Phase 1 Face Data Preparation (Detection, Alignment, Cropping)

1 Goal

Convert raw face photos into standardized, high quality 128×128 RGB crops for training.

2 *(Inputs → Outputs)*

Raw LFW images (DATA_ROOT) → Aligned 128×128 faces (ALIGNED_ROOT)

3 *Pipeline*

3.1 Load random image (safe decode via np.fromfile + cv2.imdecode), convert BGR → RGB

3.2 Detect faces & keypoints with MTCNN (detector.detect_faces).

3.3 Select largest face by area when multiple detections.

3.4 Align using eye keypoints: compute roll angle (atan2), rotate with cv2.getRotationMatrix2D + warpAffine.

3.5 Crop expanded box with 20% margin, fallback to box only crop if keypoints missing.

3.6 Resize to (128,128) using cv2.INTER_AREA; save as JPEG to ALIGNED_ROOT.

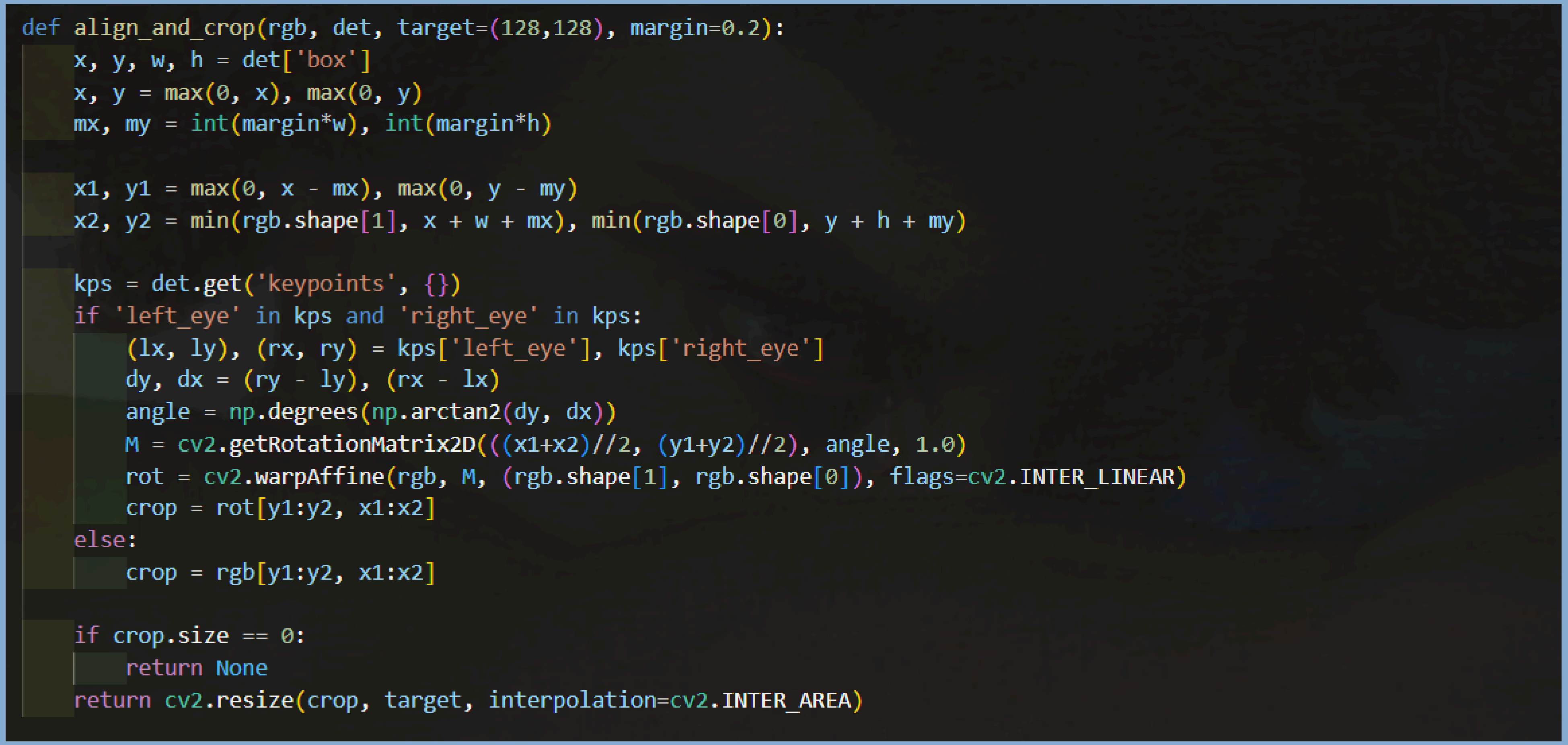
3.7 Visual QA: draw detection boxes; show original vs aligned crops (matplotlib).

