Kelompok 2

Kuis 2 Praktikum Mengenali Foto e-KTP

TEAM 02

Adinda Wahyu L
2141720096

Alfan Olivan
2141720078

Lukas Valentino
2141720032

M Driya Ananta
2141720224

Wiradarma N B
2141720184

Pre-Processing

Pre-Processing yang digunakan yaitu :

- Grayscale
- Gaussian Blur

Pre-processing

```python
def preprocess_image(image_path):
    """
    Preprocess the input image by converting it to grayscale, applying Gaussian blur, and finding edges.

    Parameters:
    - image_path: Input image file path.

    Returns:
    - edged: Processed image with edges.
    - original: Original input image.
    """

    # Read the input image
    original = cv2.imread(image_path)

    # Convert the image to grayscale
    gray = cv2.cvtColor(original, cv2.COLOR_BGR2GRAY)

    # Apply Gaussian blur
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)

    # Find edges using Canny edge detection
    edged = cv2.Canny(blurred, 75, 200)

    return edged, original
```

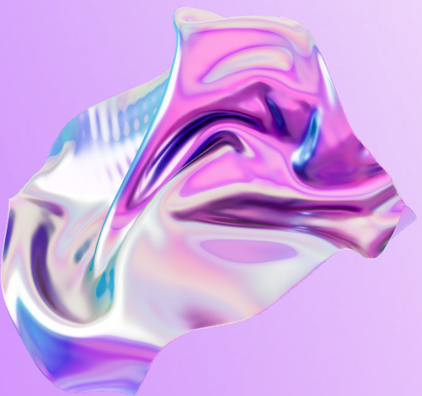
Lokalisasi

Lokalisasi yang digunakan yaitu :

- Canny Edge Detection
- Contours
- Order Point
- Four Point Transform
- Convert Object

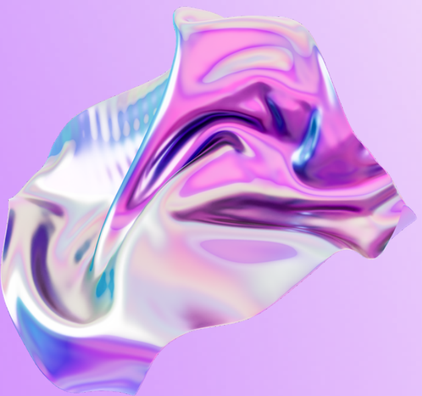
Contours

```python
def find_contours(edged_image, num_contours=5):
    """
    Find contours in the edged image and keep only the largest ones.

    Parameters:
    - edged_image: Edged image (numpy array).
    - num_contours: Number of largest contours to keep.

    Returns:
    - largest_contours: List of largest contours.
    """

    # Find contours in the edged image
    contours = cv2.findContours(edged_image.copy(), cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
    contours = imutils.grab_contours(contours)

    # Sort contours by area in descending order
    largest_contours = sorted(contours, key=cv2.contourArea, reverse=True)[:num_contours]

    return largest_contours
```



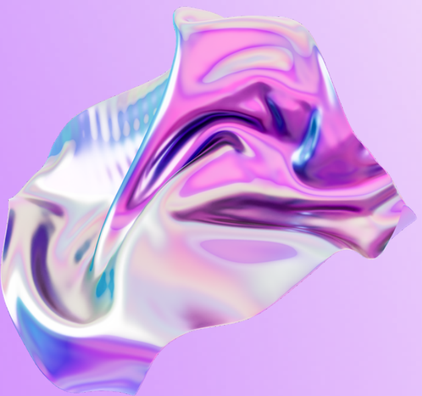
Order Point

```python

#function to order points to proper rectangle
def order_points(pts):
    # initialzie a list of coordinates that will be ordered
    # such that the first entry in the list is the top-left,
    # the second entry is the top-right, the third is the
    # bottom-right, and the fourth is the bottom-left
    rect = np.zeros((4, 2), dtype="float32")

    # the top-left point will have the smallest sum, whereas
    # the bottom-right point will have the largest sum
    s = pts.sum(axis=1)
    rect[0] = pts[np.argmin(s)]
    rect[2] = pts[np.argmax(s)]

    # now, compute the difference between the points, the
    # top-right point will have the smallest difference,
    # whereas the bottom-left will have the largest difference
    diff = np.diff(pts, axis=1)
    rect[1] = pts[np.argmin(diff)]
    rect[3] = pts[np.argmax(diff)]

    # return the ordered coordinates
    return rect
```

