Face Detection Technical Assignment report

Nima Daryabar
nima.daryabar@gmail.com

Abstract

This report presents the development of a real-time facial detection and smile classification system using a dataset created from user-captured images. The project involves capturing real-time video frames, detecting faces using OpenCV's pre-trained classifiers, and classifying detected faces as "Smiling" or "Not Smiling" using a deep learning model. A MobileNetV2-based binary classifier was trained for smile classification, leveraging offline data augmentation (by saving augmented images into corresponding datasets) and real-time augmentation during training to improve model performance. Various accuracy metrics were calculated to assess performance. The smile detection system was implemented in Python, containerized using Docker, and integrated with Bash scripting for seamless execution. The pretrained model was initially trained with all layers frozen. Later, half of the model's layers were unfrozen and fine-tuned, and the accuracy of this fine-tuned model was compared against the initial frozen-layer model. Evaluation results demonstrated moderate real-time performance and classification accuracy, primarily due to the small-scale model, dataset limitations, and the need for improved hyperparameter tuning. This report provides a detailed overview of the dataset collection, model training, performance evaluation, and real-time system integration.

1. Introduction

Face detection is a computer vision technology that identifies human faces in digital images and videos. It works as a crucial first step in various applications, including facial recognition systems, photography, and human-computer interaction. Face detection can be considered a specific case of object-class detection, where the goal is to locate and distinguish human faces from other objects like buildings or trees. While early algorithms focused on detecting frontal faces, modern techniques aim to handle variations in facial orientation, lighting, occlusions, and expressions [6].

1.1. Applications of Face Detection

- **photography**: Many digital cameras use face detection for autofocus, ensuring that faces remain in sharp focus [6].
- **Marketing**: Retailers utilize face detection to gather demographic data (age, gender) and deliver targeted advertisements [6].
- **Emotional Inference**: AI-powered systems analyze facial expressions to infer emotions, help in mental health assessments and human-computer interaction [6].

1.2. Algorithms

A notable algorithm for face detection is the Viola–Jones object detection framework, which utilizes Haar Cascade Classifiers for real-time face detection [7]. This approach is based on:

- Haar-like Features: These features compare pixel intensity differences to identify facial structures like eyes, noses, and edges [7].
- Integral Images: Enables rapid calculation of feature sums to improve detection efficiency [6].
- AdaBoost Training: Selects the most relevant features to distinguish between faces and non-faces [7].
- Cascade of Classifiers: Speeds up detection by filtering out non-face regions in early stages [7].

Haar Cascade Classifiers are widely used due to their efficiency and real-time capabilities, but they struggle with extreme variations in lighting, occlusions, and facial angles. Modern deep learning approaches, such as CNN-based face detectors, have largely outperformed Haar cascades in robustness and accuracy. However, due to its low computational cost, Haar cascade remains a preferred choice for embedded and resource-limited applications [7], [4]. Despite advancements, face detection still faces challenges such as handling low-light conditions, occlusions (e.g., glasses, masks), and diverse facial orientations. Research in computer vision continues to improve detection

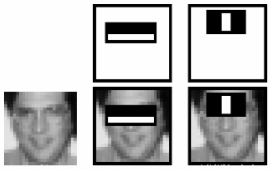

Figure 1. Haar feature types [4].

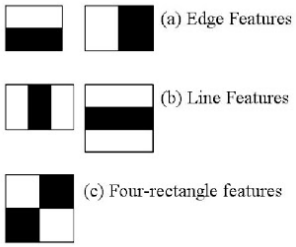

Figure 2. Haar cascade face detection algorithm [4].

accuracy through deep learning and adaptive AI models [7], [4]. Figure 1 presents different types of Haar-like features and Figure 2 demonstrates how Haar-like features are applied to an image.

1.3. Project Workflow and Technologies Used

In this project, we implemented a real-time face detection and smile classification system using the OpenCV and TensorFlow frameworks. The Haar Cascade Classifier was selected for face detection, which provides a balance between speed and accuracy for real-time applications. To enhance performance, multi-threading is used to separate video capture, frame processing, and face detection into different threads, optimizing execution speed.

