### Overview of the Process

This section focuses on data preparation for a robust face detection and quality control measures using OpenCV's Haar Cascade classifier, MTCNN and custom sampling techniques. The goal is to ensure consistent image quality and proper facial feature representation across the dataset.

### Key Components

1. **Detection Pipeline**:
   - OpenCV Haar Cascade face detection - Fast and efficient face detection using pre-trained cascade classifiers
   - Custom sampling methodology
   - Quality verification system
2. **Quality Control Process**:
   - Sample 15 images per person (5 start/middle/end)
   - Face detection confidence thresholds
   - Manual review flagging system
3. **Technical Requirements**:
   - Centered facial features
   - Consistent lighting/background
   - Standardized dimensions
   - Clear facial visibility
4. **Verification Steps**:
   - Automated face detection checks
   - Sample image visual inspection
   - Systematic issue identification
   - Quality metrics logging

## Convert images from HEIC to JPG format

In [ ]:
```
import os
import shutil
from pillow_heif import register_heif_opener
from PIL import Image
register_heif_opener()

def convert_heic_to_jpg(base_path):
    # Get all HEIC files (case insensitive)
    heic_images = [f for f in os.listdir(base_path) if f.lower().endswith
    converted_count = 0

    for img in heic_images:
        heic_path = os.path.join(base_path, img)
        jpg_path = os.path.splitext(heic_path)[0] + '.jpg'

        try:
```

```
            with Image.open(heic_path) as i:
                i.convert('RGB').save(jpg_path, 'JPEG', quality=95)
            print(f"Converted {img} to JPG")
            converted_count += 1
            # Remove original HEIC file after successful conversion
            os.remove(heic_path)
        except Exception as e:
            print(f"Error converting {img}: {e}")

    print(f"Converted {converted_count} images")

def organize_user_images(base_path):
    # Get all JPG images
    images = [f for f in os.listdir(base_path) if f.lower().endswith('.jp
    user = os.path.basename(base_path)

    for idx, img in enumerate(images, 1):
        subfolder_name = f"{user}_{idx}"
        subfolder_path = os.path.join(base_path, subfolder_name)
        os.makedirs(subfolder_path, exist_ok=True)

        old_img_path = os.path.join(base_path, img)
        new_img_name = f"{subfolder_name}.jpg"
        new_img_path = os.path.join(subfolder_path, new_img_name)
        shutil.move(old_img_path, new_img_path)

def process_all_users(paths):
    for path in paths:
        print(f"\nProcessing {path}")
        convert_heic_to_jpg(path)
        organize_user_images(path)

if __name__ == "__main__":
    paths = [
        "/Users/ryangichuru/Documents/SSD-K/Uni/2nd year/Intro to AI/CNN/
        "/Users/ryangichuru/Documents/SSD-K/Uni/2nd year/Intro to AI/CNN/
    ]
    process_all_users(paths)
```

## 1. Data Organization

The goal is to organize our image data through several key steps:

- Converting HEIC images to JPG format for compatibility (Based on code from: https://stackoverflow.com/questions/54395735/how-to-work-with-heic-image-file-types-in-python)
- Creating individual folders to store each image separately
- Organizing images into user-specific directories for better structure

In [1]:
```
import os
import shutil
from pathlib import Path

def organize_user_images():
    # Base directory where user folders are located
    base_dir = "Original Data"

    # Get all user directories
```

```
    user_dirs = [d for d in os.listdir(base_dir) if os.path.isdir(os.path

    for user in user_dirs:
        user_path = os.path.join(base_dir, user)
        # Get all images in user directory
        images = [f for f in os.listdir(user_path) if f.lower().endswith(

        # Create numbered subfolders and move images
        for idx, img in enumerate(images, 1):
            # Create subfolder name (e.g., "john_1")
            subfolder_name = f"{user}_{idx}"
            subfolder_path = os.path.join(user_path, subfolder_name)

            # Create subfolder if it doesn't exist
            os.makedirs(subfolder_path, exist_ok=True)

            # Get image extension
            _, ext = os.path.splitext(img)

            # New image name will match subfolder name
            new_img_name = f"{subfolder_name}{ext}"

            # Move and rename image
            old_img_path = os.path.join(user_path, img)
            new_img_path = os.path.join(subfolder_path, new_img_name)
            shutil.move(old_img_path, new_img_path)

if __name__ == "__main__":
    organize_user_images()
```

