box), width (the width of the bounding box) and height (the height of the bounding box) attribute associated with it. Inside the box element we have a total of 68 part elements - these part elements represent the individual (x, y) coordinates and name of the facial landmarks in the iBUG-300W dataset.

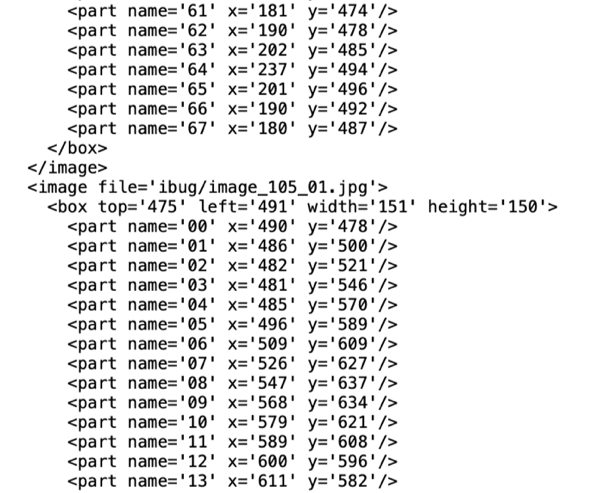


Figure 15: Testing dataset xml structure

These landmarks map to specific facial structures show in Figure 12 above and based on the visualization, we can then derive which name coordinates maps to which facial structure:

- The mouth can be accessed through points [48, 68].

- The right eyebrow through points [17, 22].

- The left eyebrow through points [22, 27].

- The right eye using [36, 42].

- The left eye with [42, 48].

- The nose using [27, 35].

- And the jaw via [0, 17].

**The idea of filtering out the points we need to use as follow:**

- The selection point will be in the <part> tag with name in range 36 to 68, so when we read each line of xml file of the iBUG 300-W dataset, we just check the condition if the <part> name in range we need, we keep it, otherwise delete it. Filtering out will be work easier with the help of xml regex file processing library.

New training file by parsing only the eye and mouth landmark coordinates from the original training and testing file like figure below:

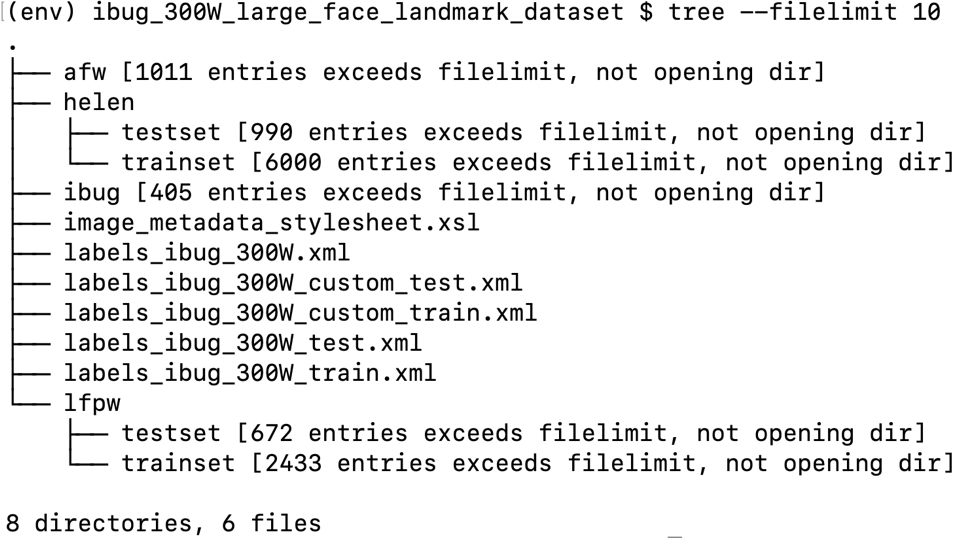


Figure 16: Dataset structure after process

2.3. Implement custom dlib shape predictor training script

2.3.1. Set up training script

Dlib shape predictor training script focus on 7 most important hyperparameters you can tune/set when training a custom dlib shape predictor. These values are:

- tree\_depth

- nu

- cascade\_depth

- feature\_pool\_size

- num\_test\_splits

- oversampling\_amount

- oversampling\_translation\_jitter

Begin with grabbing the default dlib shape predictor options:

options = dlib.shape\_predictor\_training\_options()

Then configure the tree\_depth option:

options.tree\_depth = 4

Define the tree\_depth, which, as the name suggests, controls the depth of each regression tree in the Ensemble of Regression Trees (ERTs). There will be 2^tree\_depth leaves in each tree. Smaller values of tree\_depth will lead to more shallow trees that are faster, but potentially less accurate. Larger values of tree\_depth will create deeper trees that are slower, but potentially more accurate. Typical values for tree\_depth are in the range [2, 8].

