Glasses Detection from Human Face Images.

Kajal Lochab
Computer Science and Engineering
Nirma University
Ahmedabad, India
21bce103@nirmauni.ac.in

Lakshin Pathak
Computer Science and Engineering
Nirma University
Ahmedabad, India
21bce135@nirmauni.ac.in

***Abstract*—This study uses the MobileNet architecture to provide a novel approach for identifying glasses in photos of people's faces. The goal of this effort is to correctly identify glasses in face photographs for a variety of uses, including virtual try-on apps, driver monitoring systems, and facial recognition systems. Our study offers a powerful transfer learning-based glasses identification model utilizing the MobileNet architecture. We analyze the issue formulation in detail, taking into account the evaluation metrics, optimization objectives, and mathematical framework. The data, intelligence, and application layers in our suggested architecture are tailored for effective glasses detection. By means of comprehensive testing and analysis, we exhibit the efficacy of our methodology in precisely identifying spectacles in photographs of human faces. The outcomes and conversations demonstrate how well our approach performs in various circumstances and assessment parameters. This study offers insightful information for creating efficient glasses detection algorithms that may be used in a variety of real-world contexts.**

***Index Terms*—Glasses detection, Human face images, Image processing, Transfer learning, Facial feature extraction**

I. Introduction

Facial recognition technology is now widely used in many different industries for things like verification systems, security measures, and human-computer interface. The ability to precisely recognize and assess facial traits to differentiate individuals is at the heart of these technologies. Given that spectacles can significantly alter one's face appearance, identifying the existence of eyeglasses presents a substantial problem in this field [1]. The variety in the form, style, and capacity of eyeglasses to partially obfuscate face characteristics makes it more difficult to identify in photos.

The development of deep learning methodologies has led to significant developments in multiple computer vision domains, such as object detection and categorization. In particular, transfer learning has become a potent method for using pre-existing models on novel, at times data-poor problem areas. The objective is to implement transfer learning [2] to develop an accurate and credible model that can identify eyewear in photos of faces.

Using the MobileNet architecture, we offer a new method in our research for the detection of eyeglasses. By correctly recognizing eyewear, this technique aims to improve the accuracy of facial [3] recognition systems, hence increasing their efficiency and improving user experience.

A. Motivation

Accurately identifying eyeglasses in facial photos is essential for a number of applications, including virtual try-on platforms, increased security via facial recognition systems, and the efficacy of driver vigilance programs. Because eyewear modifies face features and creates reflections and distortions, it presents special difficulties that could affect the accuracy of facial recognition technology. Similarly, precise eyewear detection is necessary for virtual fitting services in order to provide a smooth and lifelike online fitting experience. Determining if a driver is wearing eyeglasses can yield important information about their concentration and alertness levels in driver surveillance applications.

However, given the variety of eyewear shapes, sizes, and ways in which they cover different portions of the face, the process of eyewear identification poses significant challenges. Conventional image processing methods frequently find it difficult to properly account for these factors. Therefore, the creation of complex algorithms that can reliably identify eyewear in a variety of real-life circumstances is urgently needed.

B. Research Contribution

In this work, we present a novel approach to eyeglass recognition by utilizing the MobileNet architecture. MobileNet, which is well-known for its effectiveness, provides a solid foundation for our picture categorization problems. By using transfer learning [2], we are able to effectively modify a pre-trained MobileNet model to perform well in the complex task of eyewear detection, obtaining notable accuracy even with a small amount of training data.

Our research delves deeper into the theoretical underpinnings of this problem, outlining the underlying mathematics, optimization goals, and assessment standards. We take great care in crafting the assessment criteria and loss function to make sure our model can accurately identify minute details about eyewear in facial photos.

In addition, we provide a custom architecture that consists of layers for data pretreatment, analysis, and prediction that are created especially for this task. While the analysis layers use MobileNet's complex feature hierarchies to detect eyeglasses, the preparation layers prepare the photos for feature extraction. The final determination of whether or not eyewear is present in the examined photos is made by the prediction layers.

C. Organization

In this paper, we present a novel approach to recognize eyeglasses with the MobileNet framework. This work presents a robust glasses detection model within the MobileNet system by applying transfer learning concepts [2]. We provide a thorough analysis of the problem, including its mathematical foundations, optimization objectives, and evaluation metrics. In addition, we create a tailored framework with layers for practical application, cognitive layers, and data processing layers in order to improve the precision of eyewear recognition.