# 2. Environment Setup and Hardware Optimization

In [3]:
```
%pip install mlx
%pip install tensorflow

# Install other dependencies
%pip install numpy matplotlib opencv-python scikit-learn
```

```
Requirement already satisfied: mlx in /opt/homebrew/Caskroom/miniforge/bas
e/envs/mlx-env/lib/python3.12/site-packages (0.22.0)
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: tensorflow in /opt/homebrew/Caskroom/minifo
rge/base/envs/mlx-env/lib/python3.12/site-packages (2.18.0)
Requirement already satisfied: absl-py>=1.0.0 in /opt/homebrew/Caskroom/mi
niforge/base/envs/mlx-env/lib/python3.12/site-packages (from tensorflow)
(2.1.0)
Requirement already satisfied: astunparse>=1.6.0 in /opt/homebrew/Caskroo
m/miniforge/base/envs/mlx-env/lib/python3.12/site-packages (from tensorflo
w) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /opt/homebrew/Caskr
oom/miniforge/base/envs/mlx-env/lib/python3.12/site-packages (from tensorf
low) (24.12.23)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /op
t/homebrew/Caskroom/miniforge/base/envs/mlx-env/lib/python3.12/site-packag
es (from tensorflow) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in /opt/homebrew/Caskro
om/miniforge/base/envs/mlx-env/lib/python3.12/site-packages (from tensorfl
ow) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in /opt/homebrew/Caskroom/
miniforge/base/envs/mlx-env/lib/python3.12/site-packages (from tensorflow)
(18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /opt/homebrew/Caskroo
m/miniforge/base/envs/mlx-env/lib/python3.12/site-packages (from tensorflo
w) (3.4.0)
Requirement already satisfied: packaging in /opt/homebrew/Caskroom/minifor
ge/base/envs/mlx-env/lib/python3.12/site-packages (from tensorflow) (24.2)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.
3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.3 in /opt/homebrew/Caskroom/miniforg
e/base/envs/mlx-env/lib/python3.12/site-packages (from tensorflow) (5.29.
3)
Requirement already satisfied: requests<3,>=2.21.0 in /opt/homebrew/Caskro
om/miniforge/base/envs/mlx-env/lib/python3.12/site-packages (from tensorfl
ow) (2.32.3)
Requirement already satisfied: setuptools in /opt/homebrew/Caskroom/minifo
rge/base/envs/mlx-env/lib/python3.12/site-packages (from tensorflow) (75.
1.0)
Requirement already satisfied: six>=1.12.0 in /opt/homebrew/Caskroom/minif
orge/base/envs/mlx-env/lib/python3.12/site-packages (from tensorflow) (1.1
6.0)
Requirement already satisfied: termcolor>=1.1.0 in /opt/homebrew/Caskroom/
miniforge/base/envs/mlx-env/lib/python3.12/site-packages (from tensorflow)
(2.5.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /opt/homebrew/C
askroom/miniforge/base/envs/mlx-env/lib/python3.12/site-packages (from ten
sorflow) (4.12.2)
Requirement already satisfied: wrapt>=1.11.0 in /opt/homebrew/Caskroom/min
iforge/base/envs/mlx-env/lib/python3.12/site-packages (from tensorflow)
(1.17.2)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /opt/homebrew/Caskro
om/miniforge/base/envs/mlx-env/lib/python3.12/site-packages (from tensorfl
ow) (1.69.0)
Requirement already satisfied: tensorboard<2.19,>=2.18 in /opt/homebrew/Ca
skroom/miniforge/base/envs/mlx-env/lib/python3.12/site-packages (from tens
orflow) (2.18.0)
Requirement already satisfied: keras>=3.5.0 in /opt/homebrew/Caskroom/mini
forge/base/envs/mlx-env/lib/python3.12/site-packages (from tensorflow) (3.
8.0)
Requirement already satisfied: numpy<2.1.0,>=1.26.0 in /opt/homebrew/Caskr
```

```
oom/miniforge/base/envs/mlx-env/lib/python3.12/site-packages (from tensorf
low) (2.0.2)
Requirement already satisfied: h5py>=3.11.0 in /opt/homebrew/Caskroom/mini
forge/base/envs/mlx-env/lib/python3.12/site-packages (from tensorflow) (3.
12.1)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.4.0 in /opt/homebrew/Ca
skroom/miniforge/base/envs/mlx-env/lib/python3.12/site-packages (from tens
orflow) (0.4.1)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /opt/homebrew/Caskroo
m/miniforge/base/envs/mlx-env/lib/python3.12/site-packages (from astunpars
e>=1.6.0->tensorflow) (0.44.0)
Requirement already satisfied: rich in /opt/homebrew/Caskroom/miniforge/ba
se/envs/mlx-env/lib/python3.12/site-packages (from keras>=3.5.0->tensorflo
w) (13.9.4)
Requirement already satisfied: namex in /opt/homebrew/Caskroom/miniforge/b
ase/envs/mlx-env/lib/python3.12/site-packages (from keras>=3.5.0->tensorfl
ow) (0.0.8)
Requirement already satisfied: optree in /opt/homebrew/Caskroom/miniforge/
base/envs/mlx-env/lib/python3.12/site-packages (from keras>=3.5.0->tensorf
low) (0.14.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /opt/homebrew/C
askroom/miniforge/base/envs/mlx-env/lib/python3.12/site-packages (from req
uests<3,>=2.21.0->tensorflow) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /opt/homebrew/Caskroom/mini
forge/base/envs/mlx-env/lib/python3.12/site-packages (from requests<3,>=2.
21.0->tensorflow) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /opt/homebrew/Caskroo
m/miniforge/base/envs/mlx-env/lib/python3.12/site-packages (from requests<
3,>=2.21.0->tensorflow) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in /opt/homebrew/Caskroo
m/miniforge/base/envs/mlx-env/lib/python3.12/site-packages (from requests<
3,>=2.21.0->tensorflow) (2024.12.14)
Requirement already satisfied: markdown>=2.6.8 in /opt/homebrew/Caskroom/m
iniforge/base/envs/mlx-env/lib/python3.12/site-packages (from tensorboard<
2.19,>=2.18->tensorflow) (3.7)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /o
pt/homebrew/Caskroom/miniforge/base/envs/mlx-env/lib/python3.12/site-packa
ges (from tensorboard<2.19,>=2.18->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in /opt/homebrew/Caskroom/m
iniforge/base/envs/mlx-env/lib/python3.12/site-packages (from tensorboard<
2.19,>=2.18->tensorflow) (3.1.3)
Requirement already satisfied: MarkupSafe>=2.1.1 in /opt/homebrew/Caskroo
m/miniforge/base/envs/mlx-env/lib/python3.12/site-packages (from werkzeug>
=1.0.1->tensorboard<2.19,>=2.18->tensorflow) (3.0.2)
Requirement already satisfied: markdown-it-py>=2.2.0 in /opt/homebrew/Cask
room/miniforge/base/envs/mlx-env/lib/python3.12/site-packages (from rich->
keras>=3.5.0->tensorflow) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /opt/homebrew/Ca
skroom/miniforge/base/envs/mlx-env/lib/python3.12/site-packages (from rich
->keras>=3.5.0->tensorflow) (2.19.1)
Requirement already satisfied: mdurl~=0.1 in /opt/homebrew/Caskroom/minifo
rge/base/envs/mlx-env/lib/python3.12/site-packages (from markdown-it-py>=
2.2.0->rich->keras>=3.5.0->tensorflow) (0.1.2)
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: numpy in /opt/homebrew/Caskroom/miniforge/b
ase/envs/mlx-env/lib/python3.12/site-packages (2.0.2)
Requirement already satisfied: matplotlib in /opt/homebrew/Caskroom/minifo
rge/base/envs/mlx-env/lib/python3.12/site-packages (3.10.0)
Requirement already satisfied: opencv-python in /opt/homebrew/Caskroom/min
iforge/base/envs/mlx-env/lib/python3.12/site-packages (4.11.0.86)
```

```
Requirement already satisfied: scikit-learn in /opt/homebrew/Caskroom/mini
forge/base/envs/mlx-env/lib/python3.12/site-packages (1.6.1)
Requirement already satisfied: contourpy>=1.0.1 in /opt/homebrew/Caskroom/
miniforge/base/envs/mlx-env/lib/python3.12/site-packages (from matplotlib)
(1.3.1)
Requirement already satisfied: cycler>=0.10 in /opt/homebrew/Caskroom/mini
forge/base/envs/mlx-env/lib/python3.12/site-packages (from matplotlib) (0.
12.1)
Requirement already satisfied: fonttools>=4.22.0 in /opt/homebrew/Caskroo
m/miniforge/base/envs/mlx-env/lib/python3.12/site-packages (from matplotli
b) (4.55.3)
Requirement already satisfied: kiwisolver>=1.3.1 in /opt/homebrew/Caskroo
m/miniforge/base/envs/mlx-env/lib/python3.12/site-packages (from matplotli
b) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /opt/homebrew/Caskroom/m
iniforge/base/envs/mlx-env/lib/python3.12/site-packages (from matplotlib)
(24.2)
Requirement already satisfied: pillow>=8 in /opt/homebrew/Caskroom/minifor
ge/base/envs/mlx-env/lib/python3.12/site-packages (from matplotlib) (11.1.
0)
Requirement already satisfied: pyparsing>=2.3.1 in /opt/homebrew/Caskroom/
miniforge/base/envs/mlx-env/lib/python3.12/site-packages (from matplotlib)
(3.2.1)
Requirement already satisfied: python-dateutil>=2.7 in /opt/homebrew/Caskr
oom/miniforge/base/envs/mlx-env/lib/python3.12/site-packages (from matplot
lib) (2.9.0.post0)
Requirement already satisfied: scipy>=1.6.0 in /opt/homebrew/Caskroom/mini
forge/base/envs/mlx-env/lib/python3.12/site-packages (from scikit-learn)
(1.15.1)
Requirement already satisfied: joblib>=1.2.0 in /opt/homebrew/Caskroom/min
iforge/base/envs/mlx-env/lib/python3.12/site-packages (from scikit-learn)
(1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /opt/homebrew/Caskr
oom/miniforge/base/envs/mlx-env/lib/python3.12/site-packages (from scikit-
learn) (3.5.0)
Requirement already satisfied: six>=1.5 in /opt/homebrew/Caskroom/miniforg
e/base/envs/mlx-env/lib/python3.12/site-packages (from python-dateutil>=2.
7->matplotlib) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

In [4]:
```python
import mlx.core as mx
import mlx.nn as nn
import tensorflow as tf
import numpy as np

# Check if MLX can use Metal backend
print(f"MLX Metal available: {mx.metal.is_available()}")

# For TensorFlow, limit to CPU if on Apple Silicon
tf.config.set_visible_devices([], 'GPU')
```

```
MLX Metal available: True
```

In [5]:
```python
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
import os

print("TensorFlow version:", tf.__version__)
```

```
TensorFlow version: 2.18.0
```

# ## 3. Data Preparation and Face Detection

### Overview of the Process

This section focuses on implementing robust face detection and quality control measures using OpenCV's Haar Cascade classifier and custom sampling techniques. The goal is to ensure consistent image quality and proper facial feature representation across the dataset.

### Key Components

1. **Detection Pipeline**:
   - OpenCV Haar Cascade face detection [1][2] - Fast and efficient face detection using pre-trained cascade classifiers (https://docs.opencv.org/4.x/db/d28/tutorial_cascade_classifier.html) (https://docs.opencv.org/3.4.1/d7/d8b/tutorial_py_face_detection.html)
   - Custom sampling methodology
   - Quality verification system
2. **Quality Control Process**:
   - Sample 15 images per person (5 start/middle/end)
   - Face detection confidence thresholds
   - Manual review flagging system
3. **Technical Requirements**:
   - Centered facial features
   - Consistent lighting/background
   - Standardized dimensions
   - Clear facial visibility
4. **Verification Steps**:
   - Automated face detection checks
   - Sample image visual inspection
   - Systematic issue identification
   - Quality metrics logging

In [6]:
```
import cv2
import os
import matplotlib.pyplot as plt

def pick_images(folder_path):
    """
    Returns a list of 15 image paths: 5 from start, 5 from middle, 5 from
    If the folder has fewer than 15 images, it will return as many as pos
    """
    # Get all subfolders
    subfolders = [f.path for f in os.scandir(folder_path) if f.is_dir()]
    selected_paths = []
```