The next parameter is nu, a regularization parameter:

options.nu = 0.1

The nu option is a floating-point value (in the range [0, 1]) used as a regularization parameter that help model generalize. Values closer to 1 will make model fit the training data closer, but could potentially lead to overfitting. Values closer to 0 will help model generalize; however, there is a caveat to the generalization power - the closer nu is to 0, the more training data need.

Next parameter is the cascade\_depth:

options.cascade\_depth = 15

A series of cascades is used to refine and tune the initial predictions from the ERTs -the cascade\_depth will have a dramatic impact on both the accuracy and the output file size of your model. The more cascades allow for, the larger model will become (but potentially more accurate). The fewer cascades allow, the smaller model will be (but could be less accurate). The following figure from Kazemi and Sullivan’s paper demonstrates the impact that the cascade\_depth has on facial landmark alignment:

Typically you’ll want to explore cascade\_depth values in the range [6, 18], depending on your required target model size and accuracy.

Move on to the feature\_pool\_size:

options.feature\_pool\_size = 400

The feature\_pool\_size controls the number of pixels used to generate features for the random trees in each cascade. The more pixels include, the slower model will run (but could potentially be more accurate). The fewer pixels take into account, the faster model will run (but could also be less accurate).

Next parameter is the num\_test\_splits:

options.num\_test\_splits = 50

The num\_test\_splits parameter has a dramatic impact on how long it takes model to train (i.e., training/wall clock time, not inference speed). The more num\_test\_splits consider, the more accurate shape predictor - but again, be cautious with this parameter as it can cause training time to explode.

oversampling\_amount parameter:

options.oversampling\_amount = 5

The oversampling\_amount controls the amount of [data augmentation](https://pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/) applied to training data. The dlib library causes data augmentation jitter, but it is essentially the same idea as data augmentation. Here we are telling dlib to apply a total of 5 random deformations to each input image. You can think of the oversampling\_amount as a regularization parameter as it may lower training accuracy but increase testing accuracy, thereby allowing our model to generalize better. Typical oversampling\_amount values lie in the range [0, 50] where 0 means no augmentation and 50 is a 50x increase in training dataset.

Next comes the oversampling\_translation\_jitter option:

options.oversampling\_translation\_jitter = 0.1

The oversampling\_translation\_jitter controls the amount of translation augmentation applied to training dataset. Typical values for translation jitter lie in the range [0, 0.5].

The be\_verbose option simply instructs dlib to print out status messages as our shape predictor is training:

options.be\_verbose = True

Finally, we have the num\_threads parameter:

options.num\_threads = multiprocessing.cpu\_count()

This parameter is extremely important as it can dramatically speed up the time it takes to train model! The more CPU threads/cores can supply to dlib, the faster model will train. Default this value to the total number of CPUs on system; however, you can set this value as any integer (provided it’s less-than-or-equal-to the number of CPUs on your system).

After set up options for training script, the final step is to simply call train\_shape\_predictor:

dlib.train\_shape\_predictor(training\_directory, model\_output, options)

2.3.2. Training result and Evaluation

#### 

Figure 17: Training verbose

The entire training process took 15m11s. To verify that your shape predictor has been serialized to disk, ensure that custom\_predictor.dat has been created in directory structure (training\_directory)

Then evaluate its performance on both training and testing sets to verify that it’s not overfitting and that results will (ideally) generalize to images outside the training set.

error = dlib.test\_shape\_predictor(training\_directory, model\_output)

or

error = dlib.test\_shape\_predictor(testing\_directory, model\_output)

When both of these arguments are provided via the command line, dlib will handle evaluation. Dlib handles computing the mean average error (MAE) between the predicted landmark coordinates and the ground-truth landmark coordinates. The smaller the MAE, the better the predictions.