Once the face detection pipeline is established, a custom dataset of smiling and non-smiling faces is created by capturing and saving images. To improve model generalization, data augmentation techniques are applied, which will be further discussed in the Dataset section.

For smile classification, a lightweight deep learning model using MobileNetV2 is trained, due to its efficiency and low computational cost. Initially, the model was trained with all layers frozen except the classifier layers, followed by fine-tuning half of the layers to improve accuracy. This approach helped optimize the model's weights for better detection of smiling faces in real-time.

Finally, the trained model is integrated into the face detection system, enabling live classification of detected faces as "Smiling" or "Not Smiling". To ensure portability and ease of deployment, we containerized the entire application using Docker. The project was maintained using GitHub version control in a private repository [1], where all implementations, code updates, and version tracking were actively managed.

The following sections will further discuss the technologies used, dataset creation, and model performance evaluation.

1.4. Dataset

The dataset was specifically created to support a binary classification task distinguishing between smiling and non-smiling faces. It consists of images collected in real-time from a user's webcam.

1.5. Face Detection and Collection Pipeline

Data collection was performed using a real-time face detection system built with OpenCV. The video stream from the user's webcam was captured and processed using a multithreaded implementation to enhance performance and efficiency:

- **Video Capture**: Implemented via OpenCV's VideoCapture method, the script automatically searches and selects the first available webcam device.
- **Frame Processing**: Captured frames were resized to (300x300) pixels, converted to grayscale, equalized to enhance contrast using cv2.equalizeHist, and noise-reduced with Gaussian blur (cv2.GaussianBlur). These preprocessing steps help standardize image quality, reduce computational load, and improve face detection accuracy.
- **Face Detection (Haar Cascade Classifier)**: Faces within processed frames were detected using OpenCV's pre-trained Haar Cascade Classifier (haarcascade_frontalface_default.xml). Detected face coordinates were scaled back to their original frame dimensions for accurate cropping.

This multithreaded approach tries to distribute the workload to enhance frame rate and overall system responsiveness.

1.6. User Interaction for Labeling

User interaction played an important role in labeling the dataset. Detected faces were presented to users in real-time, allowing them to manually categorize each detected face into two distinct classes: "smile" or "nosmile". Keyboard commands ('s' for smiling and 'a' for non-smiling) simplified this labeling task, and the labels were concurrently saved alongside images in a CSV file for future reference.

1.7. Dataset Organization and Splitting

Following data collection, images were systematically organized and partitioned into training, validation, and testing subsets. This structured approach facilitated efficient model training and reliable evaluation.

Figure 3. Frame categorized as smile.

Figure 4. Frame categorized as non-smile.

- **Dataset Organization**: The images were then stored in distinct directories according to their class labels (”smile” or ”nosmile”). Separate directories were created programmatically to maintain clarity and ease of access.
- **Data Splitting**: To robustly evaluate the model performance and prevent overfitting, the collected images were divided using predefined ratios:
  - (60%) for training.
  - (20%) for validation.
  - (20%) for testing.

  Random shuffling of images ensured unbiased distribution among subsets. Figure 3 presents a frame categorized as smiling face, and Figure 4 depicts a frame taken as a non-smile face.

1.8. Data Augmentation

To enhance dataset diversity and mitigate overfitting, data augmentation techniques were employed using TensorFlow and Keras layers:

For this reasons, we utilized the following techniques:

- Random Flipping (Horizontal and Vertical): Added variability in face orientation.
- Random Rotation (±10%): Simulated slight head tilting.
- Random Zoom (±10%): Varied face proximity.
- Random Contrast Adjustment (±10%): Simulated variable lighting conditions

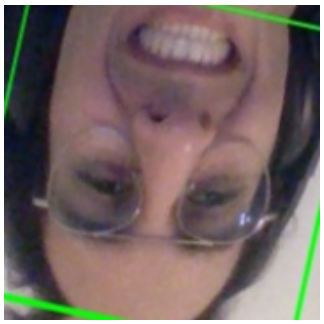

Figure 5. Horizontal and vertical flip augmented frame in smile training set.