```
    for subfolder in subfolders:
        # Get all image files in the subfolder
        all_files = os.listdir(subfolder)
        image_files = sorted([f for f in all_files if f.lower().endswith(

        if len(image_files) == 0:
            print("No images found in:", subfolder)
            continue

        n = len(image_files)

        # Select images
        if n <= 15:
            selected_paths.extend([os.path.join(subfolder, f) for f in im
        else:
            start_5 = image_files[:5]
            mid_start = (n // 2) - 2  # center minus 2
            mid_5 = image_files[mid_start:mid_start + 5]
            end_5 = image_files[-5:]

            selected_paths.extend([os.path.join(subfolder, f) for f in st

    return selected_paths

def display_images(image_paths):
    plt.figure(figsize=(15, 5))  # Set figure size for horizontal display
    num_images = min(len(image_paths), 15)  # Ensure we don't exceed 15 i
    for idx, img_path in enumerate(image_paths[:num_images]):
        # Check if file exists
        if not os.path.exists(img_path):
            raise FileNotFoundError(f"Image not found at {img_path}")

        # Read and verify image loaded correctly
        img_orig = cv2.imread(img_path, cv2.IMREAD_COLOR)
        if img_orig is None:
            raise ValueError(f"Failed to load image at {img_path}")

        img_orig = cv2.cvtColor(img_orig, cv2.COLOR_BGR2RGB)
        plt.subplot(3, 5, idx + 1)  # 3 rows, 5 columns
        plt.imshow(img_orig)
        plt.title(os.path.basename(img_path))
        plt.axis('off')

    plt.tight_layout()
    plt.show()

person_folders = [
    "/Users/ryangichuru/Documents/SSD-K/Uni/2nd year/Intro to AI/CNN/assi
    "/Users/ryangichuru/Documents/SSD-K/Uni/2nd year/Intro to AI/CNN/assi
    "/Users/ryangichuru/Documents/SSD-K/Uni/2nd year/Intro to AI/CNN/assi
]

for person_folder in person_folders:
    print(f"Processing images from: {person_folder}")
    image_paths = pick_images(person_folder)
    display_images(image_paths)
```

```
[ WARN:0@0.623] global loadsave.cpp:268 findDecoder imread_('dataset/perso
nA/img001.jpg'): can't open/read file: check file path/integrity
```

```
---------------------------------------------------------------------------
error                                     Traceback (most recent call las
t)
Cell In[6], line 4
      1 import cv2
      3 img_orig = cv2.imread("dataset/personA/img001.jpg", cv2.IMREAD_COL
OR)
----> 4 img_orig = cv2.cvtColor(img_orig, cv2.COLOR_BGR2RGB)  # convert BG
R → RGB for plt
      5 plt.imshow(img_orig)
      6 plt.title("Original Image")

error: OpenCV(4.11.0) /Users/xperience/GHA-Actions-OpenCV/_work/opencv-pyt
hon/opencv-python/opencv/modules/imgproc/src/color.cpp:199: error: (-215:A
ssertion failed) !_src.empty() in function 'cvtColor'
```

In [ ]:

```python
import cv2
import os
import numpy as np

SHOW_PREVIEW = False

CASCADE_PATH = os.path.join(cv2.data.haarcascades, "haarcascade_frontalfa
face_cascade = cv2.CascadeClassifier(CASCADE_PATH)

# Constants for controlling face bounding box acceptance
MIN_FACE_WIDTH = 50   # skip if the detected face's width < 50 px
MIN_FACE_HEIGHT = 50  # skip if the detected face's height < 50 px

def crop_face_if_needed(image_path, area_threshold=0.95, debug=False):
    img = cv2.imread(image_path)
    if img is None:
        return None, "Unable to read image file."

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeigh

    if len(faces) == 0:
        return None, "No face found; skipping."

    # Pick the largest face by area
    largest_area = 0
    chosen_box = None
    for (x, y, w, h) in faces:
        area = w * h
        if area > largest_area:
            largest_area = area
            chosen_box = (x, y, w, h)

    x, y, w, h = chosen_box
    face_area = w * h
    img_area = img.shape[0] * img.shape[1]
    coverage_ratio = face_area / float(img_area)

    if debug:
        print(f"DEBUG: {os.path.basename(image_path)} -> coverage ratio =

    if coverage_ratio >= area_threshold:
        return None, f"Face covers ~{coverage_ratio*100:.1f}% => skipping
```

```
    # Check minimal face dimension
    if w < MIN_FACE_WIDTH or h < MIN_FACE_HEIGHT:
        return None, f"Face too small (w={w}, h={h}) => skipping."

    cropped_img = img[y:y+h, x:x+w]
    return cropped_img, f"Face covers ~{coverage_ratio*100:.1f}%; cropped

def crop_faces_recursively(input_dir, output_dir, area_threshold=0.95, de
    for root, dirs, files in os.walk(input_dir):
        rel_path = os.path.relpath(root, input_dir)
        out_subdir = os.path.join(output_dir, rel_path)
        os.makedirs(out_subdir, exist_ok=True)

        for filename in files:
            if filename.lower().endswith(('.png', '.jpg', '.jpeg')):
                input_path = os.path.join(root, filename)
                output_path = os.path.join(out_subdir, filename)

                cropped_img, status = crop_face_if_needed(input_path, are

                if cropped_img is not None:
                    cv2.imwrite(output_path, cropped_img)
                    print(f"[Cropped] {os.path.relpath(input_path, input_

                    if SHOW_PREVIEW:
                        import matplotlib.pyplot as plt
                        rgb_cropped = cv2.cvtColor(cropped_img, cv2.COLOR
                        plt.figure()
                        plt.title(f"Cropped: {filename}")
                        plt.imshow(rgb_cropped)
                        plt.axis('off')
                        plt.show()
                else:
                    print(f"[Skipped] {os.path.relpath(input_path, input_

if __name__ == "__main__":
    input_folder = "/Users/ryangichuru/Documents/SSD-K/Uni/2nd year/Intro
    output_folder = "/Users/ryangichuru/Documents/SSD-K/Uni/2nd year/Intr

    crop_faces_recursively(input_folder, output_folder, area_threshold=0.
```

### Train Test Split Process

The dataset is split into training, validation and test sets using a 70-15-15 ratio. This is done recursively for each user's folder while preserving the folder structure. The process involves:

1. **Input**:
    - Enhanced dataset with cropped and preprocessed face images
    - Each user has their own folder containing their images
2. **Output Structure**:
    - train/ (70% of data)
    - val/ (15% of data)
    - test/ (15% of data)

- Each split maintains user subfolders

3. **Key Features**:

- Random shuffling with fixed seed for reproducibility [@Web: https://scikit-learn.org/stable/common_pitfalls.html#controlling-randomness]
- Handles file collisions with UUID suffixes [@Web: https://docs.python.org/3/library/uuid.html]
- Preserves folder hierarchy [@Web: https://docs.python.org/3/library/os.html#os.makedirs]
- Supports both copy and move operations

In [ ]:
```python
#!/usr/bin/env python3
import os
import shutil
import random
import uuid

def gather_images_recursively(folder_path):
    """
    Recursively collects *all* .png/.jpg/.jpeg/.bmp/.webp file paths
    under `folder_path`. Returns a list of absolute paths.
    """
    all_paths = []
    for root, dirs, files in os.walk(folder_path):
        for f in files:
            if f.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.web
                full_path = os.path.join(root, f)
                all_paths.append(full_path)
    return all_paths

def ensure_unique_filename(base_name, existing_files):
    """
    If base_name is already in existing_files, generate a unique name by
    appending a short UUID suffix.
    Returns a filename guaranteed not in existing_files.
    """
    if base_name not in existing_files:
        return base_name
    # Collision: add suffix
    name_part, ext = os.path.splitext(base_name)
    while True:
        suffix = str(uuid.uuid4())[:8]  # short random suffix
        candidate = f"{name_part}_{suffix}{ext}"
        if candidate not in existing_files:
            return candidate

def split_dataset(
    input_dir,
    output_dir,
    train_ratio=0.70,
    val_ratio=0.15,
    test_ratio=0.15,
    copy_files=True,
    seed=42
):
    """
    Recursively splits each user's folder into train/val/test sets,
```

