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 not to work as expected and our implementation can't really be used 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.

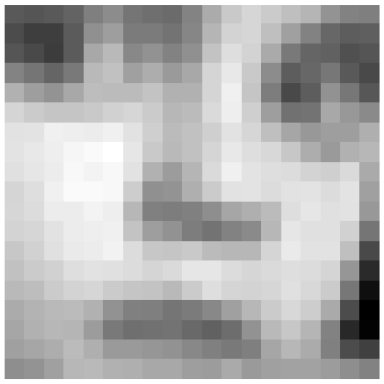
2. **Creating an Integral Image:** To reduce computational complexity, the algorithm uses an integral image, allowing the sum of pixel intensities to be calculated for any rectangular region in constant time, significantly speeding up the process.

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):

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

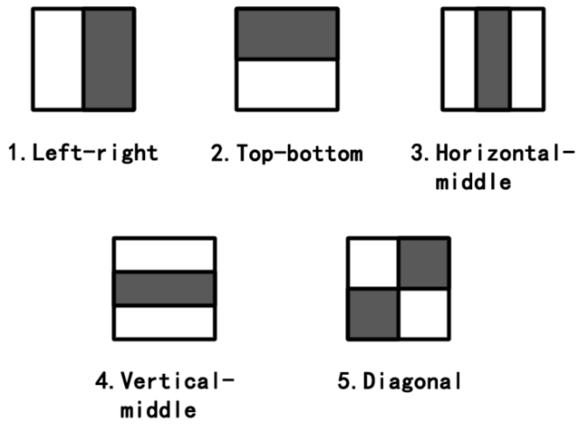
Three-rectangle feature:

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

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 $I_{\text{int}}(x, y)$ is computed as:

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

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:

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

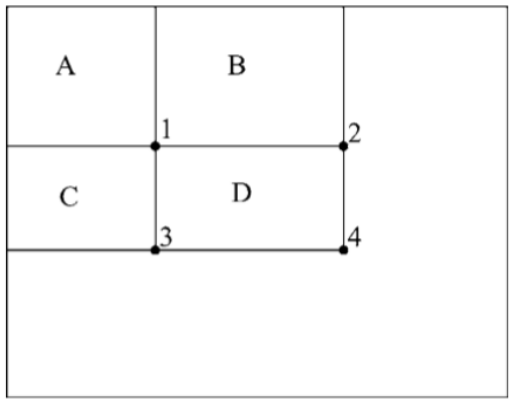


Figure 3. The sum of the pixels within rectangle D can be computed with four array references. The value of the integral image at location 1 is the sum of the pixels in rectangle A. The value at location 2 is $A + B$, at location 3 is $A + C$, and at location 4 is $A + B + C + D$. The sum within D can 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 $w_t$, $\epsilon_j = \sum_i w_i \, |h_j(x_i) - y_i|$.
    3. Choose the classifier, $h_t$, with the lowest error $\epsilon_t$.
    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**

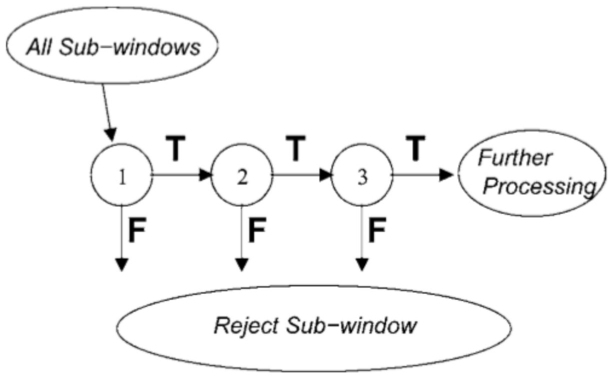
The cascade structure consists of multiple stages Sk where each stage is a strong classifier trained using AdaBoost. A region passes through all stages to be classified as a detection.

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.

