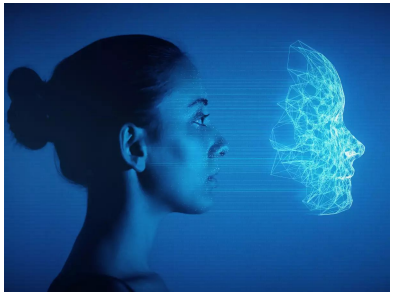


Multimodal Approach To Face and Iris Recognition

Members:
Zheng Wei Ng
Saishnu Ramesh Kumar
Ren Hao Wong

Core Concept

- **Multimodal Approach:** Combining multiple biometric modalities (enhance accuracy and robustness in identification)
- **Multimodal Fusion:** Integrates both modalities with weights to create a comprehensive biometric representation.
- **Neural Networks:** Employing deep learning architectures for feature extraction, fusion and classification.

Core Concept

- **Face Recognition:** Extract features from facial images, like ***landmarks***, and ***shape*** to allow for accurate identification and verification.

- **Iris Recognition:** Extract unique patterns in the iris like ***crypt***, ***collarette***, ***etc*** for precise identification.

Tech Stack

- **Language:** Python

- **Libraries:**
  - SciKit
  - Numpy
  - Tensorflow
  - Keras
  - OpenCV (cv2)

Dataset

- VISA Face and Iris Multimodal Biometrics Database published by: Visvesvaraya Technological University, Belagavi.
- 100 individuals aged **10 to 90 years old**.
- Captured **indoors and outdoors**.
- **Face images** captured with various cameras and scenarios, resulting in ***reflections***, ***shadows***, ***occlusions***, ***expressions***, and ***pose variations***.
- **Iris images** also ***exhibit variations***, ***occlusions***, and ***alignment***.

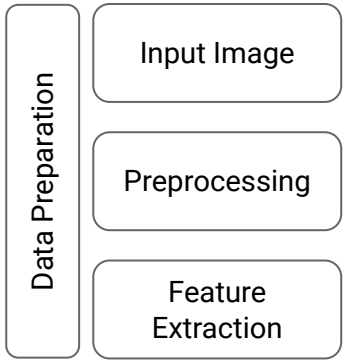
Data Preparation Architecture/ Pipeline

Input Image: Parse the data (images)

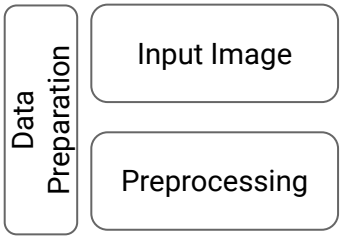
Preprocessing: Crop, binarize, get ready for next step

Feature Extraction: Extract some notable data to train the model

K-nearest neighbor



Deep Learning



Face Implementation

Data Preparation: Parsing Input

- Processes the dataset of face images stored within the directory.
  - Extracts the metadata
    - Such as **labels** and **IDs** for each image.
  - Compiles them into a list.

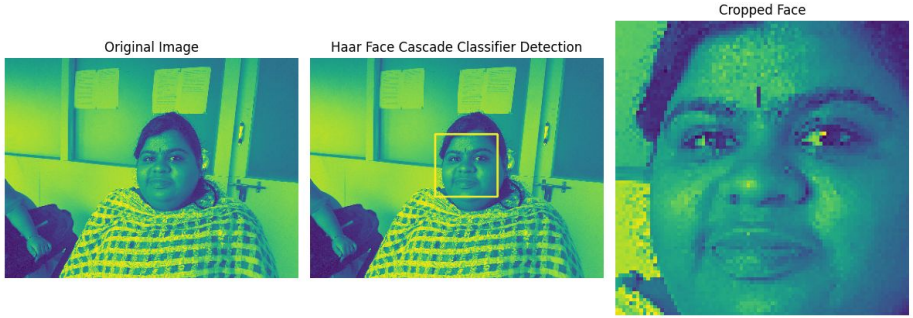
```python
def parse_face_dataset() -> tuple[list[cv2.typing.MatLike], list[str]]:
    images: list[cv2.typing.MatLike] = []
    labels: list[str] = []

    # Iterate over each directory in the base directory
    for path in glob.iglob(base_directory + '/*'):
        # Extract the filename from the path
        folder_name = os.path.basename(path)

        # Parse the filename to extract the label
        match = re.search(r'(.*?)_2017_001', folder_name)
        if match:
            label = match.group(1)
        else:
            warnings.warn(f"No match found for filename: {folder_name}")
            continue

        for image_path in glob.iglob(path + '/*'):
            try:
                # Read the image as grayscale
                image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
```