II. Background

One of the most important applications of computer vision is object identification in photos, which has several applications ranging from object recognition to scene comprehension to image classification. Historically, the discipline has relied heavily on standard machine learning approaches and manually constructed features, which unfortunately have difficulty adjusting to a large range of complex object classes. But the emergence of deep learning has revolutionized computer vision, yielding exceptional performance on a wide range of tasks, including object recognition. Convolutional Neural Networks (CNNs) [4] are one of these innovations that have gained significant traction as the mainstay of technology for several popular object detection frameworks. These networks are particularly good in learning multi-level visual data representations on their own, which makes them very useful for challenging tasks like object detection.

Furthermore, transfer learning [4] has become a key technique in deep learning, allowing models that have been pre-trained on large datasets to be applied to new problems with limited data. This strategy makes use of prior knowledge to accelerate learning and improve task performance. One streamlined CNN design that stands out for its ease of implementation on mobile and embedded platforms is MobileNet [5]. By using depthwise separable convolutions, it is easy to maintain robust accuracy with a large reduction in the number of required parameters and computing demands. Due to its small size, low processing overhead, and remarkable accuracy, MobileNet is a widely used option for a wide range of applications. MobileNet [5] serves as a potent foundation for feature extraction in object detection scenarios, providing comprehensive visual data representations helpful for identifying particular things, such as identifying eyeglasses on human faces. This ability to customize pre-trained MobileNet models for particular detection tasks highlights its versatility and effectiveness.

III. Related Work

TABLE I
State of the Art in Glasses Detection

Year	Title	Model Used	Pros	Cons	Accuracy (%)
2000	[4]	CNN	The deformable contour can be achieved through dynamic programming.	The crossing of eyebrows on the edges of glasses creates significant ambiguities.	99.52%
2004	[6]	Wavelet Based LUT Weak Classifier	High level of correctness and a fast running speed.	Lower accuracy compared to other models	95.5%
2013	[7]	SVM,GB,GNB	Achieved a suitable balance between precision and speed	Collecting training dataset is cumbersome	96%

IV. Problem Formulation

A. Mathematical Framework

1) Input Representation: The RGB photos displaying people's faces make up the raw data for the task of identifying glasses on human faces inside images. Every image is organized as a three-layer matrix, with each layer representing the intensity of a pixel throughout its width, height, and the three fundamental hues of red, green, and blue.

These photographs go through a number of preparation procedures before being analyzed using the MobileNet architecture. To increase the robustness of the model, this entails resizing the photographs to a consistent size, scaling the pixel intensity values to lie between 0 and 1, and applying image modification techniques like spinning and mirroring the images.

$$\mathbf{x} = [x_{\text{width}}, x_{\text{height}}, x_{\text{channel}}] \tag{1}$$

2) Transfer Learning: When data is scarce, utilizing transfer learning is a useful tactic for adapting current models to new tasks. In particular, this method involves adapting a previously trained MobileNet model to operate with a fresh dataset consisting of pictures of human faces that are labeled

as either wearing glasses or not in order to recognize eyewear on faces.

Using the original weights from the MobileNet model—obtained from its training on a large-scale dataset such as ImageNet—we commence this refinement process. Then, in order to minimize the cross-entropy loss, optimization techniques based on gradient descent are applied to modify the model's parameters. The difference between the actual classifications in the training dataset and the projected likelihoods of the model is measured by this loss.

$$\min_{\theta} \mathcal{L}(\theta) = \sum_{i=1}^{N} \text{CE}(y_i, f_{\text{MobileNet}}(x_i; \theta)), \tag{2}$$

3) Output Generation: The output generation procedure involves using the capabilities of the MobileNet architecture to make predictions about the identification of eyeglasses on faces in image data. The final output is a simple binary classification that determines whether or not the examined picture shows a person wearing spectacles.

After processing the image using the best-fit MobileNet framework, this model uses a softmax activation function to provide a probability distribution across the two categories (glasses presence or absence). The anticipated categorization is then determined by identifying the category that has the highest likelihood.