4 *Functions & Params*

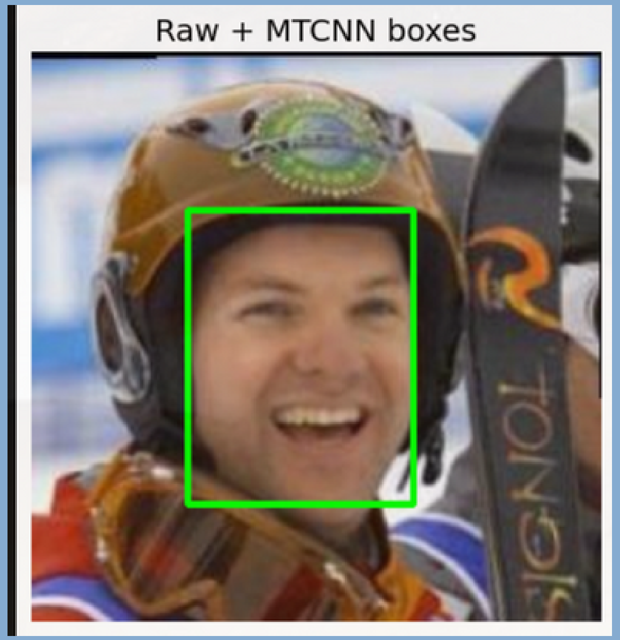
4.1 align_and_crop(rgb, det, target=(128,128), margin=0.2) → rotation aware crop.

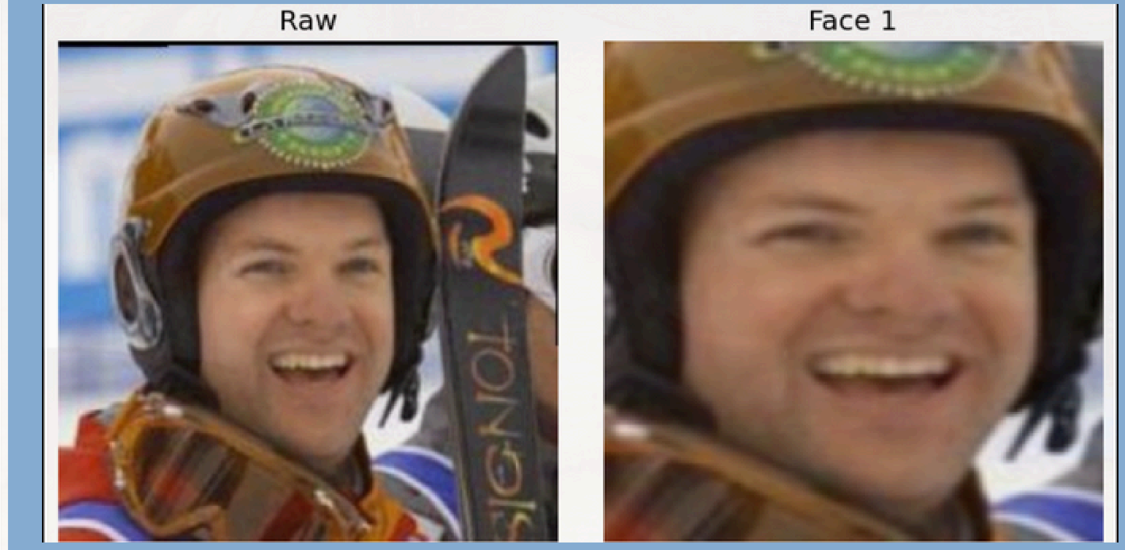
4.2 process_image(path_in, path_out) → End to End per image processing (returns True/False).