Figure 6. Contrast adjustment Augmented frame in non-smile training set.

It is important to mention that 50% of the collected images were randomly selected for augmentation. Selected images underwent resizing (to 180x180 pixels) followed by the augmentation transformations mentioned above. Augmented images were subsequently saved into the training dataset directory, enriching the dataset for improved model robustness and generalization. Figure 5, and Figure 6 show horizontal and vertical flip, and contrast adjustment augmentations respectively.

1.9. Technologies and Tools Employed

The following technologies and frameworks were used to create and augment the dataset efficiently:

- **OpenCV**: To perform face detection using Haar Cascade classifiers and image preprocessing operations such as grayscale conversion, histogram equalization, and Gaussian blur.
- **TensorFlow/Keras**: For sophisticated data augmentation leveraging built-in image augmentation layers (RandomFlip, RandomRotation, RandomZoom, RandomContrast).

This comprehensive approach to dataset creation and augmentation improved the development of a robust image classification model tailored specifically to distinguish smiling from non-smiling faces, improving the model’s accuracy and generalization capacity.

2. Methodology

The primary goal of this project was to develop a modular and interactive application enabling users to choose functionalities such as face detection, data augmentation, and model evaluation alongside other options to proceed with the application of a face detection system. This design ensures ease of interaction and modularity, allowing future scalability and code reuse across different modules.

2.1. Face Detection Approach

Face detection was implemented using OpenCV in Python, with two distinct methods:

- **Simple Face Detection (face_detection.py)**, which includes the following workflow:
  - **Video Capture**: Utilized OpenCV's VideoCapture() method to access the webcam.
  - **Preprocessing**: Each frame was resized to (300x300) pixels, converted to grayscale, enhanced for contrast using cv2.equalizeHist, and smoothed with Gaussian Blur (cv2.GaussianBlur) to reduce noise and improve detection accuracy.
  - **Detection Technique**: Haar Cascade classifier (haarcascade_frontalface_default.xml), selected for its speed and lightweight characteristics despite limitations in detecting faces at extreme angles.
  - **Scaling Detected Faces**: Faces detected in the resized frames were rescaled back to original dimensions by applying the scaling factor:
    * The original frame has a width of W and a height of H.
    * The resized frame used for face detection is 300×300.
    * To scale back, we calculate:

$$\text{scale}_x = \frac{W}{300}, \quad \text{scale}_y = \frac{H}{300},$$

These tell us how much we need to stretch the coordinates to match the original size. Finally, we rescale each detected face's bounding box by the following operations:

· Each detected face has a bounding box (x, y, w, h) in the resized frame.

· To convert it back to the original frame size we calculate these operations:

$$x' = x \times \text{scale}_x, \quad y' = y \times \text{scale}_y$$

$$w' = w \times \text{scale}_x, \quad h' = h \times \text{scale}_y$$

This ensures the bounding boxes are in the correct position in the original frame.

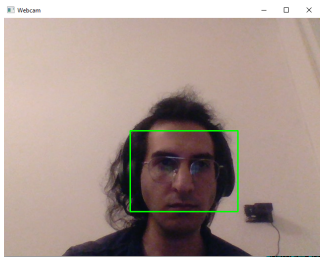

Figure 7. Detected face by drawing a bounding box.

Figure 7 shows a simple prediction by implementing Haar cascade classification using OpenCV.

- Multithreaded Face Detection (face_detection_multi_threading.py): As previously discussed in the dataset section, we briefly addressed the implementation procedure. It utilizes Python's threading module to improve the real-time responsiveness of the system by running three threads in parallel:
  - Thread 1: Captures video frames from the webcam.
  - Thread 2: Performs preprocessing (resize, grayscale conversion, noise reduction).
  - Thread 3: Detects faces using the Haar Cascade classifier. This multithreading ensures continuous, uninterrupted operation even with resource-intensive processing tasks.