B. Optimization Objective

Reducing a loss function, which quantifies the discrepancy between the actual labels of the training data and the probabilities predicted by the model, is the aim of the transfer learning process using MobileNet to identify glasses. Usually, the cross-entropy loss is used as the loss function for this, and it is defined as follows:

$$\mathcal{L}(\theta) = -\frac{1}{N} \sum_{i=1}^{N} [y_i \cdot \log(M_\theta(x_i)) + (1 - y_i) \cdot \log(1 - M_\theta(x_i))], \tag{3}$$

where θ represents the parameters of the MobileNet model, N is the number of training samples, x_i is the i-th input image, y_i is the ground truth label (0 for no glasses, 1 for glasses), and $M_\theta(x_i)$ is the output of the MobileNet model for the i-th input image with parameters θ. By assessing the discrepancy between the ground truth labels and the expected probabilities, the cross-entropy loss penalizes the model for making inaccurate predictions.

C. Evaluation Metrics

A few critical metrics are essential for gauging the effectiveness of machine learning algorithms. Two crucial measures are used in the evaluation of the code in question: accuracy and binary cross-entropy loss. The fraction of all correctly predicted predictions is represented by the accuracy measure, which is calculated using the following formula:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

The difference between actual binary outcomes and their expected probability is measured by binary cross-entropy loss.

V. Proposed Architecture

A. Data Layer

Our proposed framework's base layer is essential for gathering and preparing input photos for the purpose of recognizing eyeglasses. Our data came from a variety of high-quality photos of human faces that we obtained from Kaggle, a reputable hub for machine learning datasets and contests. This dataset includes a wide range of participants, including those who do not wear glasses, as well as those from different demographic backgrounds and with a variety of eyeglasses. We carried out a number of data cleaning and preprocessing procedures to preserve the uniformity and quality of the dataset. In order to strengthen and diversify the dataset, these stages included removing distortions, standardizing the image formats, and augmenting the dataset. We were able to access a large and detailed collection of human face photos by using the Kaggle dataset, which helped us develop a useful technique for detecting eyewear.

B. Intelligence Layer

Our intelligent framework serves as the foundation for our glasses recognition system, which is mainly designed to identify eyeglasses in pictures of faces. Advanced deep learning techniques, with a focus on convolutional neural networks (CNNs), enable this process. Through the analysis of both minor and significant patterns, these networks are skilled at differentiating particular aspects within the images, guaranteeing a high recognition accuracy rate for glasses. We use transfer learning to improve the training process, particularly when working with a little amount of labeled data. This method makes the training process more efficient by enabling us to use a pre-established set of weights to get our CNN model off to a faster start.

C. Application Layer

Our developed glasses detection model is implemented in multiple real-world scenarios at the application layer. This step entails integrating the model into various applications, such as security monitoring systems, facial recognition platforms, and even cutting-edge smart eyewear options. The model's ability to recognize glasses in real-time from still photos and live video feeds greatly improves the usability and efficacy of the applications. In addition, we are looking into ways to improve and fine-tune the model even more to make sure it satisfies the particular requirements and performance requirements of any application.

VI. Results and Discussions

A. Experiment Setup and Roles

In this section, we describe in detail the experimental setup and the roles played by each component in our glasses-detecting investigation. Creating the model, training schedule, evaluation criteria, and dataset preparation were only a few of