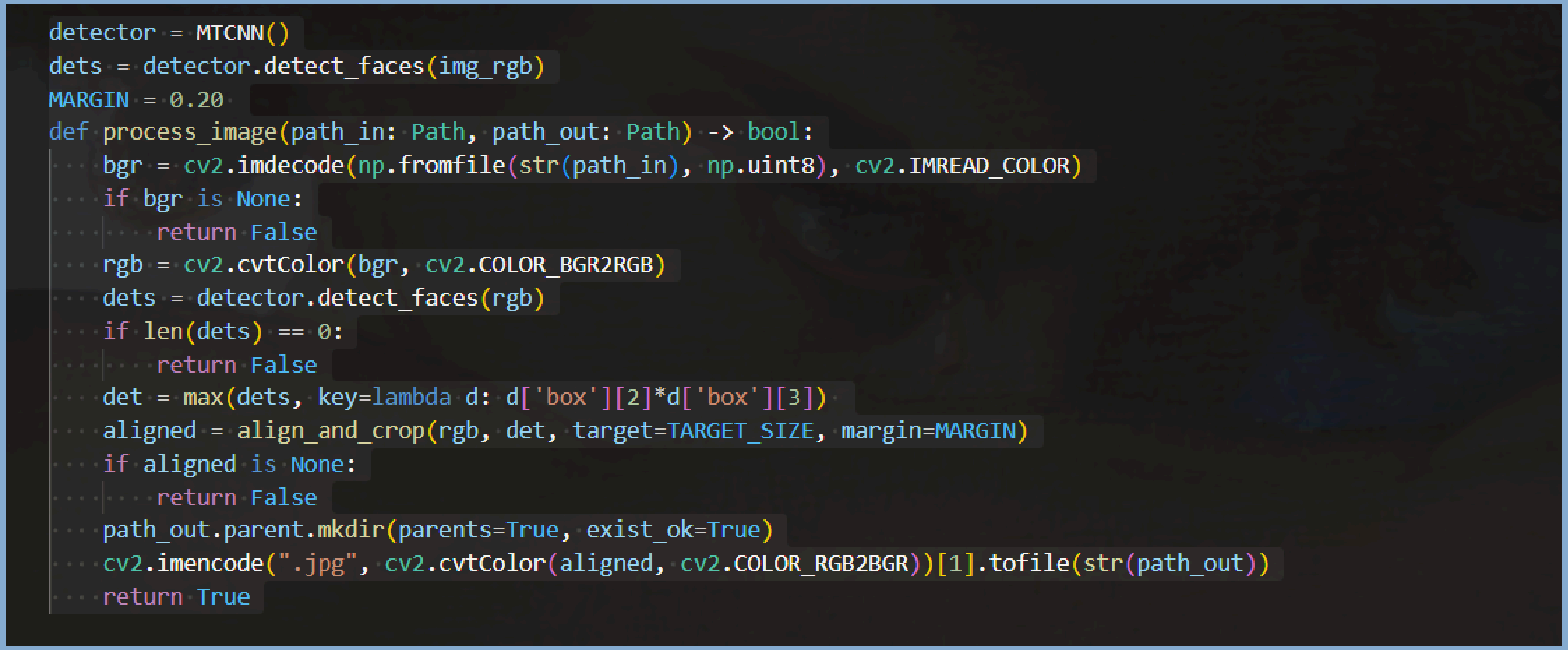
Coding



```python
def align_and_crop(rgb, det, target=(128,128), margin=0.2):
    x, y, w, h = det['box']
    x, y = max(0, x), max(0, y)
    mx, my = int(margin*w), int(margin*h)

    x1, y1 = max(0, x - mx), max(0, y - my)
    x2, y2 = min(rgb.shape[1], x + w + mx), min(rgb.shape[0], y + h + my)

    kps = det.get('keypoints', {})
    if 'left_eye' in kps and 'right_eye' in kps:
        (lx, ly), (rx, ry) = kps['left_eye'], kps['right_eye']
        dy, dx = (ry - ly), (rx - lx)
        angle = np.degrees(np.arctan2(dy, dx))
        M = cv2.getRotationMatrix2D(((x1+x2)//2, (y1+y2)//2), angle, 1.0)
        rot = cv2.warpAffine(rgb, M, (rgb.shape[1], rgb.shape[0]), flags=cv2.INTER_LINEAR)
        crop = rot[y1:y2, x1:x2]
    else:
        crop = rgb[y1:y2, x1:x2]

    if crop.size == 0:
        return None
    return cv2.resize(crop, target, interpolation=cv2.INTER_AREA)
```







```python
detector = MTCNN()
dets = detector.detect_faces(img_rgb)
MARGIN = 0.20
def process_image(path_in: Path, path_out: Path) -> bool:
    bgr = cv2.imdecode(np.fromfile(str(path_in), np.uint8), cv2.IMREAD_COLOR)
    if bgr is None:
        return False
    rgb = cv2.cvtColor(bgr, cv2.COLOR_BGR2RGB)
    dets = detector.detect_faces(rgb)
    if len(dets) == 0:
        return False
    det = max(dets, key=lambda d: d['box'][2]*d['box'][3])
    aligned = align_and_crop(rgb, det, target=TARGET_SIZE, margin=MARGIN)
    if aligned is None:
        return False
    path_out.parent.mkdir(parents=True, exist_ok=True)
    cv2.imencode(".jpg", cv2.cvtColor(aligned, cv2.COLOR_RGB2BGR))[1].tofile(str(path_out))
    return True
```