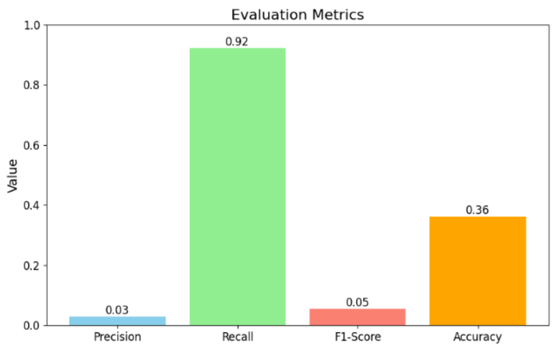
We have tested the Haar Cascade model and it yield rather promising results based on the dataset that it was trained and tested on. Although the results of the models can be improved due to the computational power of our's laptop it could omly handle a sizable amount of features. Therefore, as we loaded 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.

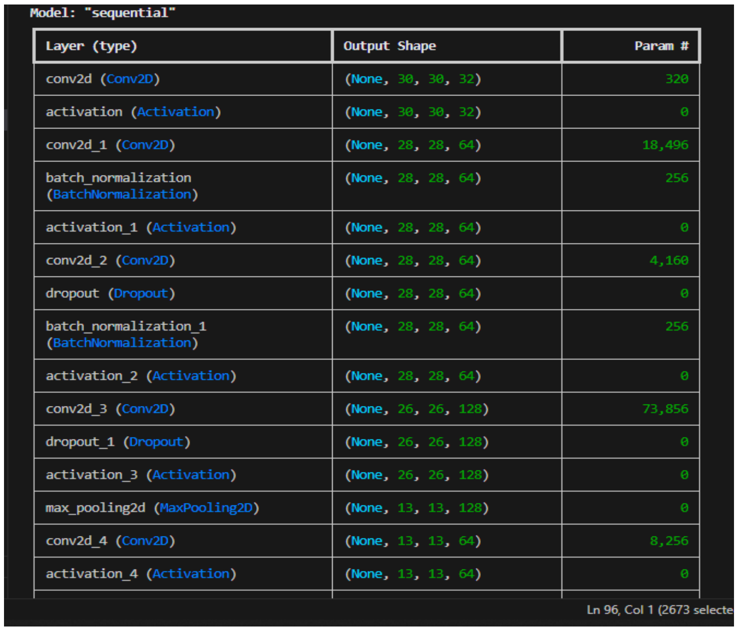
4.2 CNNs:

Dataset:

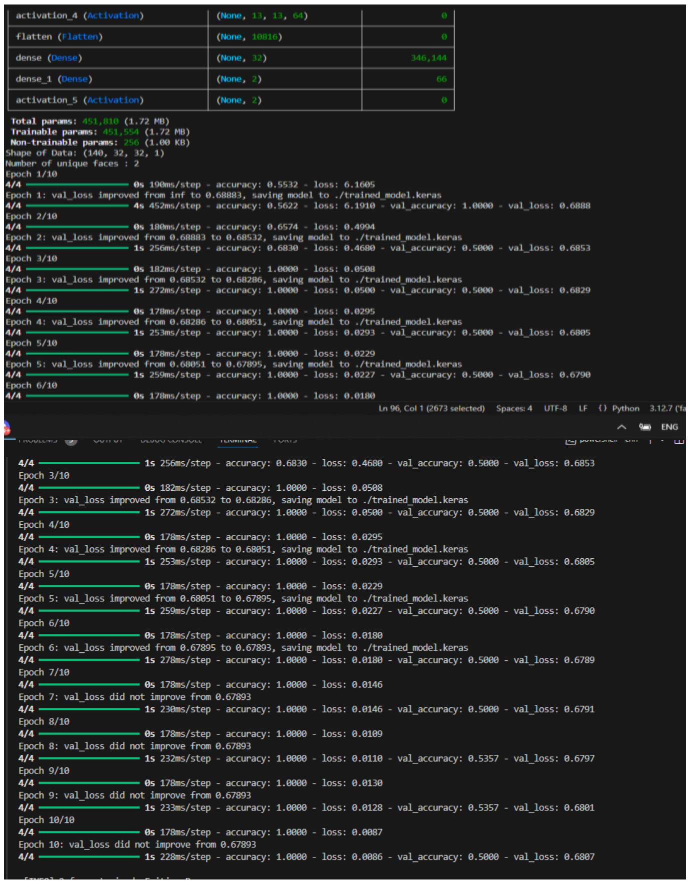
Dataset Preview

The dataset consists of 140 grayscale facial images, each resized to 32×32 pixels, with two distinct user classes. As visualized in the dataset preview, the images are arranged systematically, showcasing slight variations in facial expressions, lighting conditions, and the presence of accessories such as glasses. The first set of images predominantly displays faces without glasses, while the second set introduces variations, including glasses, further diversifying the dataset. However, the small dataset size limits the CNN's ability to generalize effectively, as it lacks sufficient diversity and quantity to capture the complexities of real-world facial recognition scenarios. The uniformity and limited number of samples per class pose challenges for the model in distinguishing subtle features, contributing to overfitting and reduced validation performance.

Results:



```
Model: "sequential"
┌──────────────────────────────────┬─────────────────────────┬──────────────┐
│ Layer (type)                     │ Output Shape            │      Param # │
├──────────────────────────────────┼─────────────────────────┼──────────────┤
│ conv2d (Conv2D)                  │ (None, 30, 30, 32)      │          320 │
├──────────────────────────────────┼─────────────────────────┼──────────────┤
│ activation (Activation)          │ (None, 30, 30, 32)      │            0 │
├──────────────────────────────────┼─────────────────────────┼──────────────┤
│ conv2d_1 (Conv2D)                │ (None, 28, 28, 64)      │       18,496 │
├──────────────────────────────────┼─────────────────────────┼──────────────┤
│ batch_normalization              │ (None, 28, 28, 64)      │          256 │
│ (BatchNormalization)             │                         │              │
├──────────────────────────────────┼─────────────────────────┼──────────────┤
│ activation_1 (Activation)        │ (None, 28, 28, 64)      │            0 │
├──────────────────────────────────┼─────────────────────────┼──────────────┤
│ conv2d_2 (Conv2D)                │ (None, 28, 28, 64)      │        4,160 │
├──────────────────────────────────┼─────────────────────────┼──────────────┤
│ dropout (Dropout)                │ (None, 28, 28, 64)      │            0 │
├──────────────────────────────────┼─────────────────────────┼──────────────┤
│ batch_normalization_1            │ (None, 28, 28, 64)      │          256 │
│ (BatchNormalization)             │                         │              │
├──────────────────────────────────┼─────────────────────────┼──────────────┤
│ activation_2 (Activation)        │ (None, 28, 28, 64)      │            0 │
├──────────────────────────────────┼─────────────────────────┼──────────────┤
│ conv2d_3 (Conv2D)                │ (None, 26, 26, 128)     │       73,856 │
├──────────────────────────────────┼─────────────────────────┼──────────────┤
│ dropout_1 (Dropout)              │ (None, 26, 26, 128)     │            0 │
├──────────────────────────────────┼─────────────────────────┼──────────────┤
│ activation_3 (Activation)        │ (None, 26, 26, 128)     │            0 │
├──────────────────────────────────┼─────────────────────────┼──────────────┤
│ max_pooling2d (MaxPooling2D)     │ (None, 13, 13, 128)     │            0 │
├──────────────────────────────────┼─────────────────────────┼──────────────┤
│ conv2d_4 (Conv2D)                │ (None, 13, 13, 64)      │        8,256 │
├──────────────────────────────────┼─────────────────────────┼──────────────┤
│ activation_4 (Activation)        │ (None, 13, 13, 64)      │            0 │
├──────────────────────────────────┼─────────────────────────┼──────────────┤
                                                  Ln 96, Col 1 (2673 selecte
```



```
│ activation_4 (Activation)          │ (None, 13, 13, 64)      │              0 │
├────────────────────────────────────┼─────────────────────────┼────────────────┤
│ flatten (Flatten)                  │ (None, 10816)           │              0 │
├────────────────────────────────────┼─────────────────────────┼────────────────┤
│ dense (Dense)                      │ (None, 32)              │        346,144 │
├────────────────────────────────────┼─────────────────────────┼────────────────┤
│ dense_1 (Dense)                    │ (None, 2)               │             66 │
├────────────────────────────────────┼─────────────────────────┼────────────────┤
│ activation_5 (Activation)          │ (None, 2)               │              0 │
└────────────────────────────────────┴─────────────────────────┴────────────────┘
 Total params: 451,810 (1.72 MB)
 Trainable params: 451,554 (1.72 MB)
 Non-trainable params: 256 (1.00 KB)
Shape of Data: (140, 32, 32, 1)
Number of unique faces : 2
Epoch 1/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 190ms/step - accuracy: 0.5532 - loss: 6.1605
Epoch 1: val_loss improved from inf to 0.68883, saving model to ./trained_model.keras
4/4 ━━━━━━━━━━━━━━━━━━━━ 4s 452ms/step - accuracy: 0.5622 - loss: 6.1910 - val_accuracy: 1.0000 - val_loss: 0.6888
Epoch 2/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 180ms/step - accuracy: 0.6574 - loss: 0.4994
Epoch 2: val_loss improved from 0.68883 to 0.68532, saving model to ./trained_model.keras
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 256ms/step - accuracy: 0.6830 - loss: 0.4680 - val_accuracy: 0.5000 - val_loss: 0.6853
Epoch 3/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 182ms/step - accuracy: 1.0000 - loss: 0.0508
Epoch 3: val_loss improved from 0.68532 to 0.68286, saving model to ./trained_model.keras
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 272ms/step - accuracy: 1.0000 - loss: 0.0500 - val_accuracy: 0.5000 - val_loss: 0.6829
Epoch 4/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 178ms/step - accuracy: 1.0000 - loss: 0.0295
Epoch 4: val_loss improved from 0.68286 to 0.68051, saving model to ./trained_model.keras
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 253ms/step - accuracy: 1.0000 - loss: 0.0293 - val_accuracy: 0.5000 - val_loss: 0.6805
Epoch 5/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 178ms/step - accuracy: 1.0000 - loss: 0.0229
Epoch 5: val_loss improved from 0.68051 to 0.67895, saving model to ./trained_model.keras
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 259ms/step - accuracy: 1.0000 - loss: 0.0227 - val_accuracy: 0.5000 - val_loss: 0.6790
Epoch 6/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 178ms/step - accuracy: 1.0000 - loss: 0.0180
                                         Ln 96, Col 1 (2673 selected)   Spaces: 4   UTF-8   LF   {} Python   3.12.7
                                                                                                        ENG
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 256ms/step - accuracy: 0.6830 - loss: 0.4680 - val_accuracy: 0.5000 - val_loss: 0.6853
Epoch 3/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 182ms/step - accuracy: 1.0000 - loss: 0.0508
Epoch 3: val_loss improved from 0.68532 to 0.68286, saving model to ./trained_model.keras
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 272ms/step - accuracy: 1.0000 - loss: 0.0500 - val_accuracy: 0.5000 - val_loss: 0.6829
Epoch 4/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 178ms/step - accuracy: 1.0000 - loss: 0.0295
Epoch 4: val_loss improved from 0.68286 to 0.68051, saving model to ./trained_model.keras
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 253ms/step - accuracy: 1.0000 - loss: 0.0293 - val_accuracy: 0.5000 - val_loss: 0.6805
Epoch 5/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 178ms/step - accuracy: 1.0000 - loss: 0.0229
Epoch 5: val_loss improved from 0.68051 to 0.67895, saving model to ./trained_model.keras
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 259ms/step - accuracy: 1.0000 - loss: 0.0227 - val_accuracy: 0.5000 - val_loss: 0.6790
Epoch 6/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 178ms/step - accuracy: 1.0000 - loss: 0.0180
Epoch 6: val_loss improved from 0.67895 to 0.67893, saving model to ./trained_model.keras
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 278ms/step - accuracy: 1.0000 - loss: 0.0180 - val_accuracy: 0.5000 - val_loss: 0.6789
Epoch 7/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 178ms/step - accuracy: 1.0000 - loss: 0.0146
Epoch 7: val_loss did not improve from 0.67893
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 230ms/step - accuracy: 1.0000 - loss: 0.0146 - val_accuracy: 0.5000 - val_loss: 0.6791
Epoch 8/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 178ms/step - accuracy: 1.0000 - loss: 0.0109
Epoch 8: val_loss did not improve from 0.67893
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 232ms/step - accuracy: 1.0000 - loss: 0.0110 - val_accuracy: 0.5357 - val_loss: 0.6797
Epoch 9/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 178ms/step - accuracy: 1.0000 - loss: 0.0130
Epoch 9: val_loss did not improve from 0.67893
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 233ms/step - accuracy: 1.0000 - loss: 0.0128 - val_accuracy: 0.5357 - val_loss: 0.6801
Epoch 10/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 178ms/step - accuracy: 1.0000 - loss: 0.0087
Epoch 10: val_loss did not improve from 0.67893
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 228ms/step - accuracy: 1.0000 - loss: 0.0086 - val_accuracy: 0.5000 - val_loss: 0.6807
```