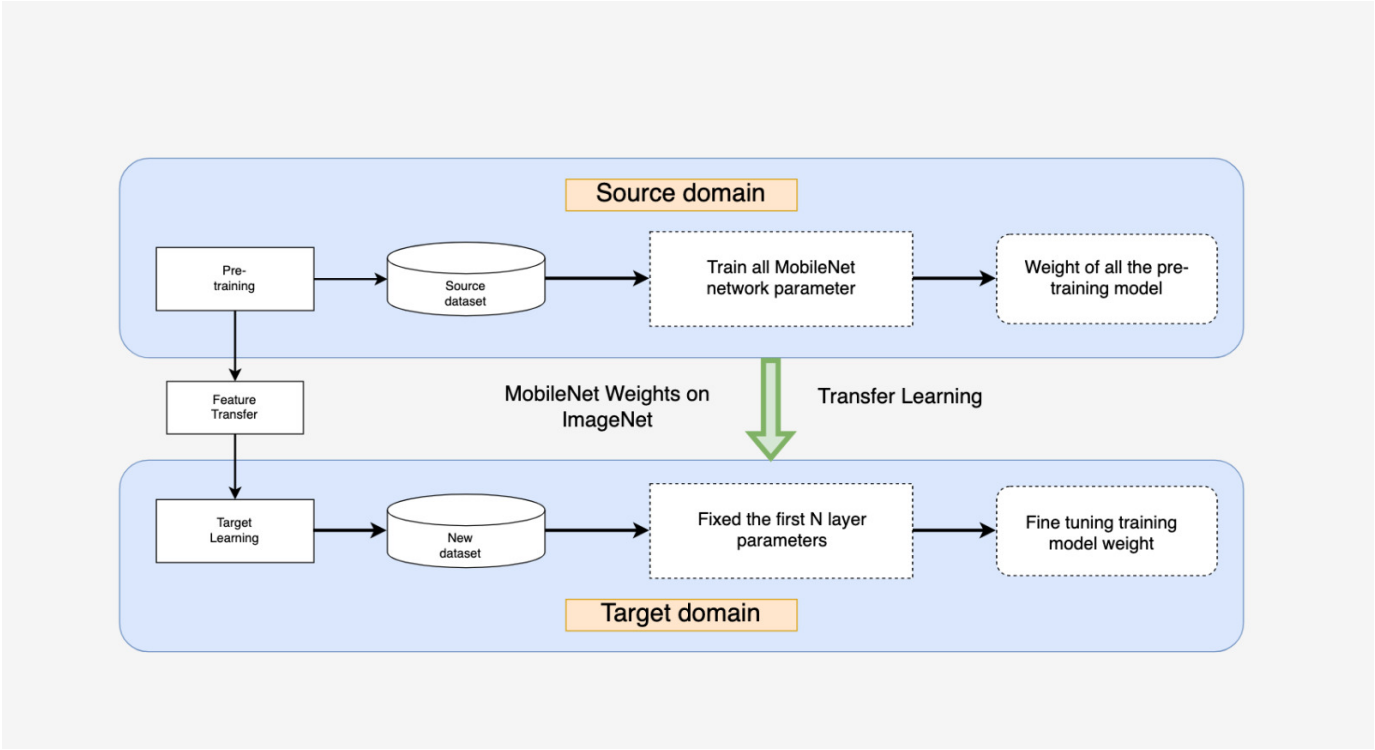

Fig. 1. Block Diagram

the crucial parts of this system. With roles given according to our responsibilities and areas of experience, our group collaborated to ensure that each step went smoothly.

Gathering and prepping data was the first step. Team members were tasked with locating human face photos in the Kaggle library and applying the appropriate cleaning and enhancing methods. Then, the intelligence layer was developed. Deep learning specialists created and implemented the convolutional neural network (CNN) model that was designed to identify eyewear.

The team worked on optimizing hyperparameters, monitoring the training progression, and modifying the model settings during the training phase. Validation and test datasets were used to assess the model's effectiveness, and team analysis identified areas for improvement. At every level, productive teamwork and collaboration were essential to achieving the project's goals.

The team roles encompassed a variety of talents from computer vision and software engineering to data science and machine learning, demonstrating diversity and synergy. Every member offered unique skills and perspectives that improved innovative problem-solving. By working together and using strategic management, we were able to conduct extensive trials, evaluate the results, and obtain important information about identifying eyewear on human faces.

B. Evaluation Metrics

In this section, we examine the metrics used to assess the eyewear identification algorithm's efficacy by contrasting its performances before and after tuning.

1) Before Fine-tuning: The model was trained using the initial environments for 10 epochs prior to fine-tuning. The following assessment metrics were obtained:

- **Test Accuracy (First 10 Epochs)**: 0.917
- **Train Accuracy (First 10 Epochs)**: 0.895

Although the model's performance could be further enhanced, its weights were initialized and modified depending on training data for a respectable level of accuracy.

2) Fine-tuning Process: To enhance the model's functionality, a procedure known as fine-tuning was implemented. The pre-trained MobileNet V2 model's top layers were unfrozen in this manner, allowing their weights to change as the model was trained. Through the application of the pre-trained network's expertise, the model was able to modify its features in order to better fit the particular goal of identifying glasses.

There were significant improvements in accuracy after the fine-tuning procedure.

- **Train Accuracy (After Fine-tuning)**: 0.999
- **Validation Accuracy (After Fine-tuning)**: 0.997

These outcomes demonstrate how fine-tuning raises the accuracy and overall performance of the model.

VII. Conclusion and Future Scope

A. Conclusion

The glasses detection model developed in this project effectively demonstrates the use of transfer learning and fine-tuning techniques in computer vision tasks. By utilizing the

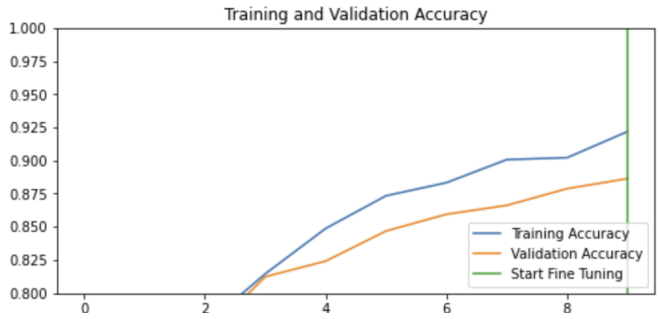

Fig. 2. Training And Validation Accuracy

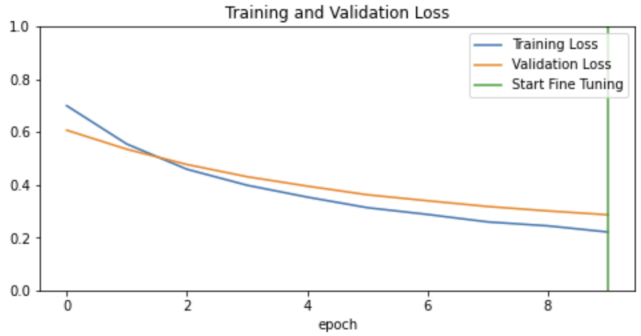

Fig. 3. Training And Validation Loss

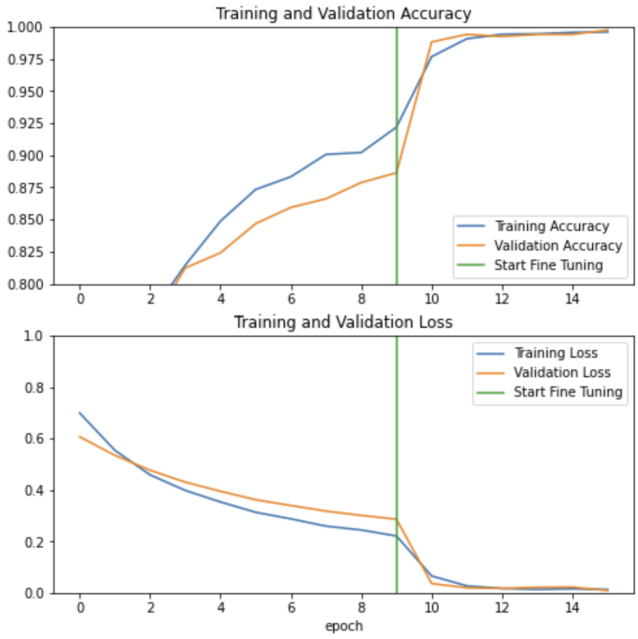

Fig. 4. After Fine Tuning

TABLE II
EXPERIMENTAL SETUP

Parameter	Value
Optimizer	Adam
Learning Rate	0.0001
Number of Epochs	15
Image Dimension	160x160
Batch Size	32
Train-Validation-Test Split Ratio	70-15-15
Loss Function	Binary Crossentropy
Performance Metric	Accuracy
Dropout	Applied (0.2)

pre-trained MobileNet [5] V2 model and adapting it for the specific task of categorizing images as either "glasses" or "no glasses", we achieved remarkable accuracy.

The initial training process showed promising results, with the model accurately distinguishing between images containing people wearing glasses and those without. However, the model's performance was further enhanced through the fine-tuning process, allowing it to refine its features and achieve near-perfect accuracy on both the training and validation datasets.

This project highlights the potential of deep learning models in solving real-world problems, such as facial recognition and attribute detection.

B. Future Scope

While the existing model does exceptionally well in identifying glasses, there are a number of avenues for more investigation and improvement.

