1. INTRODUCTION

1.1 Overview

Eye color detection is an important task in the field of image processing and computer vision. This project involves detecting the color of a person's eye using Convolutional Neural Networks (CNNs). Eye color, a distinguishing biometric trait, can be categorized into several types such as brown, blue, green, gray etc. By training a CNN on a dataset of labeled eye color images, we can classify the input eye images into their respective color categories with high accuracy.

1.2 Statement of the Problem

The problem is to create a machine learning model that can automatically detect and classify the eye color from images. Manual classification is time-consuming and subjective, hence an automated system is beneficial.

1.3 Motivation

This project is motivated by the growing applications of facial recognition and biometric systems in various domains including security, healthcare, and human-computer interaction. Detecting eye color accurately can support these systems.

1.4 Challenges

- Variation in lighting conditions
- Occlusions (e.g., hair, glasses)
- Low-resolution or blurry images
- Diverse eye colors with subtle differences

1.5 Applications

- Biometric authentication
- Criminal investigation
- Healthcare diagnostics
- Customized marketing
- Human-computer interaction

2. LITERATURE REVIEW

Various image processing and machine learning techniques have been employed for classification tasks such as eye color detection. Classical methods use handcrafted features like color histograms, texture descriptors, and shape analysis, while modern approaches leverage Convolutional Neural Networks (CNNs) for automated feature extraction and classification. Previous studies show that CNNs consistently outperform traditional techniques in image-based classification tasks, particularly due to their ability to learn hierarchical features directly from raw pixel data.

One notable study is **"Human Eye Color Classification Using CNN Based Deep Learning Approach"** by M. S. Bhargavi and P. Pranathi (2021). The authors designed a CNN model specifically for eye color classification, training it on a dataset of cropped eye images that included common eye color categories such as brown, blue, and green. They employed techniques like data augmentation (rotation, flipping, and scaling) to increase the diversity of training samples and improve the model's generalization capability. Their approach achieved high classification accuracy and demonstrated the robustness of deep learning models in handling variations in lighting, occlusions, and image quality commonly found in real-world images.

Another relevant work is **"Deep Learning for Eye Color Classification in the Wild"** by Jian Zhao et al. (2018). This study presents a more advanced approach that combines deep CNN architectures with domain-specific data preprocessing, such as normalization of the eye region and augmentation strategies like brightness adjustment and random cropping. By addressing the challenges of unconstrained environments — including different poses, lighting conditions, and image resolutions — the authors achieved improved performance and higher accuracy compared to baseline methods. These studies validate the suitability of deep learning, particularly CNNs, in eye color detection tasks, emphasizing their superior adaptability and performance in practical scenarios.

3. METHODOLOGY

3.1 Overview

The project follows a standard machine learning pipeline that includes data collection, data preparation, preprocessing, and model training. Both a custom Convolutional Neural Network (CNN) model and Transfer Learning techniques using pre-trained architectures are employed to improve classification performance and generalization.

3.1.1 Data Collection

A labeled dataset containing images of eyes in different colors such as brown, blue, green, and gray is used. Each image is annotated with the correct eye color category to enable supervised learning. The dataset is collected from publicly available sources and manually curated collections to ensure diversity. It covers a range of variations in lighting, pose, and image quality to better reflect real-world conditions.

3.1.2 Data Preparation

The dataset is divided into training, validation, and testing subsets to properly evaluate model performance. All images are resized to a uniform dimension to maintain consistency and match model input requirements. Normalization is applied to scale pixel values between 0 and 1. Data augmentation techniques such as rotation, flipping, and brightness changes are applied to the training set to enhance model generalization.

3.1.3 Pre-Processing

- Resizing images

- Normalizing pixel values

- Data augmentation (rotation, flipping)

3.1.4 Convolutional Neural Networks (CNNs)

CNNs are used to automatically learn and extract key features from eye images. Their layered structure is effective for image data, enabling accurate classification based on learned features.

3.1.5 Transfer Learning

To further enhance model performance, Transfer Learning is employed using pre-trained models such as VGG16 and ResNet50. These models, trained on large-scale datasets like ImageNet, are fine-tuned on the eye color dataset. This approach leverages learned features

from generic image data and adapts them to the specific task of eye color classification, improving accuracy and reducing training time.