```
    keeping each image in a subfolder with the same name as the image.
    """

    if abs((train_ratio + val_ratio + test_ratio) - 1.0) > 1e-5:
        raise ValueError("train_ratio + val_ratio + test_ratio must equal

    random.seed(seed)

    # Create main subfolders: train, val, test
    os.makedirs(output_dir, exist_ok=True)

    train_dir = os.path.join(output_dir, "train")
    val_dir   = os.path.join(output_dir, "val")
    test_dir  = os.path.join(output_dir, "test")
    os.makedirs(train_dir, exist_ok=True)
    os.makedirs(val_dir,   exist_ok=True)
    os.makedirs(test_dir,  exist_ok=True)

    # Identify user-level subfolders. E.g. "Saurish"
    user_folders = [
        d for d in os.listdir(input_dir)
        if os.path.isdir(os.path.join(input_dir, d))
    ]

    for user_name in user_folders:
        user_input_path = os.path.join(input_dir, user_name)
        # Recursively gather images from sub-subfolders
        all_img_paths = gather_images_recursively(user_input_path)
        random.shuffle(all_img_paths)

        total_files = len(all_img_paths)
        train_count = int(total_files * 0.70)  # 70% for training
        val_count = int(total_files * 0.15)    # 15% for validation
        test_count = total_files - (train_count + val_count)  # Remaining

        train_paths = all_img_paths[:train_count]
        val_paths   = all_img_paths[train_count:train_count + val_count]
        test_paths  = all_img_paths[train_count + val_count:]

        # Create user subfolder in train/val/test
        user_train_dir = os.path.join(train_dir, user_name)
        user_val_dir   = os.path.join(val_dir,   user_name)
        user_test_dir  = os.path.join(test_dir,  user_name)
        os.makedirs(user_train_dir, exist_ok=True)
        os.makedirs(user_val_dir,   exist_ok=True)
        os.makedirs(user_test_dir,  exist_ok=True)

        def transfer_file(src, dst_folder):
            base_name = os.path.basename(src)
            dst_path = os.path.join(dst_folder, base_name)

            if copy_files:
                shutil.copy2(src, dst_path)
            else:
                shutil.move(src, dst_path)

        for p in train_paths:
            transfer_file(p, user_train_dir)

        for p in val_paths:
```

```
            transfer_file(p, user_val_dir)

        for p in test_paths:
            transfer_file(p, user_test_dir)

        print(f"{user_name}: total={total_files} -> "
              f"train={len(train_paths)}, val={len(val_paths)}, test={len

    print("Done splitting!")
    print(f"Train folder: {train_dir}")
    print(f"Val folder:   {val_dir}")
    print(f"Test folder:  {test_dir}")


if __name__ == "__main__":
    # Adjust input_folder to our dataset's "parent" folder:
    input_folder  = "/Users/ryangichuru/Documents/SSD-K/Uni/2nd year/Intr
    output_folder = "/Users/ryangichuru/Documents/SSD-K/Uni/2nd year/Intr

    split_dataset(
        input_dir=input_folder,
        output_dir=output_folder,
        copy_files=True,
        seed=42
    )
```

In [ ]:

```
import os
import shutil

def organize_images_into_subfolders(base_path):
    """
    Organizes images in the given path into individual subfolders.
    Each subfolder will have the same name as the image (without extensio
    """
    # Walk through all directories
    for root, dirs, files in os.walk(base_path):
        for file in files:
            if file.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.
                # Get full path of the image
                file_path = os.path.join(root, file)

                # Get file name without extension
                file_name = os.path.splitext(file)[0]

                # Create new subfolder path
                new_folder = os.path.join(root, file_name)

                # Create subfolder if it doesn't exist
                os.makedirs(new_folder, exist_ok=True)

                # Move image to new subfolder
                new_file_path = os.path.join(new_folder, file)
                if file_path != new_file_path:  # Avoid moving if already
                    shutil.move(file_path, new_file_path)
                    print(f"Moved {file} to {new_folder}")

if __name__ == "__main__":
    base_path = "/Users/ryangichuru/Documents/SSD-K/Uni/2nd year/Intro to
    organize_images_into_subfolders(base_path)
```

# We will apply sharpening and blurring to our dataset now but this will not be used as of the moment

In [ ]:
```python
#!/usr/bin/env python3
"""
apply_dip_to_dataset.py

This script applies sharpening and blurring filters to facial recognition
"""

import os
import cv2
import random
import shutil
import numpy as np
import matplotlib.pyplot as plt

SHOW_SAMPLES = True
NUM_SAMPLES_TO_SHOW = 5

def sharpen_image(bgr_img):
    """Applies Laplacian sharpening filter to BGR image"""
    sharpen_kernel = np.array([[0, -1, 0],
                               [-1, 5, -1],
                               [0, -1, 0]], dtype=np.float32)
    return cv2.filter2D(bgr_img, ddepth=-1, kernel=sharpen_kernel)

def blur_image(bgr_img, ksize=8):  # Increased kernel size from 5 to 11
    """Applies Gaussian blur to BGR image"""
    return cv2.GaussianBlur(bgr_img, (ksize, ksize), 0)

def gather_images_recursively(folder_path, exts=('.png', '.jpg', '.jpeg')
    """Recursively collects image paths"""
    all_paths = []
    for root, dirs, files in os.walk(folder_path):
        for f in files:
            if f.lower().endswith(exts):
                all_paths.append(os.path.join(root, f))
    return all_paths

def apply_dip_to_dataset(input_dir, output_dir):
    """Applies filters and saves results preserving folder structure"""
    all_img_paths = gather_images_recursively(input_dir)
    if not all_img_paths:
        print(f"No images found in {input_dir}")
        return

    sample_paths = []
    if SHOW_SAMPLES:
        random.shuffle(all_img_paths)
        sample_paths = all_img_paths[:NUM_SAMPLES_TO_SHOW]

    for img_path in all_img_paths:
        rel_path = os.path.relpath(img_path, input_dir)
        bgr_img = cv2.imread(img_path)
        if bgr_img is None:
            print(f"Skipping unreadable file: {img_path}")
```

```
            continue

        sharpened = sharpen_image(bgr_img)
        blurred = blur_image(bgr_img, ksize=11)

        # Get filename without extension
        filename = os.path.splitext(os.path.basename(img_path))[0]

        # Create individual folders for each image
        sharpen_folder = os.path.join(output_dir, "sharpened", filename)
        blur_folder = os.path.join(output_dir, "blurred", filename)

        os.makedirs(sharpen_folder, exist_ok=True)
        os.makedirs(blur_folder, exist_ok=True)