Face Implementation

Data Preparation: Preprocessing

```
face_cascade = cv2.CascadeClassifier(
    'Dependencies/haarcascade_frontalface_alt2.xml')

for face_image in face_images:
    (image, image_id, label) = face_image
    image_id += 1

    faces = face_cascade.detectMultiScale(image, 1.1, 4)
```

- Processes the face images found and detects faces in each image using a Haar Cascade classifier.
  - Crop the faces.
  - Saves them to an output directory
  - Splits the dataset (80% Training, 20% Test).

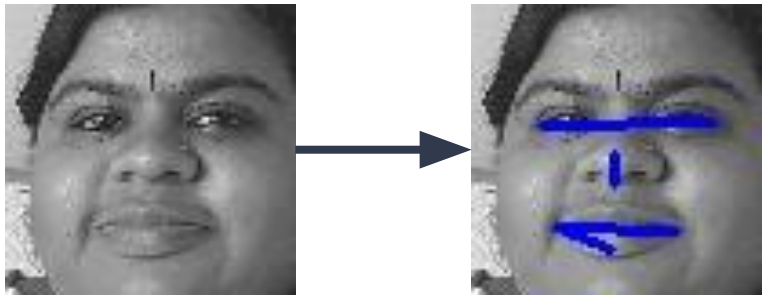


Face Implementation

Data Preparation: Feature Extraction

```
> def calculate_eye_distance(shape): ⋯
> def calculate_nose_shape(shape): ⋯
> def calculate_lips_contour(shape): ⋯
> def calculate_mouth_wrinkles(shape): ⋯
> def draw_lines(image, shape): ⋯
```

- Detects each image stored previously then predicts facial features to extract information by drawing lines and calculating:
  - Distance between the eyes
  - Nose width and height
  - Mouth width and height.



Face Implementation

Data Preparation: Feature Extraction

```
predictor = dlib.shape_predictor(
    'Dependencies/shape_predictor_68_face_landmarks.dat')

for i, d in enumerate(dets):
    shape = predictor(image, d)
    landmarks = [(shape.part(i).x, shape.part(i).y)
                 for i in range(68)]
    for (x, y) in landmarks:
        cv2.circle(image, (x, y), 1, (0, 0, 255), -1)
```

- Uses a shape predictor to locate landmarks.
    - Proceeds to extract the coordinate points of all 68 facial landmarks found.
- Continues to flatten each set into a feature vector and saves the feature vectors as NumPy files.

Iris Implementation

Data Preparation: Parsing Input

- Processes the dataset of eye images stored within the directory.
  - Extracts the metadata
    - Such as **labels** and **IDs** for each image.
  - Compiles them into a list.

