```
import
cv2
         from keras.models import load_model
         import numpy as np
         import os


         model = load_model('emojinator.h5')


         def main():
             emojis = get_emojis()
             cap = cv2.VideoCapture(0)
             x, y, w, h = 300, 50, 350, 350


             while (cap.isOpened()):
                 ret, img = cap.read()
                 img = cv2.flip(img, 1)
                 hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
                 mask2 = cv2.inRange(hsv, np.array([2, 50, 60]), np.array([25, 150,
         255]))
                 res = cv2.bitwise_and(img, img, mask=mask2)
                 gray = cv2.cvtColor(res, cv2.COLOR_BGR2GRAY)
                 median = cv2.GaussianBlur(gray, (5, 5), 0)


                 kernel_square = np.ones((5, 5), np.uint8)
                 dilation = cv2.dilate(median, kernel_square, iterations=2)
                 opening = cv2.morphologyEx(dilation, cv2.MORPH_CLOSE, kernel_square)
                 ret, thresh = cv2.threshold(opening, 30, 255, cv2.THRESH_BINARY)


                 thresh = thresh[y:y + h, x:x + w]
                 contours,kk = cv2.findContours(thresh.copy(), cv2.RETR_TREE,
         cv2.CHAIN_APPROX_NONE)[-2:]
                 if len(contours) > 0:
                     contour = max(contours, key=cv2.contourArea)
                     if cv2.contourArea(contour) > 2500:
                         x, y, w1, h1 = cv2.boundingRect(contour)
                         newImage = thresh[y:y + h1, x:x + w1]
                         newImage = cv2.resize(newImage, (50, 50))
                         pred_probab, pred_class = keras_predict(model, newImage)
                         print(pred_class, pred_probab)
                         img = overlay(img, emojis[pred_class], 400, 250, 90, 90)
```

```
        x, y, w, h = 300, 50, 350, 350
        cv2.imshow("Frame", img)
        cv2.imshow("Contours", thresh)
        k = cv2.waitKey(10)
        if k == 27:
            break


def keras_predict(model, image):
    processed = keras_process_image(image)
    pred_probab = model.predict(processed)[0]
    pred_class = list(pred_probab).index(max(pred_probab))
    return max(pred_probab), pred_class




def keras_process_image(img):
    image_x = 50
    image_y = 50
    img = cv2.resize(img, (image_x, image_y))
    img = np.array(img, dtype=np.float32)
    img = np.reshape(img, (-1, image_x, image_y, 1))
    return img


def get_emojis():
    emojis_folder = 'hand_emo/'
    emojis = []
    for emoji in range(len(os.listdir(emojis_folder))):
        print(emoji)
        emojis.append(cv2.imread(emojis_folder+str(emoji)+'.png', -1))
    return emojis


def overlay(image, emoji, x,y,w,h):
    emoji = cv2.resize(emoji, (w, h))
    try:
        image[y:y+h, x:x+w] = blend_transparent(image[y:y+h, x:x+w], emoji)
    except:
        pass
    return image


def blend_transparent(face_img, overlay_t_img):
    # Split out the transparency mask from the colour info
    overlay_img = overlay_t_img[:,:,:3] # Grab the BRG planes
    overlay_mask = overlay_t_img[:,:,3:]  # And the alpha plane


    # Again calculate the inverse mask
    background_mask = 255 - overlay_mask


    # Turn the masks into three channel, so we can use them as weights
    overlay_mask = cv2.cvtColor(overlay_mask, cv2.COLOR_GRAY2BGR)
    background_mask = cv2.cvtColor(background_mask, cv2.COLOR_GRAY2BGR)


    # Create a masked out face image, and masked out overlay
    # We convert the images to floating point in range 0.0 - 1.0
    face_part = (face_img * (1 / 255.0)) * (background_mask * (1 / 255.0))
    overlay_part = (overlay_img * (1 / 255.0)) * (overlay_mask * (1 / 255.0))


    # And finally just add them together, and rescale it back to an 8bit
integer image
    return np.uint8(cv2.addWeighted(face_part, 255.0, overlay_part, 255.0,
0.0))


keras_predict(model, np.zeros((50, 50, 1), dtype=np.uint8))
main()
```