        # Save images in their individual folders
        sharpen_out = os.path.join(sharpen_folder, f"{filename}.png")
        blur_out = os.path.join(blur_folder, f"{filename}.png")

        cv2.imwrite(sharpen_out, sharpened)
        cv2.imwrite(blur_out, blurred)

    print("Done applying filters to dataset.")
    print(f"Sharpened results in: {os.path.join(output_dir, 'sharpened')}
    print(f"Blurred results in: {os.path.join(output_dir, 'blurred')}")

    if SHOW_SAMPLES and sample_paths:
        print(f"Showing {len(sample_paths)} sample comparisons")
        for sp in sample_paths:
            bgr = cv2.imread(sp)
            if bgr is None:
                continue

            sharpened = sharpen_image(bgr)
            blurred = blur_image(bgr, ksize=11)

            rgb_orig = cv2.cvtColor(bgr, cv2.COLOR_BGR2RGB)
            rgb_sharpened = cv2.cvtColor(sharpened, cv2.COLOR_BGR2RGB)
            rgb_blurred = cv2.cvtColor(blurred, cv2.COLOR_BGR2RGB)

            fig, axes = plt.subplots(1, 3, figsize=(12, 4))
            axes[0].imshow(rgb_orig)
            axes[0].set_title("Original")
            axes[0].axis('off')

            axes[1].imshow(rgb_sharpened)
            axes[1].set_title("Sharpened")
            axes[1].axis('off')

            axes[2].imshow(rgb_blurred)
            axes[2].set_title("Blurred")
            axes[2].axis('off')

            plt.suptitle(os.path.basename(sp))
            plt.show()

if __name__ == "__main__":
    input_dir = "/Users/ryangichuru/Documents/SSD-K/Uni/2nd year/Intro to
    output_dir = "/Users/ryangichuru/Documents/SSD-K/Uni/2nd year/Intro t
```

```
    apply_dip_to_dataset(input_dir, output_dir)
```

### Data Augmentation Process

After applying basic image processing techniques (sharpening and blurring), we implement additional data augmentation separately from the main transfer learning pipeline. While this creates a longer workflow, it provides several key benefits:

1. **Better Process Understanding**:
   - Separating augmentation helps us understand exactly how the data is being transformed
   - Allows visual inspection of augmented images before training
   - Provides more control over the augmentation parameters
2. **Augmentation Techniques Applied**:
   - Rotation (±10 degrees)
   - Width/height shifts (±10%)
   - Zoom variations (±10%)
   - Brightness adjustments (±10%)
   - Horizontal flips
   - Nearest neighbor fill mode
3. **Control Benefits**:
   - Can verify quality of augmented images
   - Ability to adjust parameters based on visual results
   - Ensures consistent augmentation across training runs
   - Maintains data integrity through the process

In [ ]:
```
#!/usr/bin/env python3
"""
augment_dataset.py
Ensures exactly 3x augmented images per person within train/val/test spli
"""

import os
import shutil
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator

def augment_person_folder(datagen, person_dir, output_dir, multiplier=3):
    """
    Augments images for a single person's folder to ensure exactly multip
    Saves directly in subfolders under the person's directory.
    """
    person_name = os.path.basename(person_dir)
    person_output_dir = os.path.join(output_dir, person_name)
    os.makedirs(person_output_dir, exist_ok=True)

    # Count original images
    original_images = []
    for root, _, files in os.walk(person_dir):
        for f in files:
```

```
            if f.lower().endswith(('.png', '.jpg', '.jpeg')):
                original_images.append(os.path.join(root, f))

    n_original = len(original_images)
    n_to_generate = n_original * multiplier

    print(f"\nProcessing {person_name}:")
    print(f"Original images: {n_original}")
    print(f"To generate: {n_to_generate}")

    if n_original > 0:
        # Setup generator for this person
        person_generator = datagen.flow_from_directory(
            directory=os.path.dirname(person_dir),
            classes=[person_name],
            target_size=(180, 180),
            batch_size=1,
            shuffle=True,
            save_to_dir=None
        )

        # Generate augmented images
        for i in range(n_to_generate):
            batch = next(person_generator)
            img = batch[0][0]

            # Create sequential numbered folder directly under person's d
            folder_name = f"{i+1}"  # Just the number
            img_folder = os.path.join(person_output_dir, folder_name)
            os.makedirs(img_folder, exist_ok=True)

            # Save image with same name as folder
            img_path = os.path.join(img_folder, f"{folder_name}.jpg")
            tf.keras.preprocessing.image.save_img(img_path, img)

            if (i + 1) % 10 == 0:
                print(f"Generated {i + 1}/{n_to_generate} augmented image

    final_count = sum(1 for _ in os.walk(person_output_dir))
    print(f"Final folder count for {person_name}: {final_count}")
    return final_count

def main():
    # Paths
    base_input_dir = "/Users/ryangichuru/Documents/SSD-K/Uni/2nd year/Int
    base_output_dir = "/Users/ryangichuru/Documents/SSD-K/Uni/2nd year/In

    # Clear output dir if it exists
    if os.path.exists(base_output_dir):
        shutil.rmtree(base_output_dir)
    os.makedirs(base_output_dir)

    # Augmentation settings optimized for facial recognition
    datagen = ImageDataGenerator(
        rotation_range=10,
        width_shift_range=0.1,
        height_shift_range=0.1,
        zoom_range=0.1,
        brightness_range=[0.9, 1.1],
        horizontal_flip=True,
```

```
        fill_mode='nearest'
    )

    # Process each split (train/val/test)
    for split in ['train', 'val', 'test']:
        split_input_dir = os.path.join(base_input_dir, split)
        split_output_dir = os.path.join(base_output_dir, split)
        os.makedirs(split_output_dir, exist_ok=True)

        print(f"\nProcessing {split} split:")
        total_split_images = 0

        # Process each person within the split
        for person_name in os.listdir(split_input_dir):
            person_dir = os.path.join(split_input_dir, person_name)
            if os.path.isdir(person_dir):
                count = augment_person_folder(
                    datagen,
                    person_dir,
                    split_output_dir,
                    multiplier=3
                )
                total_split_images += count

        print(f"Total {split} folders after augmentation: {total_split_im

    print("\nAugmentation complete!")
    print(f"Output directory: {base_output_dir}")

if __name__ == "__main__":
    main()
```

In [ ]:

```
#!/usr/bin/env python3
"""
augment_dataset.py
Ensures exactly 3x augmented images per person within train/val/test spli
"""

import os
import shutil
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator

def augment_person_folder(datagen, person_dir, output_dir, multiplier=3):
    """
    Augments images for a single person's folder to ensure exactly multip
    Only saves the augmented versions.
    """
    person_name = os.path.basename(person_dir)
    person_output_dir = os.path.join(output_dir, person_name)
    os.makedirs(person_output_dir, exist_ok=True)

    # Count original images
    original_images = []
    for root, _, files in os.walk(person_dir):
        for f in files:
            if f.lower().endswith(('.png', '.jpg', '.jpeg')):
                original_images.append(os.path.join(root, f))

    n_original = len(original_images)
```

```
    n_to_generate = n_original * multiplier  # Generate multiplier times

    print(f"\nProcessing {person_name}:")
    print(f"Original images: {n_original}")
    print(f"To generate: {n_to_generate}")

    if n_original > 0:
        # Setup generator for this person
        person_generator = datagen.flow_from_directory(
            directory=os.path.dirname(person_dir),
            classes=[person_name],
            target_size=(180, 180),
            batch_size=1,
            shuffle=True,
            save_to_dir=None
        )

        # Generate augmented images
        for i in range(n_to_generate):
            batch = next(person_generator)
            img = batch[0][0]

            # Create unique folder and save image
            aug_name = f"aug_{person_name}_{i+1}"
            aug_folder = os.path.join(person_output_dir, aug_name)
            os.makedirs(aug_folder, exist_ok=True)

            # Save the augmented image
            img_path = os.path.join(aug_folder, f"{aug_name}.jpg")
            tf.keras.preprocessing.image.save_img(img_path, img)

            if (i + 1) % 10 == 0:
                print(f"Generated {i + 1}/{n_to_generate} augmented image

    final_count = sum(1 for _ in os.walk(person_output_dir))
    print(f"Final folder count for {person_name}: {final_count}")
    return final_count

def main():
    # Paths
    base_input_dir = "/Users/ryangichuru/Documents/SSD-K/Uni/2nd year/Int
    base_output_dir = "/Users/ryangichuru/Documents/SSD-K/Uni/2nd year/In

