Face Detection by Implementation of Haar Cascade with CNNs

Dinh Quang Hien
Hanoi University of Science and Technology
Hanoi, Vietnam
hien.dq224281@sis.hust.edu.vn

Nguyen Huy Hoang
Hanoi University of Science and Technology
Hanoi, Vietnam
hoang.nh224313@sis.hust.edu.vn

Abstract

Face detection is a fundamental task in computer vision, critical for applications such as surveillance, authentication, and image processing. This research explores the implementation and comparative analysis of two prominent approaches to face detection: the Viola-Jones framework and Convolutional Neural Networks (CNNs). The Viola-Jones algorithm, a pioneering method for real-time face detection, leverages Haar-like features, integral images, and AdaBoost classifiers for efficient and robust performance. On the other hand, CNNs represent a data-driven, deep learning approach that learns hierarchical feature representations from raw pixel data, often surpassing traditional methods in accuracy.

In this work, we implement and optimize both techniques, evaluating them on diverse datasets for performance metrics such as detection accuracy, computational efficiency, and robustness under challenging conditions (e.g., variations in lighting, occlusions, and pose).

1. Introduction

Face detection is an incredible technology that lets machines identify and locate human faces in images or videos.Moreover, it serves as the first step in many applications, such as facial recognition, emotion analysis, augmented reality, and security systems .It's something we often take for granted, but it's truly fascinating—especially when you think about how, every time we open a camera, it instantly recognizes our faces.

We've always wondered how it manages to do that so quickly and accurately, no matter the angle or lighting. Face detection is like giving machines the ability to "see" the world, using patterns and features like the symmetry of our eyes, nose, and mouth. Over the years, it has evolved from simple algorithms to advanced neural networks that mimic how our own brains process visual information. It's not just about the tech—it's about the way it's seamlessly integrated into our lives, from unlocking phones to adding fun filters, making it feel almost magical. With the importance and issues mentioned above, this project aims to develop a simple implementation of Viola-Jones machine learning approach to detect face while also testing out the CNNs implementation for face recognition purposes.

1.1. Desired Outcomes:

For Viola-Jones Approach [4]:

Input: The input consists of 19x19 image

Output: The model should output an indication to specify whether that's a face or not.

For CNNs Approach:

Input: A lot of jpg files

Output: The model should be able to detect a person which can be indicated as bounding boxes around the face.

1.2. Challenges

- **Limited Functionality of Model based on Viola-Jones Approach:** While we were able to test it on a certain dataset but when we use it to test on other images, it tends to do work as expected and specially making it near-impossible to use it for real-time detection tasks. Overall, this implementation is very dataset specific.

- **Limited labeled data:** Large, high-quality labeled datasets for face recognition may be scarce and the fact that we can't go around taking pictures of people's faces without permission is also a big factor, which can make training deep learning models challenging.

2. Related Work

[1] This work introduces a robust face recognition system using a deep convolutional neural network (CNN) inspired by VGGNet. The model is trained on the large-scale VGGFace dataset, containing 2.6 million images of 2,622 individuals, with a softmax loss to classify faces. The network outputs high-dimensional face descriptors, which are used for identification and verification by comparing their similarity. Preprocessing includes facial alignment based on key landmarks. The method achieves state-of-the-art results on benchmarks like LFW, highlighting the power of deep learning and large datasets for accurate and scalable face recognition.

[2] This work addresses the challenge of training deep neural networks for face recognition with limited data. The authors propose a method that combines data augmentation, transfer learning, and a novel network design to enhance model performance. They leverage pre-trained networks and fine-tune them on small datasets, supplemented by synthetic data generated through augmentation techniques. Their approach reduces overfitting and improves generalization. Experimental results demonstrate competitive accuracy on benchmark datasets, showcasing the effectiveness of their method for face recognition when training data is scarce.

[3] This work presents a method for detecting human facial emotions—such as happiness, sadness, disgust, anger, fear, neutrality, and surprise—by combining Haar Cascade classifiers for face detection with Convolutional Neural Networks (CNNs) for emotion classification. Utilizing the FER2013 dataset, which comprises 35,887 images across seven emotion categories, the proposed CNN architecture includes six convolutional layers, three max-pooling layers, and four fully connected layers, culminating in a softmax output layer. Experimental results indicate that increasing the number of training epochs leads to lower Mean Squared Error (MSE) and higher accuracy, with the model achieving a validation accuracy of 65.59%.

In recent years, deep learning techniques have shown great promise in automating both feature extraction and classification from raw image data.

3. Proposed Method

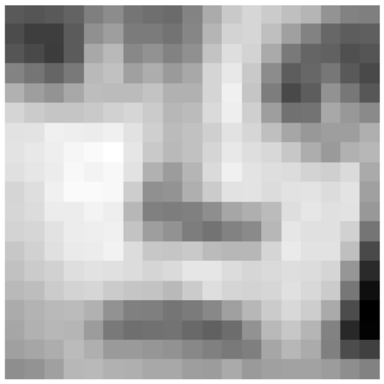
3.1 Haar Cascade Classifier

The Haar Cascade Classifier is a machine learning-based technique for object detection, specifically used for face detection. Developed by Paul Viola and Michael Jones [4], it works by using a cascade of classifiers trained on large datasets of positive (face) and negative (non-face) images. The process consists of four main stages:

1. **Haar Feature Selection:** In this stage, Haar features are calculated by evaluating differences in pixel intensities between adjacent rectangular regions of an image, helping distinguish various regions that could represent facial features. A wide variety of these Haar-like features are necessary to effectively capture face details.
2. **Creating an Integral Image:** To reduce computational complexity, the algorithm uses an integral image, allowing the sum of pixel intensities to be calculated for any rectangular region in constant time, significantly speeding up the process.
3. **AdaBoost:** This technique selects the most relevant features for classification. Not all computed features are useful for identifying faces, so AdaBoost is employed to focus on the features that are most effective.
4. **Cascading Classifiers:** The classifier is organized in multiple stages. Instead of applying all features to every region of the image, the features are grouped into separate stages, which are applied sequentially. If any stage fails to recognize a face, the region is discarded early in the process, saving time and computational resources. Only the regions that pass all stages are classified as containing a face.

3.1.1 Data structure

Input Images:



19x19 GrayScale Images

3.1.2 Algorithm

1. **Haar-like Features**

Haar-like features are calculated as the difference in pixel intensities between adjacent rectangular regions. The feature value F for a given feature type can be expressed as:

Two-rectangle feature (horizontal or vertical):

$$F = \mathrm{Sum}(R_1) - \mathrm{Sum}(R_2)$$

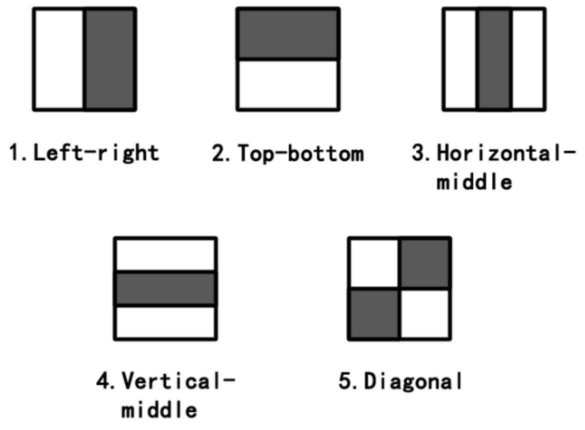
Three-rectangle feature:

$$F = \mathrm{Sum}(R_1) - 2 \times \mathrm{Sum}(R_2) + \mathrm{Sum}(R_3)$$

Four-rectangle feature:

$$F = (\mathrm{Sum}(R_1) + \mathrm{Sum}(R_4)) - (\mathrm{Sum}(R_2) + \mathrm{Sum}(R_3))$$

Where R_1, R_2, R_3, and R_4 represent regions in the image.



2. **Integral Image**

The integral image $I_{\mathrm{int}}(x, y)$ is computed as:

$$I_{\mathrm{int}}(x, y) = \sum_{i=0}^{x} \sum_{j=0}^{y} I(i, j)$$

Where $I(i, j)$ is the pixel intensity at location (i, j).
Using the integral image, the sum of intensities over a rectangular region (x_1, y_1) to (x_2, y_2) can be computed in constant time:

$$\mathrm{Sum}(R) = I_{\mathrm{int}}(x_2, y_2) - I_{\mathrm{int}}(x_2, y_1 - 1) - I_{\mathrm{int}}(x_1 - 1, y_2) + I_{\mathrm{int}}(x_1 - 1, y_1 - 1)$$

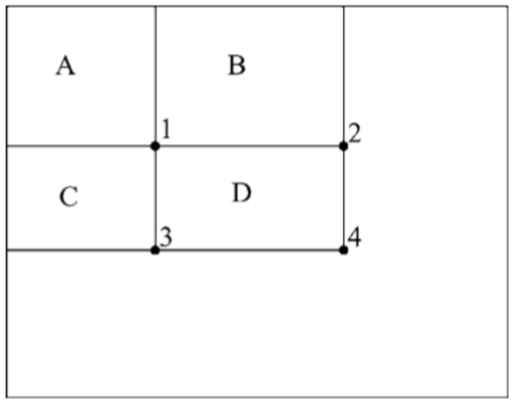


Figure 3. The sum of the pixels within rectangle D can be computed with four array references. The value of the integral image at location 1 is the sum of the pixels in rectangle A. The value at location 2 is $A + B$, at location 3 is $A + C$, and at location 4 is $A + B + C + D$. The sum within D can be computed as $4 + 1 - (2 + 3)$.

3. **AdaBoost**

- Given example images $(x_1, y_1), \dots, (x_n, y_n)$ where $y_i = 0, 1$ for negative and positive examples respectively.
- Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively, where m and l are the number of negatives and positives respectively.
- For $t = 1, \dots, T$:
  1. Normalize the weights,
  $$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$
  so that w_t is a probability distribution.
  2. For each feature, j, train a classifier h_j which is restricted to using a single feature. The error is evaluated with respect to w_t, $\epsilon_j = \sum_i w_i \left| h_j(x_i) - y_i \right|$.
  3. Choose the classifier, h_t, with the lowest error ϵ_t.
  4. Update the weights:
  $$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}$$
  where $e_i = 0$ if example x_i is classified correctly, $e_i = 1$ otherwise, and $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.
- The final strong classifier is:
$$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$
where $\alpha_t = \log \frac{1}{\beta_t}$

4. **Cascade Structure**

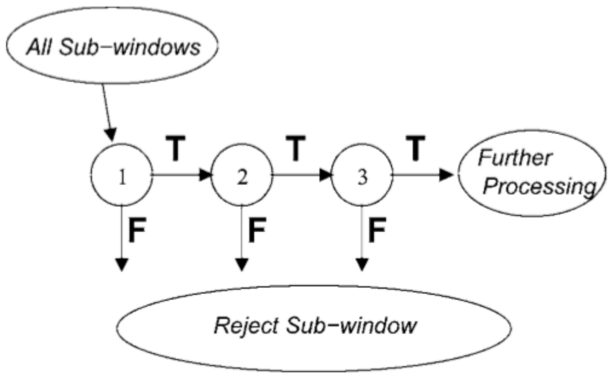
The cascade structure consists of multiple stages Sk where each stage is a strong classifier trained using AdaBoost. A region passes through all stages to be classified as a detection.

Cascade Logic:

- Let Tk be the threshold for stage Sk.
- For an input x:
  - If Sk(x) < Tk then reject the region.
  - Else, pass the region to the next stage.

BoostedCascade

To speed up detection process, as well as to decrease the false positive rate while sustaining a high detection rate (i.e. to improve precision), Viola and Jones invent the boosted cascade. That is, to cascade multiple AdaBoost classifiers.



5. **Sliding Window**

The sliding window approach systematically scans the image I(x, y) using a window of size w×h:

Window (x, y) = I (x+w, y+h)

At each position, the cascade classifier is applied. The process is repeated at multiple scales by resizing the image I to detect objects of varying sizes.

3.2 CNNs

3.1.1 Data Acquisition and Preprocessing

- **Data Collection**: The script `01_face_dataset.py` captures 70 images of a user's face via webcam, storing them in a 'dataset' directory. This process ensures a sufficient number of samples for training the model.
- **Face Detection**: The system employs the Haar Cascade classifier (`haarcascade_frontalface_default.xml`) to detect and extract facial regions from the captured images. Haar Cascades are efficient for real-time face detection due to their low computational cost.
- **Data Storage Structure**: Captured images are stored in a hierarchical directory structure, with each subdirectory corresponding to a unique user ID. This organization facilitates efficient data retrieval during training and recognition phases.

3.1.2 Model Architecture

Convolutional Neural Network (CNN): The model is defined in Model.py and comprises multiple layers:

  - **Convolutional Layers**: Extract spatial features from input images by applying convolution operations with learnable filters.
  - **Activation Functions**: Introduce non-linearities into the model, enabling it to capture complex patterns.
  - **Pooling Layers**: Reduce the spatial dimensions of feature maps, decreasing computational load and mitigating overfitting.
  - **Fully Connected Layers**: Integrate features learned by convolutional layers to perform classification.
- **Model Compilation**: The model is compiled with a loss function suitable for classification tasks and an optimizer that adjusts weights during training to minimize loss.

3.1.3 Training Process

- **Data Preparation**: The script 02_face_training.py loads images from the dataset, labels them appropriately, and preprocesses them (e.g., resizing, normalization) to ensure consistency.
- **Training**: The model is trained on the preprocessed dataset, adjusting its weights through backpropagation to minimize classification errors. The training process involves multiple epochs, with each epoch iterating over the entire dataset.
- **Model Saving**: Upon completion, the trained model's weights are saved as 'trained_model.h5' for future use in recognition tasks.

3.1.4 Recognition Phase

- **Real-Time Recognition**: The script 03_face_recognition.py initializes the webcam and utilizes the trained CNN model to identify faces in real-time. Detected faces are compared against the stored representations to determine identity.

3.1.5 Computational Complexity

- **Training Complexity**: Training CNNs is computationally intensive, with complexity dependent on factors such as the number of layers, filter sizes, and dataset size. The backpropagation algorithm, essential for training, has a time complexity of O(n * m), where 'n' is the number of training samples and 'm' is the number of model parameters.
- **Inference Complexity**: During recognition, the model's inference time is influenced by its depth and architecture. While CNNs can be computationally demanding, optimizations and hardware accelerations (e.g., GPUs) enable real-time performance.

4. Experiment

4.1 Haar Cascade:

Dataset:

The data is described at http://cbcl.mit.edu/software-datasets/FaceData2.html, and downloaded from www.ai.mit.edu/courses/6.899/lectures/faces.tar.gz
Each image is 19x19 and greyscale.
Training set: 2,429 faces, 4,548 non-faces
Test set: 472 faces, 23,573 non-faces.

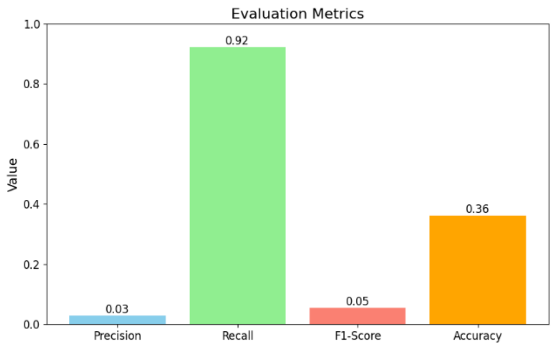
We have tested the Haar Cascade model and it yield rather promising results based on the dataset that it was trained and tested on. Although the results of the models can be improved due to the computational power of our's laptop it could omly handle a sizable amount of features. Therefore, as we loaded in 2429 images that was labeled "Face" while the other 4548 images were labeled "Non-Face" the model were only able to generated 2496 features created. Here's *(min, max)* features height and width:

min_feature_height = 8
max_feature_height = 10
min_feature_width = 8
max_feature_width = 10

Results:

Faces: 435/472 (92.16%),
Non-Faces: 8254/23573 (35.01%)
Face-Specific Metrics:
  Precision (Faces): 0.9216
  Recall (Faces): 0.9216
  F1-Score (Faces): 0.9216

Overall Metrics:



Discussion:

The evaluation metrics of the Haar Cascade model highlight both its strengths and its limitations when applied to the CBCL face dataset. While the model performs well in detecting faces, achieving a recall of **92.16%**, the overall performance across both face and non-face classifications is less impressive, with a precision of **2.76%**, an F1-score of **5.36%**, and an accuracy of **36.14%**. These metrics reveal critical insights into the model's behavior and areas for improvement.

Strengths: High Recall for Faces

The recall metric for faces, at **92.16%**, indicates that the model is highly effective at detecting true positive instances of faces in the dataset. This suggests that the Haar Cascade approach successfully learns features that are representative of face patterns, allowing it to localize most face instances in the test set. This high recall makes the model particularly useful in applications where minimizing missed detections (false negatives) is crucial, such as in surveillance or safety-critical systems.

Weaknesses: Low Precision and F1-Score

The low precision (**2.76%**) signifies that the model generates a substantial number of false positives, incorrectly classifying non-face images as faces. This drastically reduces its reliability in scenarios where accurate face detection is required. Consequently, the F1-score, a harmonic mean of precision and recall, is also low at **5.36%**, reflecting the imbalance between the model's ability to detect true positives and its tendency to misclassify non-face images.

Overall Accuracy and Class Imbalance

The overall accuracy of **36.14%** is influenced heavily by the class imbalance in the test set, which contains significantly more non-face images (**23,573 non-faces vs. 472 faces**). Despite correctly identifying a high percentage of faces, the model struggles with non-face images, leading to poor performance on the dominant class. This imbalance likely skews the model's decision boundary, resulting in suboptimal performance for non-face classifications.