2.2. Model Training

Given the goal of deploying the application on edge devices, the model selection emphasized efficiency and size without significantly compromising accuracy. Therefore, lightweight CNN architectures, particularly MobileNetV2 [3] and EfficientNet (B0-B7) [2], were considered. Ultimately, MobileNetV2 was selected due to its favorable accuracy-to-performance trade-off suitable for resource-constrained environments.

2.3. MobileNetV2 Architecture

Sandler et al. introduced MobileNetV2, a lightweight deep learning model optimized for mobile and embedded devices. It builds on MobileNetV1 by incorporating inverted residual blocks with linear bottlenecks, improving efficiency while preserving accuracy. Each block expands, processes, and compresses features, reducing computation

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Figure 8. MobileNetV2 layers' structure summary [5].

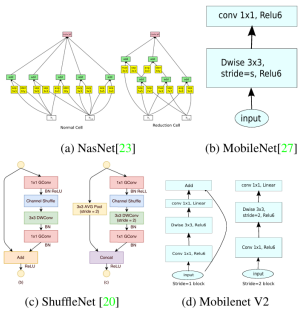

Figure 9. Convolutional block comparison summary of various architectures [5].

without losing important information. Depthwise separable convolutions further enhance efficiency. Key parameters include the width multiplier and input resolution (e.g., 224x224). MobileNetV2 reduces operations by 75% compared to standard CNNs while maintaining competitive accuracy, making it ideal for object detection, segmentation, and real-time AI applications in resource-constrained environments [5]. Figure 8 demonstrates the summary over MobileNetV2 layers' structure, and Figure 9 presents the comparison of different architectures compared to MobileNetV2's architecture.

2.4. Training Procedure

: We implemented the training stage in TensorFlow and Keras frameworks. Input Dimension are set to 224x224 pixels. Batch size is set to 32 and Adam optimizer is selected as the optimizer algorithm. Learning rate is set to start at 1e-4, and reduced to 1e-5 during fine-tuning. We run training and fine-tuning each for 10 epochs. We initially train by freezing all the layers excluding the classifier layers. Followed by fine-tuning, since the model is built with 154 layers, 77 out of 154 layers are unfrozen to adjust their weights, to enhance the model's generalization.

It is important to mention that data augmentation is applied during the training to prevent overfitting and improve the model accuracy and the techniques employed included:

- Apply random crop to each image of the batch
- Randomly flip images horizontally
- Randomly adjust brightness of images

Metrics	Value
Accuracy	0.5304
Precision	0.5135
Recall	1.0000
F1-score	0.6786

Table 1. Initial model training Results.

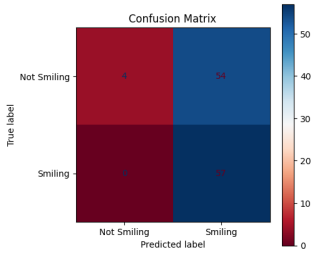

Figure 10. Initial model training confusion matrix.

- Randomly adjust the contrast of images
- Randomly adjust the hue of images
- Randomly adjust the saturation of images

The trained MobileNetV2 model was then integrated into a real-time customized face detection pipeline (customized_face_detection.py). Detected faces were classified as "Smiling" or "Not Smiling," indicated visually by green and orange bounding boxes, respectively.

3. Results

The performance evaluation of the trained models (both initial and fine-tuned) was conducted using the test dataset. The accuracy metrcis in the evaluation process included: **Accuracy**, **Precision**, **Recall**, and **F1-score**. Confusion matrices is plotted for visual evaluation of model performance and better understanding of the model's prediction accuracy on the test dataset.

3.1. Initial Model training Evaluation

: Freezing the model layers and training the model's classifier resulted in (53%) in accuracy, which is considered as moderate accuracy, but F1 scores show a slightly better result (0.67). It is of interest that recall is (1.00) with precision around (0.51). Table 1 represents the results for the discussed initial model training.

The Initial model training confusion matrix shows that the model heavily favors predicting "Smiling." Out of 58 "Not Smiling" images, only 4 are correctly classified, while 54 are misclassified as "Smiling," and all 57 "Smiling" images are correctly predicted. Figure 10 represents the obtained results.