- **Enhanced Dataset**: A greater variety of photos with different backdrops, lighting, and stances could improve the model's resilience and generalization capabilities.
- **Fine-tuning Strategies**: A variety of fine-tuning techniques, such as modifying the learning rate schedules or unfreezing different layers of the pre-trained model, can be investigated in order to maximize the model's performance.
- **Multi-Class Classification**: The model's potential applications in many domains can be increased by extending its classification capabilities to include multiple sets of photos, such as various styles of eyeglasses or facial traits.
- **Real-time Implementation**: Real-time applications such as smart glasses or surveillance systems would require the model to be optimized in terms of inference speed and resource efficiency.

By concentrating on these areas for development, further iterations of the glasses detection model can improve practicality and accuracy in real-world scenarios..

REFERENCES

[1] S. Bekhet and H. Alahmer, "A robust deep learning approach for glasses detection in non-standard facial images," *IET Biometrics*, vol. 10, no. 1, pp. 74–86, 2021.

[2] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *Journal of Big data*, vol. 3, pp. 1–40, 2016.

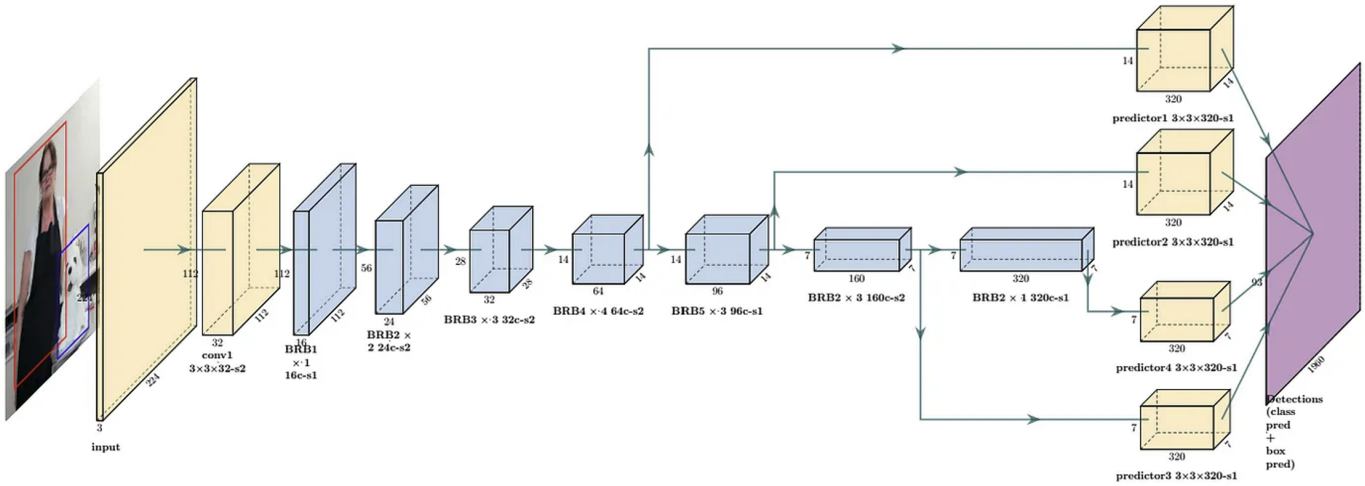

Fig. 5. MobileNet Architecture

[3] A. Fernández, R. García, R. Usamentiaga, and R. Casado, "Glasses detection on real images based on robust alignment," *Machine Vision and Applications*, vol. 26, pp. 519–531, 2015.

[4] Z. Jing and R. Mariani, "Glasses detection and extraction by deformable contour," in *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, vol. 2, pp. 933–936, IEEE, 2000.

[5] K.-Y. Kim and K.-B. Song, "Eyeball tracking and object detection in smart glasses," in *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, pp. 1799–1801, IEEE, 2020.

[6] B. Wu, H. Ai, and R. Liu, "Glasses detection by boosting simple wavelet features," in *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, vol. 1, pp. 292–295, IEEE, 2004.

[7] H. Le, T. Dang, and F. Liu, "Eye blink detection for smart glasses," in *2013 IEEE International Symposium on Multimedia*, pp. 305–308, IEEE, 2013.