    # Clear output dir if it exists
    if os.path.exists(base_output_dir):
        shutil.rmtree(base_output_dir)
    os.makedirs(base_output_dir)

    # Augmentation settings optimized for facial recognition
    datagen = ImageDataGenerator(
        rotation_range=10,
        width_shift_range=0.1,
        height_shift_range=0.1,
        zoom_range=0.1,
        brightness_range=[0.9, 1.1],
        horizontal_flip=True,
        fill_mode='nearest'
    )

    # Process each split (train/val/test)
```

```
    for split in ['train', 'val', 'test']:
        split_input_dir = os.path.join(base_input_dir, split)
        split_output_dir = os.path.join(base_output_dir, split)
        os.makedirs(split_output_dir, exist_ok=True)

        print(f"\nProcessing {split} split:")
        total_split_images = 0

        # Process each person within the split
        for person_name in os.listdir(split_input_dir):
            person_dir = os.path.join(split_input_dir, person_name)
            if os.path.isdir(person_dir):
                count = augment_person_folder(
                    datagen,
                    person_dir,
                    split_output_dir,
                    multiplier=3
                )
                total_split_images += count

        print(f"Total {split} folders after augmentation: {total_split_im

    print("\nAugmentation complete!")
    print(f"Output directory: {base_output_dir}")

if __name__ == "__main__":
    main()
```

## Step 2 Data Augmentation Process

After applying basic image processing techniques (sharpening and blurring), we implement additional data augmentation separately from the main transfer learning pipeline. While this creates a longer workflow, it provides several key benefits:

1. **Better Process Understanding**:
    - Separating augmentation helps us understand exactly how the data is being transformed
    - Allows visual inspection of augmented images before training
    - Provides more control over the augmentation parameters
2. **Augmentation Techniques Applied**:
    - Rotation (±10 degrees)
    - Width/height shifts (±10%)
    - Zoom variations (±10%)
    - Brightness adjustments (±10%)
    - Horizontal flips
    - Nearest neighbor fill mode
3. **Control Benefits**:
    - Can verify quality of augmented images
    - Ability to adjust parameters based on visual results
    - Ensures consistent augmentation across training runs
    - Maintains data integrity through the process

In [ ]:
```
#!/usr/bin/env python3
"""
augment_dataset.py
Ensures exactly 3x augmented images per person within train/val/test spli
"""

import os
import shutil
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator


def augment_person_folder(datagen, person_dir, output_dir, multiplier=3):
    """
    Augments images for a single person's folder to ensure exactly multip
    Only saves the augmented versions.
    """
    person_name = os.path.basename(person_dir)
    person_output_dir = os.path.join(output_dir, person_name)
    os.makedirs(person_output_dir, exist_ok=True)

    # Count original images
    original_images = []
    for root, _, files in os.walk(person_dir):
        for f in files:
            if f.lower().endswith(('.png', '.jpg', '.jpeg')):
                original_images.append(os.path.join(root, f))

    n_original = len(original_images)
    n_to_generate = n_original * multiplier  # Generate multiplier times

    print(f"\nProcessing {person_name}:")
    print(f"Original images: {n_original}")
    print(f"To generate: {n_to_generate}")

    if n_original > 0:
        # Setup generator for this person
        person_generator = datagen.flow_from_directory(
            directory=os.path.dirname(person_dir),
            classes=[person_name],
            target_size=(180, 180),
            batch_size=1,
            shuffle=True,
            save_to_dir=None
        )

        # Generate augmented images
        for i in range(n_to_generate):
            batch = next(person_generator)
            img = batch[0][0]

            # Create unique folder and save image
            aug_name = f"aug_{person_name}_{i+1}"
            aug_folder = os.path.join(person_output_dir, aug_name)
            os.makedirs(aug_folder, exist_ok=True)

            # Save the augmented image
            img_path = os.path.join(aug_folder, f"{aug_name}.jpg")
            tf.keras.preprocessing.image.save_img(img_path, img)
```

```
            if (i + 1) % 10 == 0:
                print(f"Generated {i + 1}/{n_to_generate} augmented image

    final_count = sum(1 for _ in os.walk(person_output_dir))
    print(f"Final folder count for {person_name}: {final_count}")
    return final_count


def offline_augment_dataset(
    base_input_dir,
    base_output_dir,
    multiplier=3,
    shear_range=10,
    channel_shift_range=50,
    width_shift_range=0.15,
    height_shift_range=0.15,
    zoom_range=0.1,
    horizontal_flip=True,
    fill_mode='nearest'
):
    """
    For train/val/test in `base_input_dir`, create a mirrored structure i
    `base_output_dir`, and produce new images via the given transformatio
    """
    abs_in = os.path.abspath(base_input_dir)
    abs_out = os.path.abspath(base_output_dir)
    if abs_out.startswith(abs_in):
        raise ValueError(
            f"Output folder '{base_output_dir}' is inside/same as input '
            "Must be distinct to avoid overwriting or scanning itself."
        )

    # Recreate output
    if os.path.exists(base_output_dir):
        shutil.rmtree(base_output_dir)
    os.makedirs(base_output_dir)

    datagen = ImageDataGenerator(
        shear_range=shear_range,
        channel_shift_range=channel_shift_range,
        width_shift_range=width_shift_range,
        height_shift_range=height_shift_range,
        zoom_range=zoom_range,
        horizontal_flip=horizontal_flip,
        fill_mode=fill_mode
    )

    # Creating directories for train/val/test
    for subset_name in ["train", "val", "test"]:
        subset_in = os.path.join(base_input_dir, subset_name)
        if not os.path.isdir(subset_in):
            print(f"Warning: No {subset_name} folder in {base_input_dir},
            continue

        subset_out = os.path.join(base_output_dir, subset_name)
        os.makedirs(subset_out, exist_ok=True)

        # For each class subfolder
        for person_name in os.listdir(subset_in):
```

```
            person_dir = os.path.join(subset_in, person_name)
            if os.path.isdir(person_dir):
                out_dir = os.path.join(subset_out, person_name)
                os.makedirs(out_dir, exist_ok=True)
                augment_person_folder(datagen, person_dir, out_dir, multi

    print(f"\nOffline augmentation complete for '{base_input_dir}'")
    print(f"Augmented data placed in '{base_output_dir}'")


def main():
    # Paths
    base_input_dir = "/Users/ryangichuru/Documents/SSD-K/Uni/2nd year/Int
    base_output_dir = "/Users/ryangichuru/Documents/SSD-K/Uni/2nd year/In

    # Clear output dir if it exists
    if os.path.exists(base_output_dir):
        shutil.rmtree(base_output_dir)
    os.makedirs(base_output_dir)

    # Augmentation settings optimized for facial recognition
    datagen = ImageDataGenerator(
        rotation_range=10,
        width_shift_range=0.1,
        height_shift_range=0.1,
        zoom_range=0.1,
        brightness_range=[0.9, 1.1],
        horizontal_flip=True,
        fill_mode='nearest'
    )