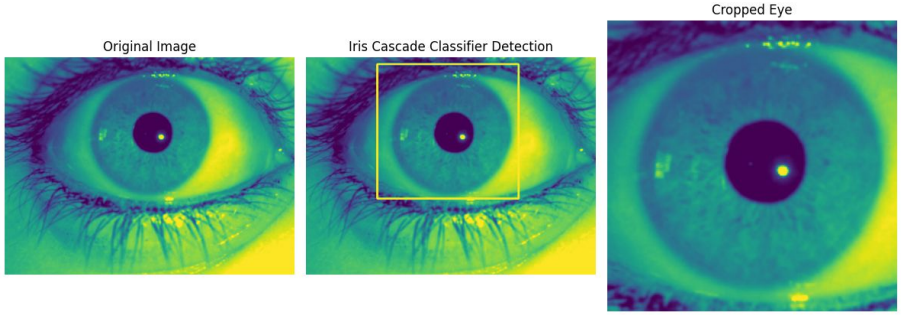
```
def parse_iris_dataset(keep_reflections: bool = False) -> tuple[list, list, list[str]]:
    left_images: list = []
    left_labels: list[str] = []
    right_images: list = []
    right_labels: list[str] = []

    for path in glob.iglob(base_directory + '/*'):
        folder_name = os.path.basename(path)

        label = folder_name

        # Process Left Eye
        for image_path in glob.iglob(path + '/L/*'):
            image = cv2.imread(image_path)

            image_hough_processed = process_hough(image_path, image, 50)
```

Iris Implementation

Data Preparation: Preprocessing

```
def process_hough(imagepath, image, radius):
    image = cv2.resize(image, (640, 480), interpolation = cv2.INTER_LINEAR)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    gray = cv2.medianBlur(image, 11)
    # edge = cv2.Canny(image, 100, 200)
    ret, _ = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

    circles = cv2.HoughCircles(gray, cv2.HOUGH_GRADIENT, 1,
                               50, param1=ret, param2=30, minRadius=20, maxRadius=100)
```

- Processes the eye images found and detects Irises in each image using **Hough Circles Algorithm**
  - Crop the eye (keep iris)
  - Saves them to an output directory
  - splits the dataset (80% Training, 20% Test).



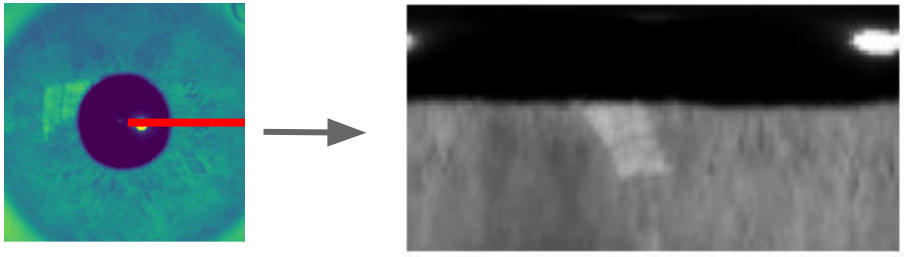
Migrated from Iris Cascade Classifier -> Hough Circles

Iris Implementation

Data Preparation: Preprocessing

```
q = np.arange(0.00, np.pi * 2, 0.01)
inn = np.arange(0, int(image.shape[0] / 2), 1)

cartisian_image = np.empty(shape=[inn.size, int(image.shape[1]), 3])
m = interp1d([np.pi * 2, 0], [0, image.shape[1]])

for r in inn:
    for t in q:
        polarX = int((r * np.cos(t)) + image.shape[1] / 2)
        polarY = int((r * np.sin(t)) + image.shape[0] / 2)
        try:
            cartisian_image[int(r)][int(m(t)-1)] = image[polarY][polarX]
```

- Getting the images ready for feature extraction using different algorithms
  - **Hough Circles:** Find Circles (Irises) in the image
  - **Daugman Rubber Sheet Model:** Turn Iris Into a sheet



Iris Implementation

Data Preparation: Feature Extraction

```
ccoeffs = pywt.dwt2(img[:, :, 0], 'haar')
LL, (LH, HL, HH) = ccoeffs
for coef in [LL, LH, HL, HH]:
    features.append(np.mean(coef))
    features.append(np.std(coef))
    features.append(np.max(coef))
    features.append(np.min(coef))
    features.append(np.median(coef))
```

- Use **Discrete Wavelet Transform** to extract features from the image
  - Separates into different bands “sheets”
  - Directional data
    - Horizontal
    - Vertical
    - Diagonal

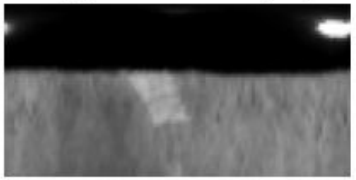
Why?

