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 accuracy through deep learning and adaptive AI models [7], [4].

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.

- **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.

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

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.

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:

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

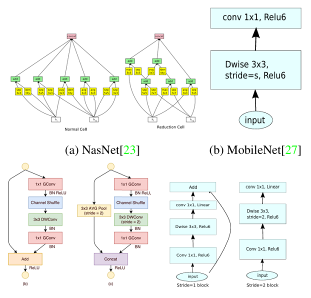
$$w' = w \times \text{scale}_x, \quad h' = h \times \text{scale}_y$$

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

- 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).

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 1. MobileNetV2 layers' structure summary



(a) NasNet[23] (b) MobileNet[27]

(c) ShuffleNet [20] (d) Mobilenet V2

Figure 2. Convolutional block comparison summary of various architectures

– 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 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 1 demonstrates the summary over MobileNetV2 layers' structure, and Figure 2 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
- 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

4. Future works

we can use multi-processing

4.1. Methods

The following is a suggested structure for your report:

- Introduction (10%): describe the problem you are working on, why it's important, and an overview of your results.
- Related Work (10%): discuss published work or similar apps that relates to your project. How is your approach similar or different from others?
- Dataset (15%): describe the data you are working with for your project. What type of data is it? Where did it come from? How much data are you working with? Did you have to do any preprocessing, filtering, etc., and why?

- Method (30%): discuss your approach for solving the problems that you set up in the introduction. Why is your approach the right thing to do? Did you consider alternative approaches? It may be helpful to include figures, diagrams, or tables to describe your method or compare it with others.
- Experiments (30%): discuss the experiments that you performed. The exact experiments will vary depending on the project, but you might compare with prior work, perform an ablation study to determine the impact of various components of your system, experiment with different hyperparameters or architectural choices. You should include graphs, tables, or other figures to illustrate your experimental results.
- Conclusion (5%): summarize your key results; what have you learned? Suggest ideas for future extensions.

5. Formatting your paper

All text must be in a two-column format. The total allowable width of the text area is $6\frac{7}{8}$ inches (17.5 cm) wide by $8\frac{7}{8}$ inches (22.54 cm) high. Columns are to be $3\frac{1}{4}$ inches (8.25 cm) wide, with a $\frac{5}{16}$ inch (0.8 cm) space between them. The main title (on the first page) should begin 1.0 inch (2.54 cm) from the top edge of the page. The second and following pages should begin 1.0 inch (2.54 cm) from the top edge. On all pages, the bottom margin should be 1-1/8 inches (2.86 cm) from the bottom edge of the page for 8.5 $\times$ 11-inch paper; for A4 paper, approximately 1-5/8 inches (4.13 cm) from the bottom edge of the page.

5.1. Margins and page numbering

All printed material, including text, illustrations, and charts, must be kept within a print area 6-7/8 inches (17.5 cm) wide by 8-7/8 inches (22.54 cm) high. Page numbers should be in footer with page numbers, centered and .75 inches from the bottom of the page and make it start at the correct page number rather than the 4321 in the example. To do this fine the line (around line 23)

```
%\ifcvprfinal\pagestyle{empty}\fi
\setcounter{page}{4321}
```

where the number 4321 is your assigned starting page.

Make sure the first page is numbered by commenting out the first page being empty on line 46

```
%\thispagestyle{empty}
```

5.2. Type-style and fonts

Wherever Times is specified, Times Roman may also be used. If neither is available on your word processor, please use the font closest in appearance to Times to which you have access.

MAIN TITLE. Center the title 1-3/8 inches (3.49 cm) from the top edge of the first page. The title should be in Times 14-point, boldface type. Capitalize the first letter of nouns, pronouns, verbs, adjectives, and adverbs; do not capitalize articles, coordinate conjunctions, or prepositions (unless the title begins with such a word). Leave two blank lines after the title.

AUTHOR NAME(s) and AFFILIATION(s) are to be centered beneath the title and printed in Times 12-point, non-boldface type. This information is to be followed by two blank lines.

The ABSTRACT and MAIN TEXT are to be in a two-column format.

MAIN TEXT. Type main text in 10-point Times, single-spaced. Do NOT use double-spacing. All paragraphs should be indented 1 pica (approx. 1/6 inch or 0.422 cm). Make sure your text is fully justified—that is, flush left and flush right. Please do not place any additional blank lines between paragraphs.

Figure and table captions should be 9-point Roman type as in Table 1. Short captions should be centred.
Callouts should be 9-point Helvetica, non-boldface type. Initially capitalize only the first word of section titles and first-, second-, and third-order headings.

FIRST-ORDER HEADINGS. (For example, **1. Introduction**) should be Times 12-point boldface, initially capitalized, flush left, with one blank line before, and one blank line after.

SECOND-ORDER HEADINGS. (For example, **1.1. Database elements**) should be Times 11-point boldface, initially capitalized, flush left, with one blank line before, and one after. If you require a third-order heading (we discourage it), use 10-point Times, boldface, initially capitalized, flush left, preceded by one blank line, followed by a period and your text on the same line.

5.3. Footnotes

Please use footnotes[1] sparingly. Indeed, try to avoid footnotes altogether and include necessary peripheral observations in the text (within parentheses, if you prefer, as in this sentence). If you wish to use a footnote, place it at the bottom of the column on the page on which it is referenced. Use Times 8-point type, single-spaced.

5.4. References

List and number all bibliographical references in 9-point Times, single-spaced, at the end of your paper. When referenced in the text, enclose the citation number in square brackets, for example [**?**]. Where appropriate, include the name(s) of editors of referenced books.

[1]This is what a footnote looks like. It often distracts the reader from the main flow of the argument.

Method	Frobnability
Theirs	Frumpy
Yours	Frobbly
Ours	Makes one's heart Frob

Table 1. Results. Ours is better.

5.5. Illustrations, graphs, and photographs

All graphics should be centered. Please ensure that any point you wish to make is resolvable in a printed copy of the paper. Resize fonts in figures to match the font in the body text, and choose line widths which render effectively in print. Many readers (and reviewers), even of an electronic copy, will choose to print your paper in order to read it. You cannot insist that they do otherwise, and therefore must not assume that they can zoom in to see tiny details on a graphic.

When placing figures in LaTeX, it's almost always best to use `\includegraphics`, and to specify the figure width as a multiple of the line width as in the example below

```
\usepackage[dvips]{graphicx} ...
\includegraphics[width=0.8\linewidth]
                {myfile.eps}
```

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