- Evaluate our custom landmark predictor on the training set:



- Evaluate our custom landmark predictor on the testing set:



1. Testing and reviewing

3.1. Implement the shape predictor inference script

The model implement process includes 10 steps:

- Loading the face detector. The detector allows us to find a face in an image/video prior to localizing landmarks on the face. We’ll be using dlib’s HOG + Linear SVM face detector. Alternatively, you could use Haar cascades (great for resource-constrained, embedded devices) or a more accurate [deep learning face detector](https://pyimagesearch.com/2018/02/26/face-detection-with-opencv-and-deep-learning/)

- Loading the facial landmark predictor

- Initializing our webcam stream

- Loop over frames from video

- Grab a frame, resize it, and convert to grayscale

- Applies face detection using dlib’s HOG + Linear SVM algorithm

- Take dlib’s rectangle object and convert it to OpenCV’s standard (x, y, w, h) bounding box ordering

- Use our custom dlib shape predictor to predict the location of our landmarks

- Convert the returned coordinates to a NumPy array

- Loop over the predicted landmark coordinates and draw them individually as small dots on the output frame

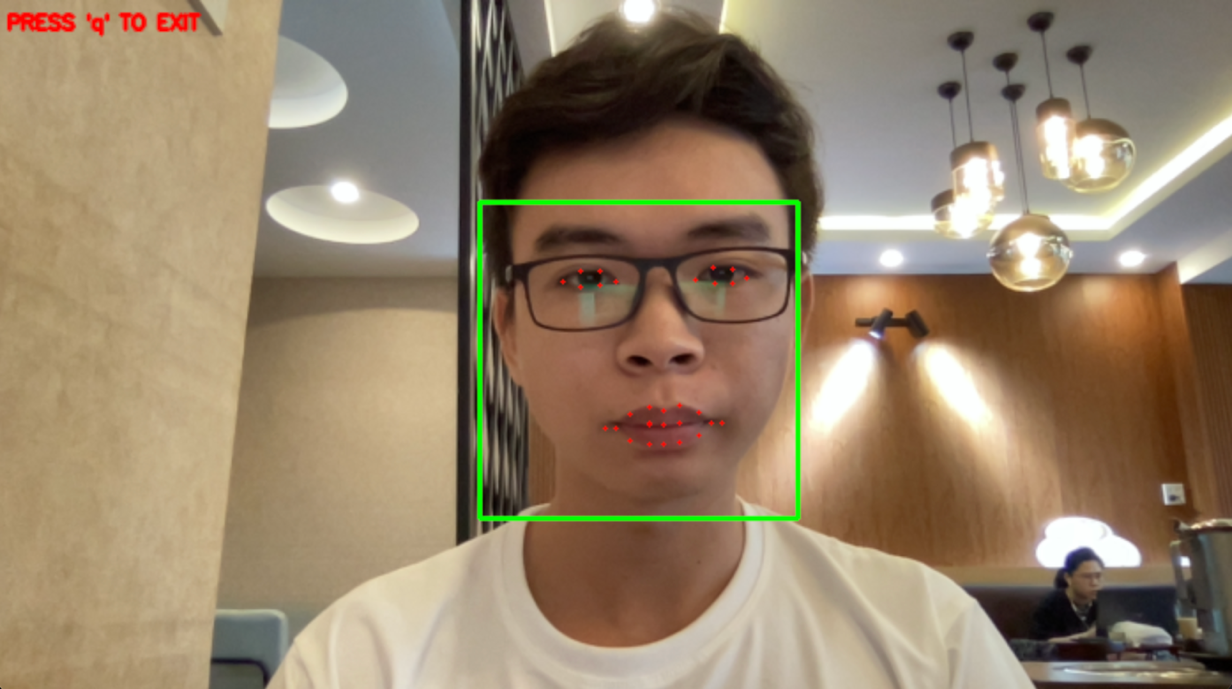


Figure 18: Testing in good light

3.2. Benchmark

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | STASM | CompASM | EXEM | RCPR | SDM | ESR | ERT (ours) |
| LFPW | - | - | 0.040 | 0.035 | 0.035 | 0.034\*\* | 0.038 |
| HELEN | 0.111 | 0.091 | - | 0.065 | 0.059 | 0.059 | 0.049 |
| IBUG\*\*\* | - | - | - | - | 0.075 | 0.075 | 0.064 |

Table 2: Benchmark\* with other model

\* The reported results are the average landmark distance from the ground-truth landmarks normalized by the inter-ocular distance. In all the experiments its is assumed that the bounding box of the face is given. In practice we used OpenCV for detecting the faces. For the cases that the face detector failed, we generated a bounding box with at least 80% overlap with the ground truth bounding box.