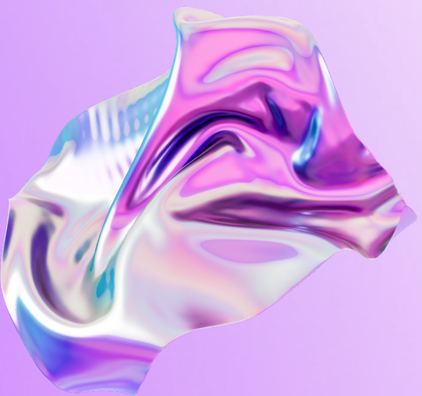


Four Point

```python
def four_point_transform(image, pts):
    # obtain a consistent order of the points and unpack them
    # individually
    rect = order_points(pts)

    # # multiply the rectangle by the original ratio
    # rect *= ratio

    (tl, tr, br, bl) = rect

    # compute the width of the new image, which will be the
    # maximum distance between bottom-right and bottom-left
    # x-coordiates or the top-right and top-left x-coordinates
    widthA = np.sqrt(((br[0] - bl[0]) ** 2) + ((br[1] - bl[1]) ** 2))
    widthB = np.sqrt(((tr[0] - tl[0]) ** 2) + ((tr[1] - tl[1]) ** 2))
    maxWidth = max(int(widthA), int(widthB))

    # compute the height of the new image, which will be the
    # maximum distance between the top-right and bottom-right
    # y-coordinates or the top-left and bottom-left y-coordinates
    heightA = np.sqrt(((tr[0] - br[0]) ** 2) + ((tr[1] - br[1]) ** 2))
    heightB = np.sqrt(((tl[0] - bl[0]) ** 2) + ((tl[1] - bl[1]) ** 2))
    maxHeight = max(int(heightA), int(heightB))

    # now that we have the dimensions of the new image, construct
    # the set of destination points to obtain a "birds eye view",
    # (i.e. top-down view) of the image, again specifying points
    # in the top-left, top-right, bottom-right, and bottom-left
    # order
    dst = np.array([
        [0, 0],
        [maxWidth - 1, 0],
        [maxWidth - 1, maxHeight - 1],
        [0, maxHeight - 1]], dtype="float32")

    # compute the perspective transform matrix and then apply it
    M = cv2.getPerspectiveTransform(rect, dst)
    warped = cv2.warpPerspective(image, M, (maxWidth, maxHeight))

    # return the warped image
    return warped
```



Convert Object



```python
def convert_object(image, screen_size = None, isDebug = False):

    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    gray = cv2.bilateralFilter(gray, 11, 17, 17)  # 11  //TODO
 11 FRO OFFLINE MAY NEED TO TUNE TO 5 FOR ONLINE

    gray = cv2.medianBlur(gray, 5)
    edged = cv2.Canny(gray, 30, 400)

    if isDebug  : cv2_imshow(edged)

    countours, hierarcy = cv2.findContours(edged, cv2.RETR_LIST, cv2.CHAIN_APPROX_NONE)

    if isDebug : print('length of countours ', len(countours))

    imageCopy = image.copy()
    if isDebug : cv2_imshow(cv2.drawContours(imageCopy, countours, -1, (0, 255, 0), 1))


    # approximate the contour
    cnts = sorted(countours, key=cv2.contourArea, reverse=True)
    screenCntList = []
    scrWidths = []
    for cnt in cnts:
        peri = cv2.arcLength(cnt, True)  # cnts[1] always rectangle O.o
        approx = cv2.approxPolyDP(cnt, 0.02 * peri, True)
        screenCnt = approx
        # print(len(approx))

        if (len(screenCnt) == 4):

            (X, Y, W, H) = cv2.boundingRect(cnt)
            # print('X Y W H', (X, Y, W, H))
            screenCntList.append(screenCnt)
            scrWidths.append(W)

    print('Screens found :', len(screenCntList))
    print('Screen Dimentions', scrWidths)

    screenCntList, scrWidths = findLargestCountours(screenCntList, scrWidths)

    if not len(screenCntList) >=2: #there is no rectangle found
        return None
    elif scrWidths[0] != scrWidths[1]: #mismatch in rect
        return None

    imageWithRectangle = cv2.drawContours(image.copy(), [screenCntList[0]], -1, (0, 255, 0),
3)
    if isDebug :   cv2_imshow(cv2.drawContours(image.copy(), [screenCntList[0]], -1, (0, 255
, 0), 3))

    pts = screenCntList[0].reshape(4, 2)
    print('Found bill rectagle at ', pts)
    rect = order_points(pts)
    print(rect)

    warped = four_point_transform(image, pts)

    warp = cv2.cvtColor(warped, cv2.COLOR_BGR2GRAY)
    warp = exposure.rescale_intensity(warp, out_range=(0, 255))

    if(isDebug):
        cv2_imshow(image)
        cv2_imshow(warp)

    return warped, imageWithRectangle, edged
```

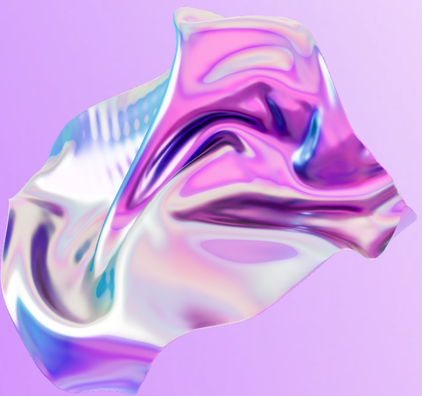
Segmentasi

Segmentasi yang digunakan yaitu :

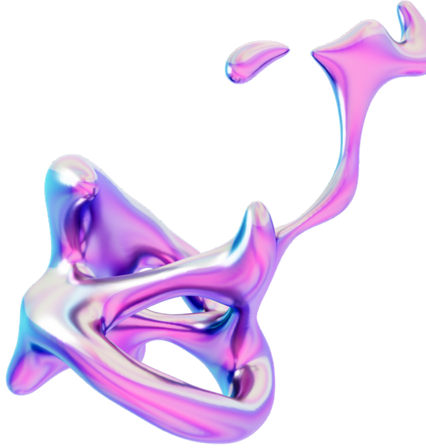
- Crop Image With Max Contours
- Detect & Crop Face

Crop Image With Max Contour

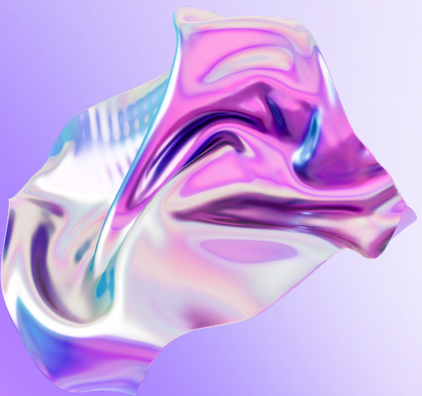
```python
def crop_image_with_max_contours(original_image, contours):
    """
    Crop the original image based on the maximum contours.

    Parameters:
    - original_image: Original input image (numpy array).
    - contours: List of contours.

    Returns:
    - cropped_image: Cropped image.
    """

    # Create a mask for the contours
    mask = np.zeros_like(original_image)

    # Draw the contours on the mask
    cv2.drawContours(mask, contours, -1, (255, 255, 255), thickness=cv2.FILLED)

    # Bitwise AND operation to get the region of interest (ROI)
    masked_image = cv2.bitwise_and(original_image, mask)

    # Find the bounding box of the contours
    x, y, w, h = cv2.boundingRect(np.vstack(contours))

    # Crop the original image based on the bounding box
    cropped_image = original_image[y:y+h, x:x+w]

    return cropped_image
```



Detect & Crop Faces

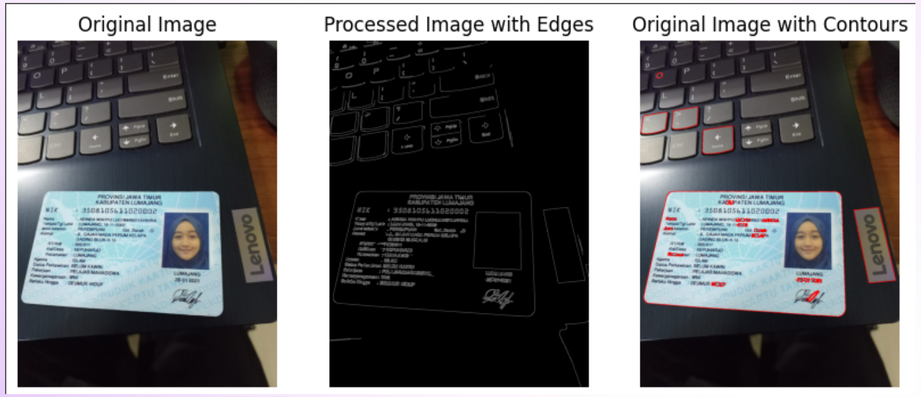


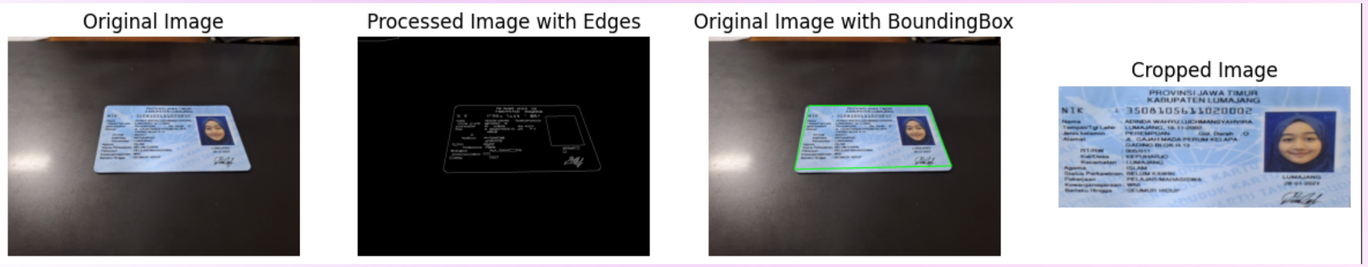
```python
def detect_and_crop_faces(img):
    # Convert into grayscale
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Load the cascade
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_alt2.xml')

    # Detect faces
    faces = face_cascade.detectMultiScale(gray, 1.1, 4)

    # Draw rectangle around the first detected face and crop it
    if len(faces) > 0:
        x, y, w, h = faces[0]
        cv2.rectangle(img, (x, y), (x+w, y+h), (0, 0, 255), 2)
        cropped_face = img[y:y+h, x:x+w]
    else:
        # No face detected
        cropped_face = None


# Return the original image with rectangles drawn around the first detected face and the cropped face
    return img, cropped_face
```

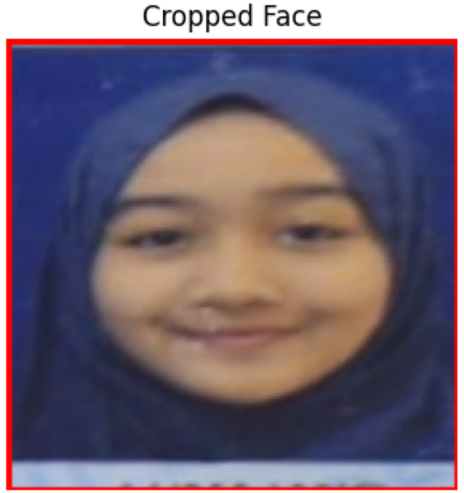


Testing Model and Predict Model

Testing Adinda Wahyu

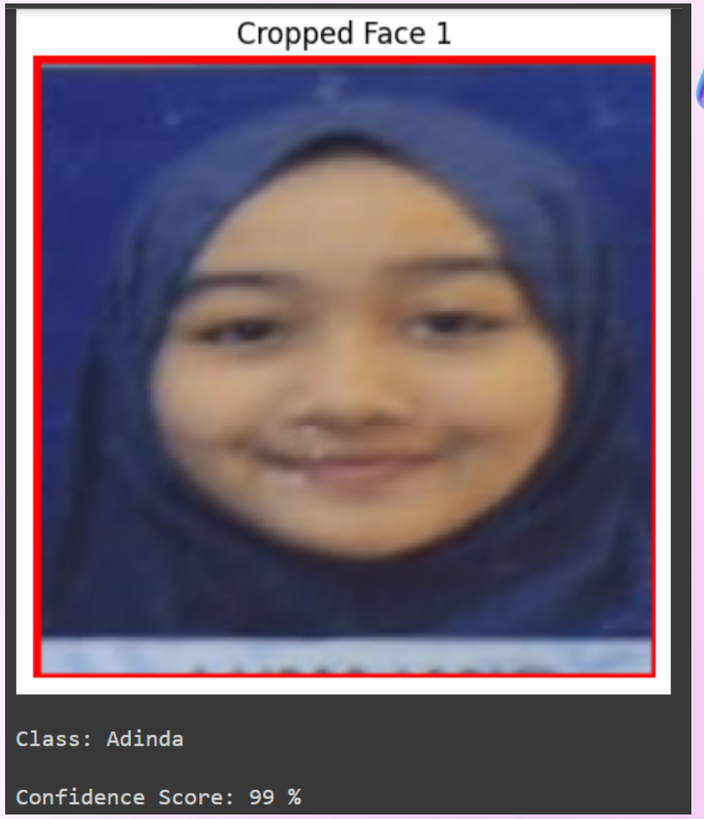






Predict Adinda Wahyu

```python
# Predict the model
prediction = model.predict(image)
index = np.argmax(prediction)
class_name = class_names[index]
confidence_score = prediction[0][index]

# Display the cropped face using plt
plt.imshow(cv2.cvtColor(cropped_face, cv2.COLOR_BGR2RGB))
plt.title('Cropped Face 2')
plt.axis('off')
plt.show()

# Print prediction and confidence score
print("\nClass:", class_name[2:])
print("Confidence Score:", str(np.round(confidence_score * 100))[:-2], "%")
```



Class: Adinda
Confidence Score: 99 %
Terdeteksi : Adinda

Testing Alfan Olivan

Original Image

Processed Image with Edges

Original Image with Contours

Original Image

Processed Image with Edges

Original Image with BoundingBox

Cropped Image

Original Image with Face

Cropped Face

Predict Alfan Olivan

```python
# Predict the model
prediction = model.predict(image)
index = np.argmax(prediction)
class_name = class_names[index]
confidence_score = prediction[0][index]

# Display the cropped face using plt
plt.imshow(cv2.cvtColor(cropped_face, cv2.COLOR_BGR2RGB))
plt.title('Cropped Face 2')
plt.axis('off')
plt.show()

# Print prediction and confidence score
print("\nClass:", class_name[2:])
print("Confidence Score:", str(np.round(confidence_score * 100))[:-2], "%")
```

Class: Alfan Olivan
Confidence Score: 60 %
Terdeteksi : Driya

Testing Lukas

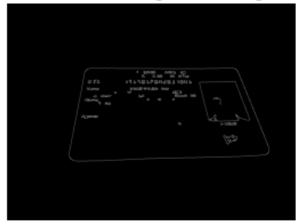
Original Image

Processed Image with Edges

Original Image with Contours

Original Image

Processed Image with Edges



Original Image with BoundingBox

Cropped Image

Original Image with Face

Cropped Face

Predict Lukas Valentino

```python
# Predict the model
prediction = model.predict(image)
index = np.argmax(prediction)
class_name = class_names[index]
confidence_score = prediction[0][index]

# Display the cropped face using plt
plt.imshow(cv2.cvtColor(cropped_face, cv2.COLOR_BGR2RGB))
plt.title('Cropped Face 2')
plt.axis('off')
plt.show()

# Print prediction and confidence score
print("\nClass:", class_name[2:])
print("Confidence Score:", str(np.round(confidence_score * 100))[:-2], "%")

Output
Class: Lukas
Confidence Score: 50 %
```