Metrics	Value
Accuracy	0.5385
Precision	0.5385
Recall	1.0000
F1-score	0.7000

Table 2. Fine-Tuned model Results.

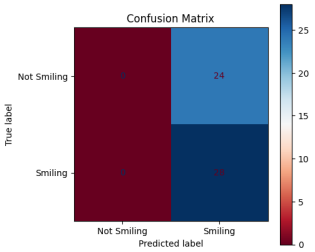

Figure 11. Fine-tuned model confusion matrix.

3.2. Fine-Tuned Model Evaluation

After unfreezing half the layers of the model and fine-tuning it for another 10 epochs, the model accuracy does not show a significant increase and it shows almost the same value as the initial training accuracy, (53.8%). The precision is slightly improved by 0.02, and the recall, like the initial model training, shows the value of (1.00). F1-score has slightly improved and reached to (0.70). Table 2 demonstrates the corresponding results obtained from the fine-tuning stage.

confusion matrix obtained from the evaluation of the fine-tuned model shows the model's bias toward "Smiling" is even stronger: it never correctly predicts "Not Smiling," labeling all 24 such images as "Smiling," while correctly classifying all 28 "Smiling" images. This suggests issues like class imbalance or insufficient features for the "Not Smiling" class. Figure 11 illustrates the discussed confusion matrix.

4. Discussion

Results indicate relatively moderate accuracy, mainly due to the limited size of the collected dataset. Fine-tuning the model shows the same accuracy as the initial model training, but overall performance remained constrained by dataset scale and limited feature extraction capacity. The confusion matrices show that the model is unable to catch the overall pattern as shown in Figure 12, and Figure 13, and the model classifies both smiling and non-smiling images with a smiling label.

5. Conclusion and Future Work

Based on these results, future developments will focus on enhancing detection by implementing OpenCV's DNN face detection module for greater robustness, particularly for faces at extreme angles. Additionally, integrating advanced deep learning models, such as transformer-based architectures with self-attention mechanisms, will be explored to improve classification accuracy. Feature extraction will be refined using advanced methods like Scale-Invariant Feature Transform (SIFT) to obtain more robust features. Moreover, performance optimization will be prioritized by re-implementing critical components, particularly real-time face detection, in C++, which could significantly enhance processing speed and improve suitability for real-world deployment on edge devices. Leveraging multiprocessing in Python could also improve the overall performance.

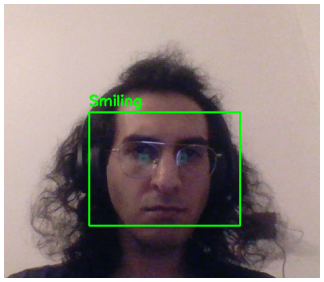

Figure 12. Non-smiling face classified as smiling face by integrated classification model.

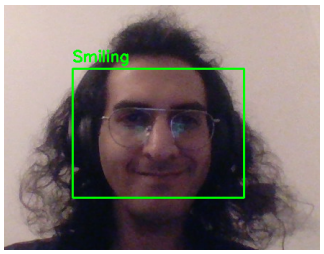

Figure 13. smiling face classified as smiling face by integrated classification model.

References

[1] Nima Daryabar. Face detection github repository. https://github.com/nimad70/cv$_{a}ssignment$.

[2] keras.io. Efficientnet b0 to b7. https://keras.io/api/applications/efficientnet/.

[3] keras.io. Mobilenet, mobilenetv2, and mobilenetv3. https://keras.io/api/applications/mobilenet/.

[4] OpenCV. Opencv: Face detection using haar cascades. https://docs.opencv.org/4.x/d2/d99/tutorial$_{j}s_{f}ace_{d}etection.html$.

[5] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks, 2019.

[6] Wikipedia. Face detection. https://en.wikipedia.org/wiki/Face$_{d}etection$.

[7] Wikipedia. Viola–jones object detection framework. https://en.wikipedia.org/wiki/Viola