4.2 CNNs:

5. Conclusion

While the Haar Cascade model demonstrates strong recall for faces, its low precision and poor overall accuracy indicate that it struggles with the high false-positive rate for non-face images. These results suggest that while Haar Cascades remain a viable baseline for face detection, further optimization or the use of more advanced techniques is necessary to achieve a more balanced and robust performance across all classes. With additional computational resources and methodological enhancements, this model has the potential to serve as an effective foundation for face detection in constrained settings.
Given that the Haar Cascade model was implemented from scratch and has demonstrated promising recall performance on the CBCL face dataset, future work can focus on enhancing the model’s utility and extending its application to real-time detection tasks. These plans include both improving the current model and exploring its deployment in practical scenarios :

1. Improving Model Robustness

- **Expanding Feature Sets**: Increase the range and diversity of Haar-like features to improve the model's capacity for distinguishing faces from non-faces. By allowing larger feature sizes and incorporating additional variations, the model may better handle complex patterns in real-world data.
- **Optimizing Training**: Experiment with advanced boosting algorithms, such as GentleBoost or RealBoost, to enhance the model's learning process, particularly for handling class imbalances and reducing false positives.
- **Fine-Tuning Hyperparameters**: Systematically explore hyperparameter tuning, including adjusting the number of weak classifiers, learning rates, and feature thresholds, to improve overall precision and accuracy.

2. Transition to Real-Time Applications

Implementing the Haar Cascade model for real-time face detection is a natural progression. Key steps include:

- **Integration with Real-Time Video Streams**: Develop a pipeline for processing video frames from cameras in real time. This involves optimizing the model for low-latency inference.
- **Optimized Implementation**: Rewrite the model in a high-performance programming language like C++ or leverage frameworks such as OpenCV to enable efficient processing.
- **Hardware Acceleration**: Utilize GPU or FPGA acceleration to further speed up computation, ensuring the model operates at frame rates suitable for real-time applications (e.g., 30+ frames per second).

3. Performance Evaluation on Real-World Data

- Test the model on more diverse datasets, including real-world images and videos, to evaluate its generalization performance. Incorporating datasets with varying lighting, poses, and occlusions can reveal areas for improvement.
- Conduct experiments in different environments (e.g.,

indoor, outdoor, low light) to assess its robustness and adaptiveness.

4. Enhancing Real-World Usability

- **Multi-Face Detection**: Extend the model to detect multiple faces in a single frame, incorporating techniques like non-maximum suppression to handle overlapping detections.
- **Face Tracking**: Integrate the detection pipeline with tracking algorithms such as Kalman Filters or SORT (Simple Online and Realtime Tracking) to maintain consistent identification of faces across frames.
- **Confidence Scoring**: Add confidence scores to predictions, allowing dynamic thresholds for balancing precision and recall based on application requirements.

5. Exploration of Hybrid Approaches

While Haar Cascade is a reliable and interpretable algorithm, integrating it with modern machine learning techniques could yield significant performance gains:

- **CNN-Assisted Detection**: Combine the efficiency of Haar-like features for quick localization with CNNs for more accurate classification, particularly in challenging scenarios.
- **Feature Fusion**: Investigate the fusion of Haar features with additional descriptors (e.g., Histogram of Oriented Gradients, SIFT) to enhance detection accuracy.

6. Applications in Real-Time Systems

Finally, the ultimate goal is to deploy the enhanced model in real-world applications, including:

- **Surveillance Systems**: Use the model for monitoring and detecting faces in crowded areas, improving safety and security.
- **Interactive Applications**: Incorporate face detection in human-computer interaction systems, such as virtual assistants, gaming interfaces, or AR/VR devices.
- **Mobile Deployment**: Adapt the model for resource-constrained environments like mobile phones and embedded systems, enabling widespread usage in consumer applications.

References

[1] Omkar M Parkhi, Andrea Vedaldi, and Andrew Zisserman. Deep face recognition. In BMVC, vol ume 1, page 6, 2015.

[2] Xi Peng, Nalini Ratha, and Sharathchandra Pankanti. Learning face recognition from limited training data using deep neural networks. In Pat tern Recognition (ICPR), 2016 23rd International Conference on, pages 14421447. IEEE, 2016.

[3] Rasheed, M., Kausar, N., Mehmood, A., Jamil, N., & Shah, M. I. Face Recognition Emotions Detection Using Haar Cascade Classifier and Convolutional Neural Network. In ResearchGate, 2022.

[4] Viola, P., & Jones, M. J. (2001). Rapid object detection using a boosted cascade of simple features. *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (CVPR 2001),1,511,518.