APPENDIX

Here is the code used in this research:

```


import matplotlib.pyplot as plt
import numpy as np
import os
import tensorflow as tf
import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
import pickle
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import shutil
import random

# Data cleaning:
with open("../input/datacleaningglassesnoglasses/glasses.txt", "rb") as fp:
    glasses = pickle.load(fp)

plt.figure(figsize=(12, 12))
ran_num = []
for i in range(0, 9):
    n = random.randint(0, len(glasses))
    ran_num.append(n)

for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(mpimg.imread(glasses[ran_num[i]]))
    plt.title("glasses")
    plt.axis("off")

with open("../input/datacleaningglassesnoglasses/no_glasses.txt", "rb") as fp:
    no_glasses = pickle.load(fp)

plt.figure(figsize=(12, 12))
ran_num = []
for i in range(0, 9):
    n = random.randint(0, len(no_glasses))
    ran_num.append(n)

for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(mpimg.imread(no_glasses[ran_num[i]]))
    plt.title("no_glasses")
    plt.axis("off")

with open("../input/datacleaningglassesnoglasses/no_clear.txt", "rb") as fp:
    no_clear = pickle.load(fp)

plt.figure(figsize=(12, 12))
ran_num = []
for i in range(0, 9):
    n = random.randint(0, len(no_clear))
    ran_num.append(n)

for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(mpimg.imread(no_clear[ran_num[i]]))
    plt.title("no_clear")
    plt.axis("off")

print("The length of the different groups:" + "-Glasses: " + str(len(glasses)) + " -No glasses: " + str(
    len(no_glasses)) + " -No clear: " + str(len(no_clear)))

# Data processing:
BATCH_SIZE = 32
IMG_SIZE = (160, 160)

all_images = glasses + no_glasses
```

```
data_dir = "/kaggle/input/datacleaningglassesnoglasses/Images/Images/"

train_dataset = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    validation_split=0.3,
    subset="training",
    shuffle=True,
    seed=123456,
    image_size=IMG_SIZE,
    batch_size=BATCH_SIZE)

validation_dataset = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    validation_split=0.3,
    subset="validation",
    shuffle=True,
    seed=123456,
    image_size=IMG_SIZE,
    batch_size=BATCH_SIZE)

class_names = train_dataset.class_names
print(class_names)

plt.figure(figsize=(12, 12))
for images, labels in train_dataset.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(class_names[labels[i]])
        plt.axis("off")

val_batches = tf.data.experimental.cardinality(validation_dataset)
test_dataset = validation_dataset.take(val_batches // 5)
validation_dataset = validation_dataset.skip(val_batches // 5)

print('Number of training batches: %d' % tf.data.experimental.cardinality(train_dataset))
print('Number of validation batches: %d' % tf.data.experimental.cardinality(validation_dataset))
print('Number of test batches: %d' % tf.data.experimental.cardinality(test_dataset))

AUTOTUNE = tf.data.AUTOTUNE

train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)

# Preparing base model:
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
    tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
])

preprocess_input = tf.keras.applications.mobilenet_v2.preprocess_input

rescale = tf.keras.layers.experimental.preprocessing.Rescaling(1. / 127.5, offset=-1)

IMG_SHAPE = IMG_SIZE + (3,)

base_model = tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,

    include_top=False,

    weights='imagenet')

image_batch, label_batch = next(iter(train_dataset))
feature_batch = base_model(image_batch)
print(feature_batch.shape)

base_model.trainable = False

global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)

prediction_layer = tf.keras.layers.Dense(1)
prediction_batch = prediction_layer(feature_batch_average)
print(prediction_batch.shape)

inputs = tf.keras.Input(shape=(160, 160, 3))
x = data_augmentation(inputs)
x = preprocess_input(x)
x = base_model(x, training=False)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)

model.summary()

base_learning_rate = 0.0001
model.compile(optimizer=tf.keras.optimizers.Adam(lr=base_learning_rate),
              loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=['accuracy'])

print(len(model.trainable_variables))

initial_epochs = 10
loss0, accuracy0 = model.evaluate(validation_dataset)

print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(accuracy0))

model_fit = model.fit(train_dataset,
                      epochs=initial_epochs,