\*\* Our reimplementation of ESR performed significantly worse than the result reported in their original paper (~0.043 as opposed to 0.034).

\*\*\* We only used the AFW subset in addition to the training set of LFPW, and HELEN for training. For test we used IBUG set with test of LFPW and HELEN.

**STASM, CompASM**: [Interactive Facial Feature Localization](http://www.ifp.illinois.edu/~vuongle2/helen/eccv2012_helen_final.pdf).  
**EXEM**: [Localizing Parts of Faces Using a Consensus of Exemplars](http://homes.cs.washington.edu/~neeraj/papers/nk_cvpr2011_faceparts.pdf).  
**RCPR**: [Robust Face Landmark Estimation Under Occlusion](http://vision.caltech.edu/~xpburgos/papers/ICCV13%20Burgos-Artizzu.pdf)  
**SDM**: [Supervised Descent Method and its Applications to Face Alignment](http://www.ri.cmu.edu/pub_files/2013/5/main.pdf).  
**ESR**: [Face Alignment by Explicit Shape Regression](http://research.microsoft.com/en-us/people/yichenw/cvpr12_facealignment.pdf).

CONCLUSION

1. Summary of Results

From the analysis and processing of the input dataset, we find that it is possible to add or subtract custom points on the facial landmark dataset to suit the problem we need to solve (speed, model size).

After the process of research and implement, the result obtained will be a machine learning model that can predict facial landmarks. In addition, we can not only recognize facial landmarks, we can also detection other objects by creating our own dataset and training based on dlib’s library.

Customizing the parameters in the dlib library depends on your problem solving needs, so research and comparision is essential when adjusting parameters.

1. Future Work

Developing more data about facial points makes the dataset more diverse and suitable for many subjects.

Build an interface so that users can create their own datasets according to the requirements of the object detection problem and train a machine learning model from that custom dataset.

REFERENCES

1. Đ. T. Nghị, P. N. Khang, “Nguyên lý máy học”, *NXB Đại học Cần Thơ*, 2012.
2. G. Bradski, A. Kaehler, “Learning OpenCV”, *O’Reilly Media*, CA, USA, 2008.
3. D. E. King, “Dlib-ml: A Machine Learning Toolkit”, *Journal of Machine Learning Research 10*, pp.1755-1758, 2009.
4. Q. Bi, K. E. Goodman, J. Kaminsky, J. Lessler, “What is Machine Learning? A Primer for the Epidemiologist”, *American Journal of Epidemiology*, 188(22), pp.2222-2239, 2019.
5. R. Szeliski, “Computer Vision: Algorithms and Applications”, *Springer*, London, England, 2011.
6. A. Dey, “Machine Learning Algorithms: A Review”, *International Journal of Computer Science and Information Technologies*, 7(3), pp.1174-1179, 2016.
7. M. Andrychowicz, M. Denil, S. G. Colmenarejo, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, N. de Freitas, “Learning to learn by gradient descent by gradient descent”, *arXiv e-prints*, p .arXiv:1606.04474v2, 11 2016.
8. V. Kazemi, J. Sullivan, “One Millisecond Face Alignment with an Ensemble of Regression Trees”, *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 2014.
9. R. Perrot, P. Bourdon, D. Helbert, “Implementing Cascaded of Regression tree-based Face Landmarking: an in-Depth Overview”, *Image and Vision Computing*, Elsevier, 102, pp.103976, 2020.
10. A. Rosebrock (2019), “Training a custom dlib shape predictor”, Available: <https://www.pyimagesearch.com/2019/12/16/training-a-custom-dlib-shape-predictor/>. [Accessed June 1, 2021]