We can extract more features from the same data (higher accuracy)

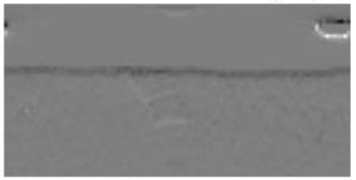
Iris Implementation
Data Preparation: Feature Extraction

With Reflection

Approximation (LL)



Horizontal Detail (LH)

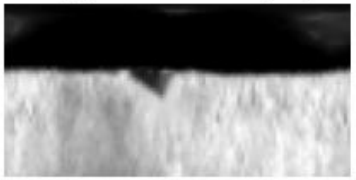


Vertical Detail (HL)

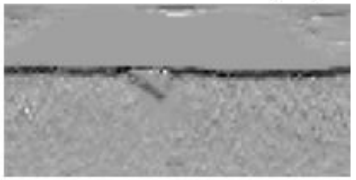
Diagonal Detail (HH)

Removed Reflection

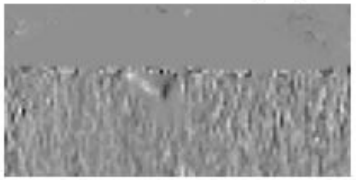
Approximation (LL)



Horizontal Detail (LH)



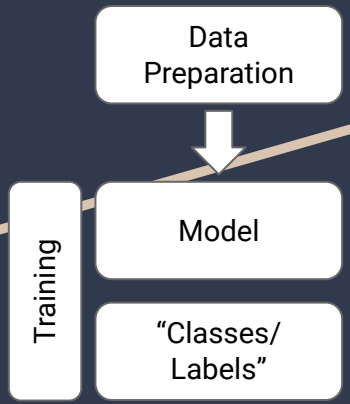
Vertical Detail (HL)



Diagonal Detail (HH)

Single Modal Implementation

Training

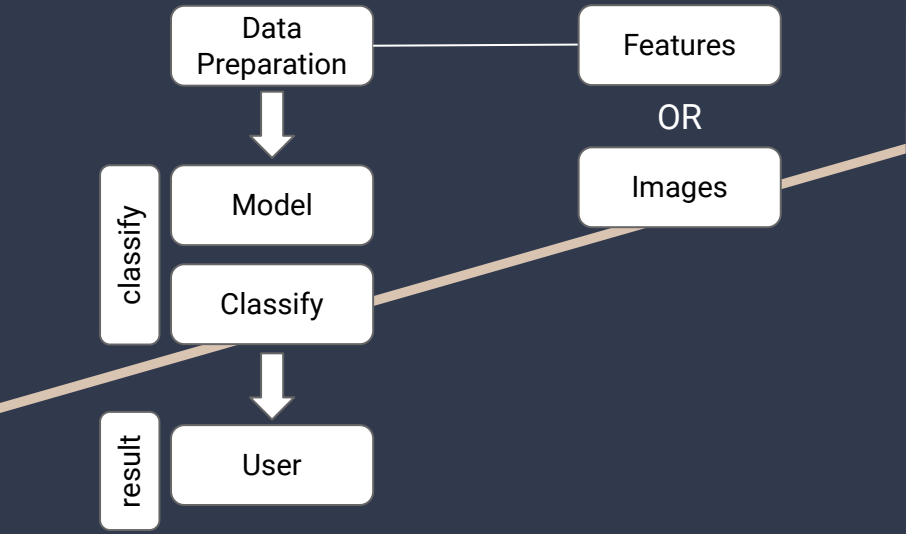


- **K-Nearest Neighbour**
  - Trained 2 models with built in scikit library
    - Iris
    - Face
  - Uses the ball-tree algorithm
  - Fast