Class: Lukas
Confidence Score: 990 %
Terdeteksi : Lukas

Testing Wira

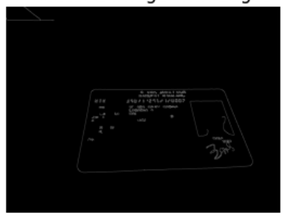
Original Image

Processed Image with Edges

Original Image with Contours

Original Image

Processed Image with Edges



Original Image with BoundingBox

Cropped Image

Original Image with Face

Cropped Face

Predict Wiradharma

```python
# Predict the model
prediction = model.predict(image)
index = np.argmax(prediction)
class_name = class_names[index]
confidence_score = prediction[0][index]

# Display the cropped face using plt
plt.imshow(cv2.cvtColor(cropped_face, cv2.COLOR_BGR2RGB))
plt.title('Cropped Face 2')
plt.axis('off')
plt.show()

# Print prediction and confidence score
print("\nClass:", class_name[2:])
print("Confidence Score:", str(np.round(confidence_score * 100))[:-2], "%")

Output
Class: Alfan Olivan
Confidence Score: 99 %
```

Class: Wiradharma
Confidence Score : 99 %
Terdeteksi : Wira

Testing Driya

Original Image

Processed Image with Edges

Original Image with Contours

Original Image

Processed Image with Edges

Original Image with BoundingBox

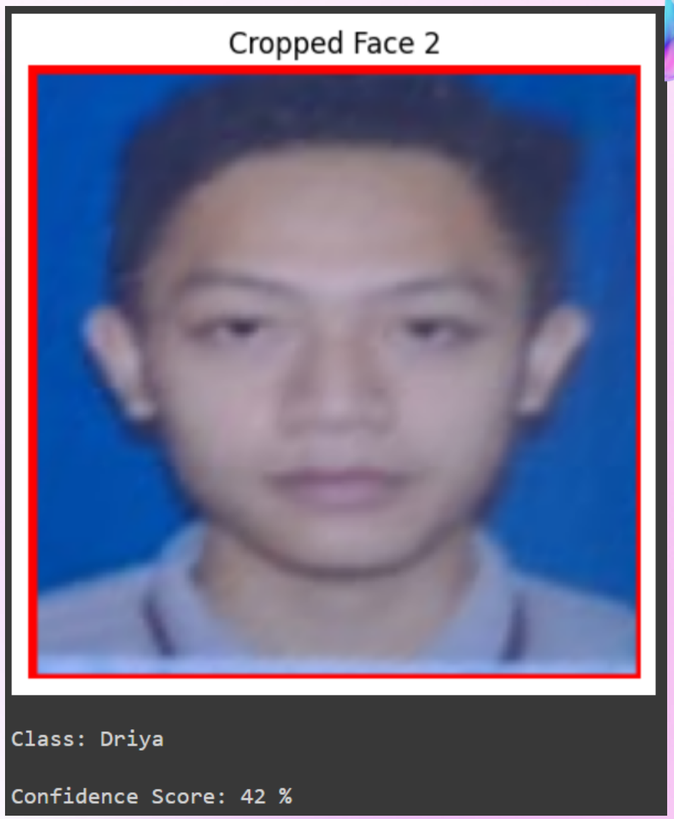
Cropped Image

Original Image with Face

Cropped Face

Predict Driya

```
# Predict the model
prediction = model.predict(image)
index = np.argmax(prediction)
class_name = class_names[index]
confidence_score = prediction[0][index]

# Display the cropped face using plt
plt.imshow(cv2.cvtColor(cropped_face,
cv2.COLOR_BGR2RGB))
plt.title('Cropped Face 2')
plt.axis('off')
plt.show()

# Print prediction and confidence score
print("\nClass:", class_name[2:])
print("Confidence Score:",
str(np.round(confidence_score * 100))[:-2], "%")
```



Class: Driya
Confidence Score: 42%

THANK YOU

Have a great day!