3.2 Architecture of Proposed System

The CNN and Transfer Learning models are trained on the dataset and tested on unseen data. Results show that the models achieve high accuracy in detecting eye color, with Transfer Learning models often outperforming baseline CNNs due to their ability to leverage pre-learned visual features.

3.2.1 Input Layer

Input Shape: Images resized to a uniform shape, e.g., (128, 128, 3).

Preprocessing: Standard preprocessing (e.g., normalization, resizing).

Data Augmentation: to make the model more robust to variations in eye position and lighting.

3.2.2 Pre- trained feature extractor

Model Choices: Common backbones:

- Convolutional Neural Networks (CNNs) are used to automatically learn and extract key features from eye images.

- Transfer Learning which further enhance model performance, Transfer Learning is employed using pre-trained models such as VGG16 and ResNet50.

Layer Freezing:

- Initially freeze all layers for transfer learning.
- Optionally unfreeze top n layers for fine-tuning after initial training phase.

Why Pre- trained?

- Leverages rich feature hierarchies learned from millions of images (ImageNet).
- Reduces training time and need for a huge labeled dataset.

3.2.3 Custom Convolutional and Pooling Layers

Purpose: Adapt pre-trained features to task-specific patterns (e.g., iris texture, color gradients).

These layers act as **task-specific feature refiners**.

3.2.4 Connected Layers

- **Flatten the output** from the last convolutional layer.
- Add **Dense layers** to transform features into higher-level representations.
- **Dropout** is crucial to prevent overfitting due to limited eye datasets.

3.2.5 Output Layer

- 4 output neurons = 4 eye color classes.

- Softmax ensures outputs are class probabilities that sum to 1.

4. SYSTEM REQUIREMENTS

4.1 Hardware Requirements

- **Processor:** Intel Core i5 or higher (or equivalent AMD processor)

- **RAM:** Minimum 8 GB (16 GB recommended for faster training)

- **Storage:** At least 10 GB of free disk space

- **GPU (Optional but Recommended):** NVIDIA GPU with CUDA support for faster CNN training

- Display: 1024x768 resolution or higher

4.2 Software Requirements

- **Operating System:** Windows 10/11, Ubuntu 18.04 or higher, or macOS

- **Programming Language:** Python 3.7 or above

- **Libraries and Frameworks:**

  - TensorFlow or PyTorch (for CNN and Transfer Learning model training)

  - NumPy

  - OpenCV (for image processing)

  - scikit-learn (for traditional ML algorithms)

  - Matplotlib / Seaborn (for visualizing results)

- **IDE:** Jupyter Notebook, VS Code, or PyCharm

- **Other Tools:**

  - Anaconda (optional, for environment management)

  - Git (for version control)

5. CODING

5.1 Load and Preprocess Dataset

```
import cv2
import numpy as np
import os
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.model_selection import train_test_split

# Define eye color labels
eye_colors = ['Blue', 'Brown', 'Gray', 'Green']
image_size = (128, 128)

# Load dataset
def load_images(data_path):
    images = []
    labels = []

    for label, color in enumerate(eye_colors):
        color_path = os.path.join(data_path, color)
        if not os.path.exists(color_path):
            continue

        for file in os.listdir(color_path):
            img_path = os.path.join(color_path, file)
            img = cv2.imread(img_path)
```

```
        if img is None:
            continue

        img = cv2.resize(img, image_size)
        images.append(img)
        labels.append(label)

  return np.array(images), np.array(labels)
# Path to dataset
dataset_path = "./eye_images/train"
X, y = load_images(dataset_path)

# Normalize images
X = X / 255.0

# Split data into train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print("Dataset has been extracted and loaded successfully!")
```

5.2 Data Augmentation

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import random

def random_brightness_contrast(img):
    alpha = 1.0 + random.uniform(-0.3, 0.3)  # Contrast control
    beta = random.randint(-30, 30)           # Brightness control
    img = cv2.convertScaleAbs(img, alpha=alpha, beta=beta)
    return img

def random_gamma(img):
    gamma = random.uniform(0.7, 1.5)
    inv_gamma = 1.0 / gamma
        table = np.array([(i / 255.0) ** inv_gamma * 255 for i in np.arange(0, 256)]).astype("uint8")
    return cv2.LUT(img, table)

def coarse_dropout(img, max_holes=5, max_size=0.2):
    h, w, _ = img.shape
    img_copy = img.copy()
    for _ in range(random.randint(1, max_holes)):
        mask_h = int(random.uniform(0.05, max_size) * h)
        mask_w = int(random.uniform(0.05, max_size) * w)
        top = random.randint(0, h - mask_h)
        left = random.randint(0, w - mask_w)
        img_copy[top:top+mask_h, left:left+mask_w] = 0
```

```
  return img_copy


# Define custom augmentation function
def apply_custom_augmentations(img):
    if random.random() < 0.2:
        img = cv2.flip(img, 0)  # Vertical flip

    if random.random() < 0.7:
        img = cv2.flip(img, 1)  # Horizontal flip
    if random.random() < 0.5:
        img = coarse_dropout(img)
    if random.random() < 0.5:
        img = random_gamma(img)
    if random.random() < 1.0:
        img = random_brightness_contrast(img)
    return img

# Load the image
img_path = "./eye_images/train/blue/blue1.jpg"
img = cv2.imread(img_path)

# Check if image is loaded correctly
if img is None:
    print(f"Error: Could not load image at {img_path}")
else:
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

```
    # Apply custom augmentations
    augmented = apply_custom_augmentations(img)

    # Display images
    fig, axes = plt.subplots(1, 2, figsize=(10, 5))
    axes[0].imshow(img)
    axes[0].set_title("Original Image")
    axes[0].axis("off")

    axes[1].imshow(augmented)
    axes[1].set_title("Augmented Image")
    axes[1].axis("off")

    plt.show()
```

5.3 Build and Train model

```
# Model Architecture
from tensorflow.keras import layers, models

model = models.Sequential([
    layers.Input(shape=(128, 128, 3)),
    layers.Conv2D(32, (3, 3), activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(4, activation='softmax')  # <-- must match number of classes
```

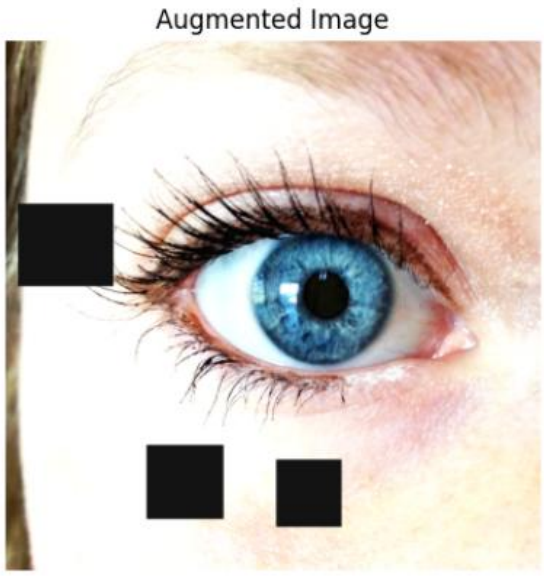
```
  ])

# Model Compilation
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=50,
    verbose=1,
    callbacks=[es, lr, myCallback()]
)
```

5.4 Evaluate The Model

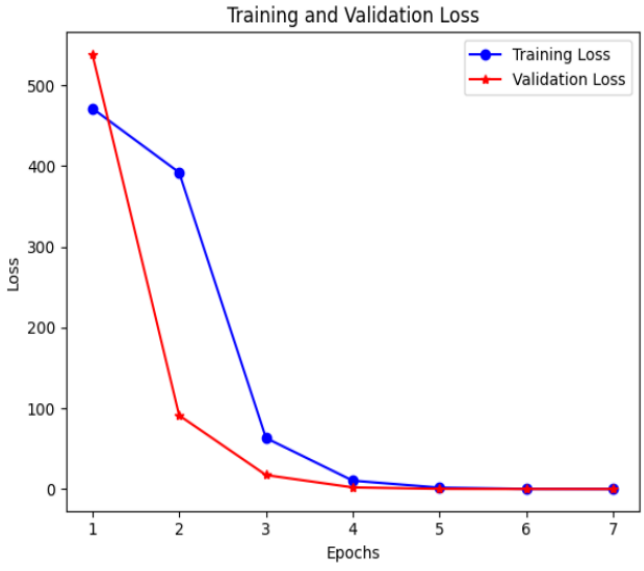
```
import matplotlib.pyplot as plt
# Replace `history` with your actual variable name if different
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
# Plot Accuracy
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(epochs, acc, 'bo-', label='Training Accuracy')
plt.plot(epochs, val_acc, 'r*-', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
# Plot Loss
plt.subplot(1, 2, 2)
plt.plot(epochs, loss, 'bo-', label='Training Loss')
plt.plot(epochs, val_loss, 'r*-', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.tight_layout()
plt.show()
```

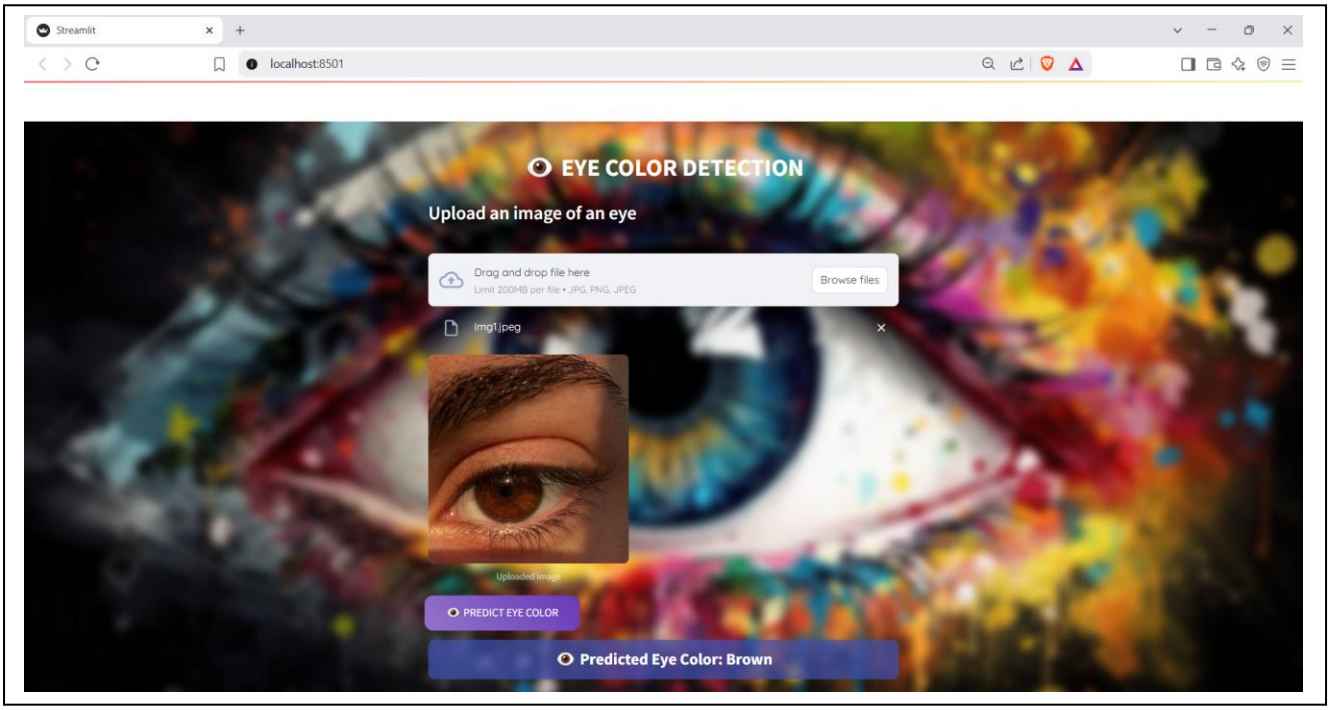
5.5 Test the model with New Image

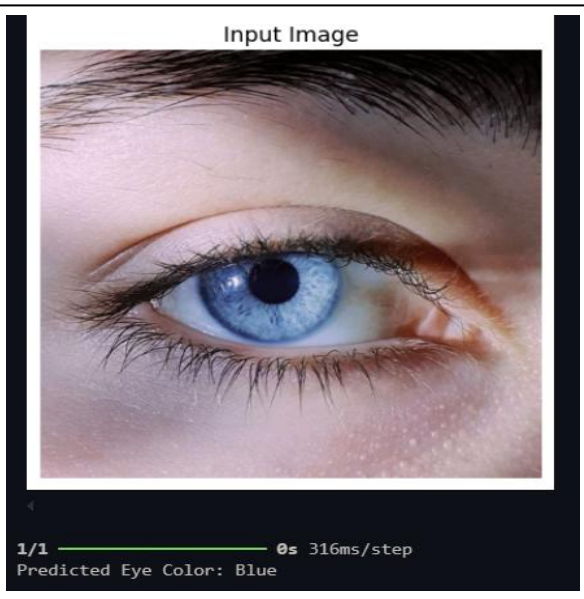
```
def predict_eye_color_from_array(img):
    img = cv2.resize(img, image_size)
    img = img / 255.0
    img = np.expand_dims(img, axis=0)

    prediction = model.predict(img)
    predicted_class = np.argmax(prediction)

    return eye_colors[predicted_class]

test_image = cv2.imread("./predicting_images/img5.jpeg")
test_image = cv2.cvtColor(test_image, cv2.COLOR_BGR2RGB)

plt.imshow(test_image)
plt.axis('off')
plt.title("Input Image")
plt.show()
predicted_color = predict_eye_color_from_array(test_image)
print("Predicted Eye Color:", predicted_color)
```

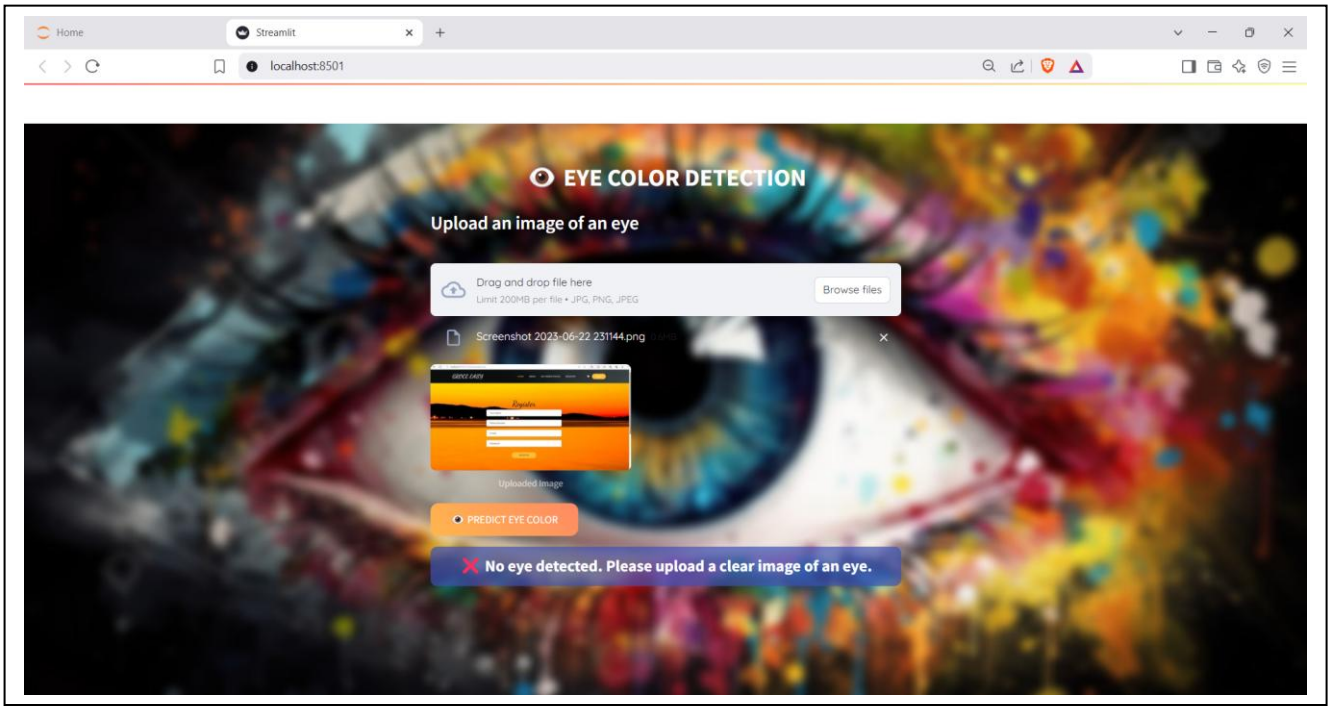



```
Epoch 1/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 7s 1s/step - accuracy: 0.1219 - loss: 594.5156 - val_accuracy: 0.2500 - val_loss: 805.4342 - learning_rate: 0.0010
Epoch 2/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 5s 817ms/step - accuracy: 0.2781 - loss: 493.2723 - val_accuracy: 0.3750 - val_loss: 71.1610 - learning_rate: 0.0010
Epoch 3/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 786ms/step - accuracy: 0.3570 - loss: 61.2180 - val_accuracy: 0.4875 - val_loss: 5.8067 - learning_rate: 0.0010
Epoch 4/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 738ms/step - accuracy: 0.4000 - loss: 15.9327 - val_accuracy: 0.7625 - val_loss: 0.8667 - learning_rate: 0.0010
Epoch 5/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 663ms/step - accuracy: 0.7461 - loss: 1.3595 - val_accuracy: 0.8500 - val_loss: 0.5906 - learning_rate: 0.0010
Epoch 6/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 764ms/step - accuracy: 0.8258 - loss: 0.6210 - val_accuracy: 0.9125 - val_loss: 0.3484 - learning_rate: 0.0010
Epoch 7/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 456ms/step - accuracy: 0.9406 - loss: 0.2951
Reached 95% accuracy so cancelling training!
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 773ms/step - accuracy: 0.9461 - loss: 0.2742 - val_accuracy: 1.0000 - val_loss: 0.0584 - learning_rate: 0.0010
```




Streamlit
localhost:8501
EYE COLOR DETECTION
Upload an image of an eye
Drag and drop file here
Limit 200MB per file • JPG, PNG, JPEG
Browse files
img1.jpeg
Uploaded Image
PREDICT EYE COLOR
Predicted Eye Color: Brown


Input Image
1/1 ——————— 0s 316ms/step
Predicted Eye Color: Blue


Home
Streamlit
localhost:8501
EYE COLOR DETECTION
Upload an image of an eye
Drag and drop file here
Limit 200MB per file • JPG, PNG, JPEG
Browse files
Screenshot 2023-06-22 231144.png
Uploaded Image
PREDICT EYE COLOR
No eye detected. Please upload a clear image of an eye.

7. CONCLUSION AND FUTURE SCOPE

This project demonstrates the effectiveness of CNN and Transfer Learning in detecting eye color from images. In the future, we aim to:

- **Expand the dataset for better generalization**: Increasing the size and diversity of the dataset (different lighting, angles, ethnicities) helps the model learn more robust features, reducing overfitting and improving accuracy on unseen data.
- **Incorporate real-time detection**: Implementing real-time detection involves optimizing the model for speed and deploying it with tools like OpenCV or TensorFlow Lite, allowing the system to predict instantly from live camera input.
- **Implement more advanced deep learning architectures**: Using more powerful models like ResNet, EfficientNet, or Vision Transformers can capture complex patterns better, leading to improved performance compared to simpler CNNs.
- **Improve preprocessing for better feature extraction**: Enhancing preprocessing steps like normalization, noise reduction, and advanced augmentation techniques ensures cleaner, more informative inputs, making it easier for the model to learn important features.

8. REFERENCE

- Deep Learning by Ian Goodfellow, Yoshua Bengio, and Aaron Courville.

- ImageNet Classification with Deep Convolutional Neural Networks by Alex Krizhevsky et al.

- Human Eye Color Classification Using CNN Based Deep Learning Approach by M. S. Bhargavi and P. Pranathi (2021).

- Deep Learning for Eye Color Classification in the Wild by Jian Zhao et al. (2018).

- Research papers and online datasets relevant to image classification and CNN.