                      validation_data=validation_dataset)

acc = model_fit.history['accuracy']
val_acc = model_fit.history['val_accuracy']
loss_ = model_fit.history['loss']
val_loss_ = model_fit.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.ylim([0.8, 1])
plt.plot([initial_epochs - 1, initial_epochs - 1],
         plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss_, label='Training Loss')
plt.plot(val_loss_, label='Validation Loss')
```

```
plt.ylim([0, 1.0])
plt.plot([initial_epochs - 1, initial_epochs - 1],
         plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()

loss, accuracy = model.evaluate(test_dataset)
loss, accuracy1 = model.evaluate(train_dataset)
print('Test accuracy :', accuracy)
print('Train accuracy :', accuracy1)

# Model with fine tunning
base_model.trainable = True

print("Number of layers in the base model: ", len(base_model.layers))
fine_tune_at = 100
for layer in base_model.layers[:fine_tune_at]:
    layer.trainable = False

model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer=tf.keras.optimizers.RMSprop(lr=base_learning_rate / 10),
              metrics=['accuracy'])

len(model.trainable_variables)

fine_tune_epochs = 5
total_epochs = initial_epochs + fine_tune_epochs

model_fit_fine = model.fit(train_dataset,
                           epochs=total_epochs,
                           initial_epoch=model_fit.epoch[-1],
                           validation_data=validation_dataset)

acc += model_fit_fine.history['accuracy']
val_acc += model_fit_fine.history['val_accuracy']
loss_ += model_fit_fine.history['loss']
val_loss_ += model_fit_fine.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.ylim([0.8, 1])
plt.plot([initial_epochs - 1, initial_epochs - 1],
         plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss_, label='Training Loss')
plt.plot(val_loss_, label='Validation Loss')
plt.ylim([0, 1.0])
plt.plot([initial_epochs - 1, initial_epochs - 1],
         plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()

loss, accuracy = model.evaluate(test_dataset)
loss, accuracy1 = model.evaluate(train_dataset)
loss, accuracy2 = model.evaluate(validation_dataset)
print('Test accuracy :', accuracy)
print('Train accuracy :', accuracy1)
print('Validation accuracy :', accuracy2)

# Prediction
image_batch, label_batch = test_dataset.as_numpy_iterator().next()
predictions = model.predict_on_batch(image_batch).flatten()

print('Raw Predictions:\n', predictions)

predictions = tf.nn.sigmoid(predictions)

print('Raw Predictions 2:\n', predictions)

predictions = tf.where(predictions < 0.5, 0, 1)

print('Predictions:\n', predictions.numpy())
print('Labels:\n', label_batch)

plt.figure(figsize=(12, 12))
for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(image_batch[i].astype("uint8"))
    plt.title(class_names[predictions[i]])
    plt.axis("off")

# Convert to TFLite
model_dir = "/kaggle/working/model"
tf.saved_model.save(model, model_dir)

loaded = tf.saved_model.load(model_dir)
print(list(loaded.signatures.keys()))

with open('model.tflite', 'wb') as f:
    converter = tf.lite.TFLiteConverter.from_saved_model(model_dir)
    converter.optimizations = [tf.lite.Optimize.DEFAULT]
    tflite_model = converter.convert()
    f.write(tflite_model)

with open('model-full.tflite', 'wb') as f:
    converter = tf.lite.TFLiteConverter.from_saved_model(model_dir)
    tflite_model = converter.convert()
    f.write(tflite_model)

# Debug
# Load the TFLite model and allocate tensors.
interpreter = tf.lite.Interpreter(model_path="model.tflite")
interpreter.allocate_tensors()

# Get input and output tensors.
input_details = interpreter.get_input_details()
print(f"Input details: {input_details}")
output_details = interpreter.get_output_details()

input_shape = input_details[0]['shape']

# Test the model on input data.
for images, labels in train_dataset.take(1):
    for i in range(9):
        print(f"images[i].numpy()={images[i].numpy()}")
        # input_data = np.array(np.random.random_sample(input_shape), dtype=np.float32)
        input_data = [images[i].numpy().astype("float32")]
        print(f"input_data={input_data}")
        interpreter.set_tensor(input_details[0]['index'], input_data)

        interpreter.invoke()

        # The function `get_tensor()` returns a copy of the tensor data.
```

```
        # Use `tensor()` in order to get a pointer to the tensor.
        output_data = interpreter.get_tensor(output_details[0]['index'])
        print(output_data)
```