    # Process each split (train/val/test)
    for split in ['train', 'val', 'test']:
        split_input_dir = os.path.join(base_input_dir, split)
        split_output_dir = os.path.join(base_output_dir, split)
        os.makedirs(split_output_dir, exist_ok=True)

        print(f"\nProcessing {split} split:")
        total_split_images = 0

        # Process each person within the split
        for person_name in os.listdir(split_input_dir):
            person_dir = os.path.join(split_input_dir, person_name)
            if os.path.isdir(person_dir):
                count = augment_person_folder(
                    datagen,
                    person_dir,
                    split_output_dir,
                    multiplier=3
                )
                total_split_images += count

        print(f"Total {split} folders after augmentation: {total_split_im

    print("\nAugmentation complete!")
    print(f"Output directory: {base_output_dir}")

if __name__ == "__main__":
    main()
```

### Dataset Rebalancing Process

The goal was to ensure equal representation of each subject in our facial recognition dataset to prevent model bias. Initial analysis revealed uneven distribution of images across subjects, which could lead to the model overfitting to subjects with more training data while underperforming on those with fewer samples. This rebalancing process aims to create a more equitable dataset by:

1. **Counting Images**: Systematically count images for each subject across train/val/test splits
2. **Finding Minimum**: Determine minimum number of images per subject in each split
3. **Random Selection**: Randomly select equal numbers of images per subject
4. **Copying Data**: Create new balanced dataset structure with selected images

#### Key Functions

1. **count_face_images()**: Traverses dataset structure counting images per subject
2. **rebalance_dataset()**: Ensures equal representation by copying minimum number of images
3. **main()**: Orchestrates rebalancing across multiple dataset versions

In [ ]:
```python
import os
import shutil
import random
from collections import defaultdict

def count_face_images(dataset_path):
    """
    Count images with faces for each person in each split.
    Returns a nested dictionary of counts and image paths.
    """
    data = defaultdict(lambda: defaultdict(list))

    for split in ['train', 'val', 'test']:
        split_path = os.path.join(dataset_path, split)
        if not os.path.exists(split_path):
            continue

        for person in os.listdir(split_path):
            person_path = os.path.join(split_path, person)
            if not os.path.isdir(person_path):
                continue

            # Get all image folders
            for img_folder in os.listdir(person_path):
                folder_path = os.path.join(person_path, img_folder)
                if os.path.isdir(folder_path):
                    # Each folder should contain exactly one image
                    images = [f for f in os.listdir(folder_path)
                            if f.lower().endswith(('.jpg', '.jpeg', '.png
                    if images:
                        data[split][person].append(folder_path)

    return data
```

```
def rebalance_dataset(input_path, output_path):
    """
    Rebalance dataset ensuring equal numbers of images per person per spl
    """
    # Get current distribution
    data = count_face_images(input_path)

    # Find minimum counts for each split
    min_counts = {}
    for split in ['train', 'val', 'test']:
        if split in data:
            counts = [len(data[split][person]) for person in data[split]]
            min_counts[split] = min(counts) if counts else 0

    print("\nMinimum counts per split:")
    for split, count in min_counts.items():
        print(f"{split}: {count}")

    # Create output directory structure
    os.makedirs(output_path, exist_ok=True)

    # Rebalance each split
    for split in ['train', 'val', 'test']:
        if split not in min_counts or min_counts[split] == 0:
            continue

        target_count = min_counts[split]
        split_output = os.path.join(output_path, split)
        os.makedirs(split_output, exist_ok=True)

        print(f"\nRebalancing {split} split to {target_count} images per

        for person in data[split]:
            person_images = data[split][person]
            random.shuffle(person_images)  # Randomize selection
            selected_images = person_images[:target_count]

            # Create person directory in output
            person_output = os.path.join(split_output, person)
            os.makedirs(person_output, exist_ok=True)

            print(f"  {person}: {len(selected_images)} images")

            # Copy selected images
            for idx, img_folder in enumerate(selected_images, 1):
                # Create new folder name
                new_folder_name = f"{person}_{idx}"
                new_folder_path = os.path.join(person_output, new_folder_
                os.makedirs(new_folder_path, exist_ok=True)

                # Find and copy the image
                for file in os.listdir(img_folder):
                    if file.lower().endswith(('.jpg', '.jpeg', '.png')):
                        src = os.path.join(img_folder, file)
                        dst = os.path.join(new_folder_path, f"{new_folder
                        shutil.copy2(src, dst)
                        break

def main():
```

```
    base_dir = "/Users/ryangichuru/Documents/SSD-K/Uni/2nd year/Intro to

    # Process each dataset
    datasets = ['Augmented Output', 'Output Data', 'Augmented Output2']

    for dataset in datasets:
        input_path = os.path.join(base_dir, 'Rebalanced Output', dataset)
        output_path = os.path.join(base_dir, 'Final Balanced Output', dat

        if os.path.exists(input_path):
            print(f"\nProcessing dataset: {dataset}")
            rebalance_dataset(input_path, output_path)

if __name__ == "__main__":
    random.seed(42)  # For reproducibility
    main()
```

## Face Detection Pipeline

Note

This is our second implementation aimed at improving face-to-image ratio after our initial attempts produced suboptimal results. The enhanced pipeline includes more aggressive cropping and better handling of distant faces.

## Overview

This pipeline uses **MTCNN** (Multi-task Cascaded Convolutional Networks) to detect and process faces in our dataset. It standardizes face images through consistent detection, cropping, and resizing.

## Key Features

1. **Robust Face Detection**
    - Uses MTCNN's deep learning architecture
    - Filters low confidence detections (<0.9 threshold)
2. **Distant Face Handling**
    - Identifies faces with small face-to-image ratio (<0.1)
    - Attempts zoom-in via margin cropping
    - Logs paths to 'far_faces.txt' for review
3. **Standardization**
    - Crops to detected face region
    - Resizes all faces to 180x180px
    - Organizes into class-specific folders
4. **Quality Control**
    - Tracks problematic images
    - Enables manual review of edge cases
    - Maintains consistent output quality

In [ ]:
```
%pip install mtcnn opencv-python pillow numpy
```

In [ ]:
```
import cv2
import numpy as np
from mtcnn import MTCNN
import os
from PIL import Image
import matplotlib.pyplot as plt

def process_dataset(input_dir, output_dir):
    detector = MTCNN()
    far_faces = []
    far_face_images = []

    # Create output directories for each split
    splits = ['train', 'test', 'val']
    classes = ['Anh', 'Ryan', 'Saurish']

    for split in splits:
        for class_name in classes:
            os.makedirs(os.path.join(output_dir, split, class_name), exis

    # Process each split
    for split in splits:
        split_path = os.path.join(input_dir, split)

        for class_name in classes:
            class_path = os.path.join(split_path, class_name)
            if not os.path.isdir(class_path):
                continue

            # Process each image folder
            for img_folder in os.listdir(class_path):
                folder_path = os.path.join(class_path, img_folder)
                if not os.path.isdir(folder_path):
                    continue

                # Get the image from the folder
                img_files = [f for f in os.listdir(folder_path) if f.ends
                if not img_files:
                    continue

                img_path = os.path.join(folder_path, img_files[0])
                img = cv2.imread(img_path)
                if img is None:
                    continue

                rgb_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                faces = detector.detect_faces(rgb_img)

                if not faces:
                    print(f"No face detected: {img_path}")
                    continue

                for face in faces:
                    confidence = face['confidence']
                    if confidence < 0.9:
                        continue
```

```
                    x, y, w, h = face['box']
                    face_area = w * h
                    img_area = img.shape[0] * img.shape[1]
                    face_ratio = face_area / img_area

                    # Check if face is too small/far
                    if face_ratio < 0.1:
                        far_faces.append(img_path)
                        far_face_images.append(rgb_img)
                        # Attempt to zoom
                        margin = int(max(w, h) * 0.5)
                        x1 = max(0, x - margin)
                        y1 = max(0, y - margin)
                        x2 = min(img.shape[1], x + w + margin)
                        y2 = min(img.shape[0], y + h + margin)
                        face_img = img[y1:y2, x1:x2]
                    else:
                        face_img = img[y:y+h, x:x+w]

                    # Resize to standard size
                    face_img = cv2.resize(face_img, (180, 180))

                    # Save processed image
                    output_path = os.path.join(output_dir, split, class_n
                    cv2.imwrite(output_path, face_img)

    # Save list of far faces
    with open('far_faces.txt', 'w') as f:
        f.write('\n'.join(far_faces))

    # Display all distant faces
    if far_face_images:
        n_images = len(far_face_images)
        n_cols = 5
        n_rows = (n_images + n_cols - 1) // n_cols

        plt.figure(figsize=(20, 4*n_rows))
        for i, img in enumerate(far_face_images):
            plt.subplot(n_rows, n_cols, i+1)
            plt.imshow(img)
            plt.axis('off')
            plt.title(f'Distant Face {i+1}')
        plt.tight_layout()
        plt.show()

    return len(far_faces)

if __name__ == "__main__":
    # Original data path
    original_data_dir = "/Users/ryangichuru/Documents/SSD-K/Uni/2nd year/