Model Summary and Epochs

The CNN architecture comprised multiple convolutional, batch normalization, activation, dropout, and dense layers, resulting in a total of 451,810 parameters, with 451,554 trainable. The dataset consisted of 140 grayscale images of size 32×32 pixels with two unique face classes.

During training, the model's performance improved significantly across epochs. The training accuracy increased from 46.82% in the first epoch to 100% by the tenth epoch, while the training loss decreased from 15.88 to 0.0141. The model saved its weights at each epoch when the validation loss improved, indicating effective training. Validation loss consistently decreased from 0.6906 in the first epoch to 0.6706 in the final epoch. However, validation accuracy remained static at 50%, suggesting overfitting or a lack of sufficient generalization to unseen data.

Discussion:

The model demonstrated rapid convergence during training, achieving **perfect training accuracy by the seventh epoch**. This can be attributed to the depth of the network and the inclusion of batch normalization and dropout layers, which helped stabilize learning and prevent overfitting to some extent.

Despite the strong performance on the training data, the **validation accuracy plateaued at 50%,** highlighting a significant limitation. This indicates that the model failed to generalize well to the validation dataset, likely due to the following factors:

Dataset Size: The dataset contained only 140 samples, which is insufficient for training a deep CNN. The model likely memorized the training data rather than learning robust features for face recognition.

Class Imbalance: While the dataset had two unique face classes, the exact distribution of samples across these classes is unclear. Class imbalance can negatively impact the model's ability to learn effectively.

Validation Set Complexity: The validation set might have included images with features significantly different from the training set, leading to poor generalization.

Future Improvements:

To address these issues, the following strategies are proposed:

Data Augmentation: Techniques such as rotation, scaling, flipping, and cropping can artificially increase the dataset size and diversity, improving the model's generalization capability.

Larger Dataset: Acquiring a larger dataset with more diverse samples would enhance the model's ability to learn distinguishing features.

Regularization: Adding stronger regularization techniques, such as L2 weight decay, could help mitigate overfitting.

Transfer Learning: Employing a pre-trained model fine-tuned on the target dataset may significantly improve performance on small datasets.

The findings underscore the importance of dataset quality and quantity in training CNNs for face recognition. While the model's architecture and training process were effective, future work must address dataset limitations to achieve higher validation accuracy and robust real-world performance.

5. Conclusion

Both the Haar Cascade and CNN-based approaches demonstrate their potential for face detection and recognition tasks, albeit with unique strengths and challenges.

The Haar Cascade model, implemented from scratch, showcases strong recall for face detection, especially on the CBCL face dataset. However, its low precision and poor overall accuracy highlight challenges with false positives, particularly when distinguishing non-face images. Despite these limitations, Haar Cascades remain a viable baseline for face detection due to their efficiency and interpretability. Future enhancements, such as expanding feature sets, optimizing training algorithms, and transitioning to real-time applications with hardware acceleration, could improve the model's robustness and utility. Exploring hybrid approaches that integrate Haar features with modern machine learning techniques, such as CNNs, holds promise for achieving better precision and recall in real-world scenarios.

Similarly, the CNN-based model exhibited excellent training accuracy and loss reduction, achieving 100% training accuracy after ten epochs. However, the plateau in validation accuracy at 50% revealed overfitting and limited generalization capabilities. These challenges underscore the importance of larger, more diverse datasets and advanced regularization techniques to improve model robustness. While the CNN model outperformed in learning complex features, its applicability to real-world scenarios remains constrained by dataset size and generalization.

Both approaches highlight the importance of balancing computational efficiency, precision, and recall. While Haar Cascades offer a lightweight and interpretable solution for constrained settings, CNNs excel at learning intricate patterns from data, albeit with higher computational demands. Future work can focus on merging the strengths of these models—leveraging Haar features for efficient localization and CNNs for

accurate classification—to create hybrid systems optimized for both performance and real-time deployment.
By addressing dataset limitations, optimizing model architectures, and exploring advanced techniques, these models have the potential to serve as robust solutions for face detection and recognition across various applications, including surveillance, human-computer interaction, and mobile deployment.

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.