- **Deep Neural Network**
  - Trained 3 models using Tensorflow
    - L Iris
    - R Iris
    - Face
  - Uses a Siamese Neural Network
  - Slow

Single Modal Implementation

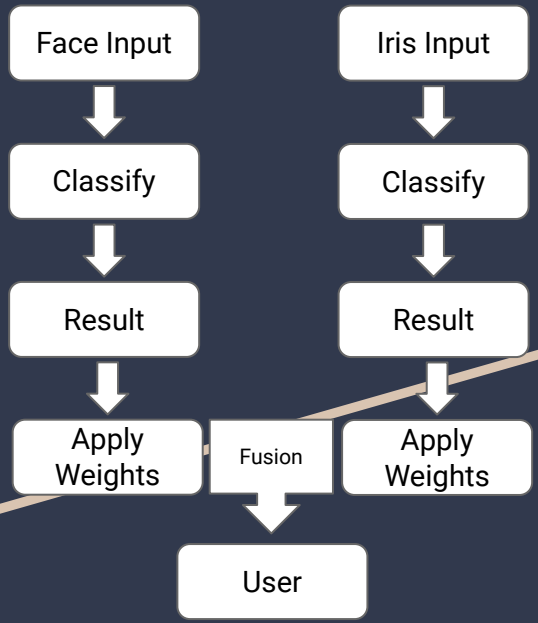
Testing



- **K-Nearest Neighbour**
  - Feed it features
    - Iris Model
    - Face Model

- **Deep Neural Network**
  - Feed it images
    - Iris Model
    - Face Model

Score Fusion Approach



- Same methodology as **Single Modal**
- Does a fusion on the **Scores** in the end
- Apply **weights** and calculate a new score before outputting **final score**

```
weight_1 = 0.5
weight_2 = 0.5

final_score = calculate_score(score_1, score_2, weight_1, weight_2) * 100
print("Scores: ")
print("scores_1: " + str(score_1) + ", " + "scores_2: " + str(score_2))
print("weights: " + str(weight_1) + ", " + str(weight_2))
print("weighted Scores: " + str(final_score) +"%")
```

Deep Neural Network Implementation

Siamese-NN

```
def create_siamese_network(input_shape, embedding_dim):
    input = layers.Input(shape=input_shape)
    x = layers.Conv2D(32, (3, 3), activation='relu')(input)
    x = layers.MaxPooling2D((2, 2))(x)
    x = layers.Conv2D(64, (3, 3), activation='relu')(x)
    x = layers.MaxPooling2D((2, 2))(x)
    x = layers.Conv2D(128, (3, 3), activation='relu')(x)
    x = layers.MaxPooling2D((2, 2))(x)
    x = layers.Conv2D(128, (3, 3), activation='relu')(x)
    x = layers.MaxPooling2D((2, 2))(x)
    x = layers.Flatten()(x)
    x = layers.Dense(512, activation='relu')(x)
    output = layers.Dense(embedding_dim)(x)
    return models.Model(input, output)
```

- Few-shots learning
  - Requires less data
- Run 2 neural network simultaneously
- Calculates similarities, comparing predictions to true label

```
# Create Siamese network branches
base_network = create_siamese_network(input_shape, embedding_dim)
input_a = layers.Input(shape=input_shape)
input_b = layers.Input(shape=input_shape)

processed_a = base_network(input_a)
processed_b = base_network(input_b)

distance = tf.abs(processed_a - processed_b)

output = layers.Dense(1, activation='sigmoid')(distance)

model = models.Model([input_a, input_b], output)

model.compile(optimizer='adam', loss='binary_crossentropy',
              metrics=['accuracy'])
```

- Note: Initially used RBNN but yielded terrible results

K-Nearest Neighbor Implementation

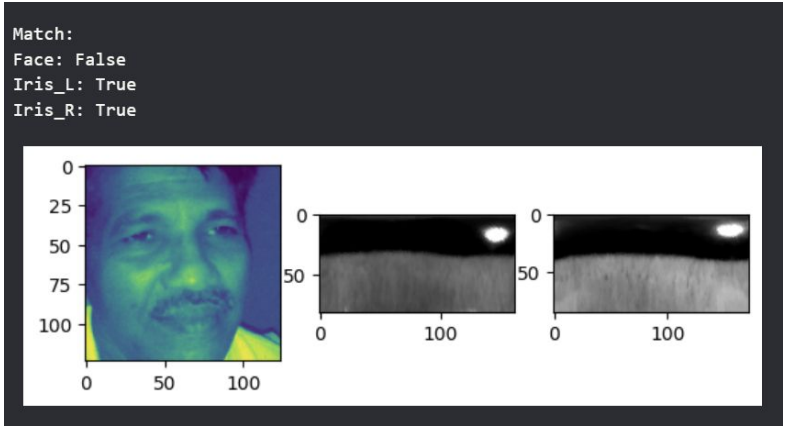
```python
def train(
    x_train,
    y_train,
    n_neighbors: int | None = None,
    knn_algo: str = 'ball_tree',
    verbose: bool = False,
):
    if n_neighbors is None:
        n_neighbors = int(round(math.sqrt(len(x_train))))
        if verbose:
            print("Chose n_neighbors automatically:", n_neighbors)

    knn_clf = neighbors.KNeighborsClassifier(
        n_neighbors=n_neighbors,
        algorithm=knn_algo,
        weights='distance',
    )
    knn_clf.fit(x_train, y_train)

    return knn_clf
```

- Train model
  - Plot points on a “graph”
  - Uses features extracted
- Test model
  - Looks at the ‘k’ closest point to it (feature)
  - Assigns to most common category

Results

Siamese Neural Network

- Single Modal (all dataset)
  - Face: ~83%
  - Iris: ~93%
- Multimodal (targeted images)



- MobileNet: ~54%
- Custom RBNN: ~27%

Results

K Nearest Neighbor

```
model = pickle.load(open(os.path.join(model_path), 'rb'))

predicted_labels, accuracy = knn.predict(x_test, y_test, model=model)

print('Accuracy:', accuracy)

cv2.destroyAllWindows()
```

- Single Modal Performance
  - Face: ~73%
  - Iris: ~96%
- Multi Modal Performance
  - Balanced Weight: ~84.5%
  - Skewed Iris(30 : 70): ~89.1%
  - Skewed Face(70 : 30): ~79.9%

```
KNN scores:
scoresL: 0.967032967032967, scoresR: 0.9593023255813954
weights: 0.5, 0.5
weighted score of Irises: 96.31676463071813%
```

```
KNN scores:
score of Faces: 73.134328358208956%
```

Thoughts

- Results can be enhanced by employing **weights** to them.
- Can compensate less ideal results from single modal.
- Dataset, for faces are inconsistent and insufficient for some classes, 1 of the classes only had 5 images.
- When data sample is insufficient, using one-shot/few-shot learning architectures can greatly improve results.

Future Work

Interests

- Improving Testing workflow support multi-image testing
- Capturing data on ourselves and testing it
- Exploring Feature Fusion

Experimentation

- Experiment with pre-trained models
- Experiment with a separate CNN model deciding weight split dynamically
- Experiment on other datasets
- Experiment with other Neural Network Architecture

References

[1] (PDF) Contributions to practical iris biometrics on smartphones

[2] Face-Iris multimodal biometric recognition system based on deep learning | Multimedia Tools and Applications

[3] A Multimodal Biometric System for Iris and Face Traits Based on Hybrid Approaches and Score Level Fusion

[4] Quality assessment-based iris and face fusion recognition with dynamic weight | The Visual Computer

[5] Deep Learning Based Face Recognition Method using Siamese Network

Thank you!