    # Already augmented path
    augmented1_dir = "/Users/ryangichuru/Documents/SSD-K/Uni/2nd year/Int

    # New augmented path
    augmented2_dir = "/Users/ryangichuru/Documents/SSD-K/Uni/2nd year/Int

    output_dir = "/Users/ryangichuru/Documents/SSD-K/Uni/2nd year/Intro t

    # Process each dataset
```

```
    for input_dir in [original_data_dir, augmented1_dir, augmented2_dir]:
        print(f"\nProcessing dataset: {os.path.basename(input_dir)}")
        far_faces = process_dataset(input_dir, output_dir)
        print(f"Found {far_faces} images with distant faces")
```

# Face Detection and Smart Zoom Processing

## Overview

This script implements face detection and intelligent zooming for preprocessing our face recognition dataset. Instead of using raw images, we leverage **MTCNN** (Multi-task Cascaded Convolutional Networks) to detect and extract high-quality face regions.

## Why Smart Zooming?

1. **Quality Control**
   We enforce strict face-to-image ratio bounds (0.15-0.6) to ensure consistent, well-framed faces.
2. **Intelligent Cropping**
   The zoom algorithm maintains face quality while removing excess background.
3. **Robust Detection**
   High confidence threshold (0.95) ensures we only keep clear, unambiguous face detections.

## Key Parameters

- **min_face_ratio**: 0.15 (minimum face-to-image ratio)
- **max_face_ratio**: 0.6 (maximum face-to-image ratio)
- **confidence_threshold**: 0.95 (minimum detection confidence)
- **zoom_margin**: 0.7 (extra margin when zooming)

## Filtering Criteria

Images are skipped if they:

- Have no detectable faces
- Have low confidence detections
- Lose face detection after zoom
- Have poor face ratios post-zoom

In [ ]:
```
import cv2
import numpy as np
from mtcnn import MTCNN
import os
from PIL import Image
```

```
def process_and_zoom_faces(input_dir, output_dir, min_face_ratio=0.15, ma
    detector = MTCNN()

    # Create output structure
    for root, dirs, _ in os.walk(input_dir):
        rel_path = os.path.relpath(root, input_dir)
        out_path = os.path.join(output_dir, rel_path)
        os.makedirs(out_path, exist_ok=True)

    skipped_images = []

    for root, _, files in os.walk(input_dir):
        for img_name in files:
            if not img_name.lower().endswith(('.jpg', '.jpeg', '.png')):
                continue

            rel_path = os.path.relpath(root, input_dir)
            input_path = os.path.join(root, img_name)
            output_path = os.path.join(output_dir, rel_path, img_name)

            img = cv2.imread(input_path)
            if img is None:
                continue

            rgb_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            faces = detector.detect_faces(rgb_img)

            if not faces:
                skipped_images.append(f"No face: {input_path}")
                continue

            # Get largest face
            face = max(faces, key=lambda x: x['box'][2] * x['box'][3])
            if face['confidence'] < 0.95:
                skipped_images.append(f"Low confidence: {input_path}")
                continue

            x, y, w, h = face['box']
            face_area = w * h
            img_area = img.shape[0] * img.shape[1]
            face_ratio = face_area / img_area

            # Calculate zoom if needed
            if face_ratio < min_face_ratio:
                margin = int(max(w, h) * 0.7)
                x1 = max(0, x - margin)
                y1 = max(0, y - margin)
                x2 = min(img.shape[1], x + w + margin)
                y2 = min(img.shape[0], y + h + margin)
                cropped = img[y1:y2, x1:x2]

                # Verify face ratio after zoom
                faces_after = detector.detect_faces(cv2.cvtColor(cropped,
                if not faces_after:
                    skipped_images.append(f"Lost face after zoom: {input_p
                    continue

                face_after = max(faces_after, key=lambda x: x['box'][2] *
                new_ratio = (face_after['box'][2] * face_after['box'][3])
```

```
                if min_face_ratio <= new_ratio <= max_face_ratio:
                    cv2.imwrite(output_path, cropped)
                else:
                    skipped_images.append(f"Bad ratio after zoom: {input_p
            else:
                cv2.imwrite(output_path, img)

    with open('skipped_images.txt', 'w') as f:
        f.write('\n'.join(skipped_images))

if __name__ == "__main__":
    original_data_dir = "/Users/ryangichuru/Documents/SSD-K/Uni/2nd year/I
    augmented1_dir = "/Users/ryangichuru/Documents/SSD-K/Uni/2nd year/Intr
    augmented2_dir = "/Users/ryangichuru/Documents/SSD-K/Uni/2nd year/Intr
    output_dir = "/Users/ryangichuru/Documents/SSD-K/Uni/2nd year/Intro to

    for input_dir in [original_data_dir, augmented1_dir, augmented2_dir]:
        print(f"\nProcessing dataset: {os.path.basename(input_dir)}")
        process_and_zoom_faces(input_dir, output_dir)
```

### Reprocessing Skipped Images

After initial face detection and processing, we reprocess any skipped images to maximize usable data. This separate reprocessing step provides several benefits:

1. **Multiple Processing Attempts**:
   - Gives images multiple chances to pass face detection
   - Helps handle temporary detection failures
   - Maximizes dataset completeness
2. **Adjusted Parameters**:
   - Multiple detection attempts with varying confidence thresholds
   - Refined face ratio calculations
   - Additional margin adjustments for zooming
3. **Quality Control**:
   - Tracks persistently problematic images
   - Maintains high quality standards
   - Ensures consistent face detection across dataset

In [ ]:
```
# Import MTCNN detector
from mtcnn import MTCNN
detector = MTCNN()

# Read skipped images file
with open('skipped_images.txt', 'r') as f:
    skipped_paths = [line.split(': ')[1].strip() for line in f.readlines(

# Try processing skipped images multiple times
for attempt in range(3):
    print(f"\nAttempt {attempt+1} to process skipped images")
    still_skipped = []

    for img_path in skipped_paths:
        try:
```

```
            # Extract output path based on which input dir contains the i
            if original_data_dir in img_path:
                base_dir = original_data_dir
            elif augmented1_dir in img_path:
                base_dir = augmented1_dir
            elif augmented2_dir in img_path:
                base_dir = augmented2_dir
            else:
                still_skipped.append(f"Unknown source dir: {img_path}")
                continue

            rel_path = os.path.relpath(os.path.dirname(img_path), base_di
            img_name = os.path.basename(img_path)
            output_path = os.path.join(output_dir, rel_path, img_name)

            # Create output directory if it doesn't exist
            os.makedirs(os.path.dirname(output_path), exist_ok=True)

            # Process image
            img = cv2.imread(img_path)
            if img is None:
                still_skipped.append(f"Failed to read: {img_path}")
                continue

            rgb_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            faces = detector.detect_faces(rgb_img)

            if not faces:
                still_skipped.append(f"No face: {img_path}")
                continue

            face = max(faces, key=lambda x: x['box'][2] * x['box'][3])
            if face['confidence'] < 0.9: # Lower confidence threshold
                still_skipped.append(f"Low confidence: {img_path}")
                continue

            x, y, w, h = face['box']
            margin = int(max(w, h) * 0.8) # Increased margin
            x1 = max(0, x - margin)
            y1 = max(0, y - margin)
            x2 = min(img.shape[1], x + w + margin)
            y2 = min(img.shape[0], y + h + margin)
            cropped = img[y1:y2, x1:x2]

            cv2.imwrite(output_path, cropped)

        except Exception as e:
            still_skipped.append(f"Error processing {img_path}: {str(e)}"
            continue

    skipped_paths = [path for path in still_skipped if not path.startswit
    print(f"Remaining skipped images: {len(skipped_paths)}")

    if not skipped_paths:
        break

# Write remaining skipped images
with open('skipped_images_final.txt', 'w') as f:
    f.write('\n'.join(still_skipped))
```

In [ ]:

In [ ]: