RMIT UNIVERSITY

VirtEngine by DET.io

VirtEngine Identification System Technical Report

P000126SE - Cloud Marketplace Blockchain

Written by Rohan Poorun

Abstract

VirtEngine is a decentralised platform harnessing blockchain technology to offer advanced digital solutions. At its heart lies the VirtEngine Identification System (VEID), a groundbreaking identity solution enabling users to manage and control their digital identity securely. Built to be integrated with the existing Waldur platform, the VEID system not only ensures the transparent and safe storage of identity details but also empowers its seamless integration into the VirtEngine Decentralised Cloud Platform. Here, users employ the VEID for registration, login, and the authorization of transactions. Specifically, when purchasing computing resources, the system utilises the stored identity credentials to verify and authorise transactions, eliminating traditional password vulnerabilities and ensuring enhanced transaction security. Beyond mere identification, VirtEngine's infrastructure, inclusive of the Identity Wallet, facial recognition, biometric data processing, and Single Sign-On (SSO) integration, provides users with heightened data security, augmented control over personal information, and streamlined access to online services.

Table of Contents:

1. Introduction

1.1 Overview

This report delves into the intricate details of the VirtEngine Identification System (VEID), offering both a comprehensive understanding of its conceptual framework and a strategic technical blueprint for its implementation. By exploring the potential technologies that can be incorporated, the report seeks to illuminate the pathways for optimising the system's efficiency and security. As we navigate through the complexities of decentralised cloud computing, this document serves as a foundational reference for future developers and designers to grasp the underpinnings of VEID and its potential technical embodiments.

1.2 Background

VirtEngine emerges as a pivotal advancement in the cloud computing domain, introducing the novel concept of decentralised platforms. When juxtaposed against today's prevailing centralised cloud offerings, decentralised cloud computing extends numerous advantages, including superior security, enhanced fault tolerance, and a fair distribution of resources. The centralised paradigms, while dominant, often come with vulnerabilities tied to single points of failure and potential data centralization. In stark contrast, decentralised systems distribute both data and operations, mitigating these risks and fostering a democratised approach to resource accessibility.
The VirtEngine Identification System (VEID) is central to the platform's functionality. It facilitates crucial tasks, from user registration and login to the authorization of decentralised transactions. But VEID's importance doesn't stop at mere authentication. It acts as a gateway for users to access VirtEngine's expansive marketplace—a space not limited to VirtEngine's own offerings. This marketplace is a dynamic ecosystem where various providers can list and offer their cloud computing resources and services, making it a hub of decentralised cloud computing possibilities.

1.3 VirtEngine Identification System Summary

The VEID System works by utilising various machine learning algorithm techniques to identify, verify and authorise users via various scopes. These scopes include:

- Identity Documents
- Biometric Data
- Facial Data

- Authorised Online Accounts (Single Sign On)
- Domain Verification
- Email Verification

Identity Wallet: This is a digital wallet that allows users to store and manage their digital identity information, such as personal details, documents, and credentials. The wallet is secured by a private key, which only the user has access to.

Identity Verification: This is a process that verifies the authenticity and accuracy of the identity information provided by the user. This can be done through various methods, such as facial recognition, document verification, and biometric authentication.

Identity Services: These are online services and resources that use the VirtEngine ID system to verify and authenticate the identity of users. These services can include online banks, e-commerce platforms, and government agencies.

Identity Network: This is the decentralised network of nodes that stores and processes the identity information on the blockchain. The network is secured by advanced cryptographic techniques and consensus algorithms, which ensure the integrity and security of the data.

Identity Mobile Application:
This is a mobile application within a hardware device that captures and verifies the identity of the user through facial recognition and other biometric authentication methods. The mobile app can be integrated with the Identity Wallet to provide an additional layer of security and method to upload and provide identity documentation and data to the system.

1.4 Registration vs Authentication vs Authorization

In the evolving world of decentralised cloud computing, ensuring a secure user experience is paramount. The VirtEngine platform, with its integration of the Waldur front end, aims to achieve this security through a multi-tiered approach to user identification: Registration, Authentication, and Authorization.

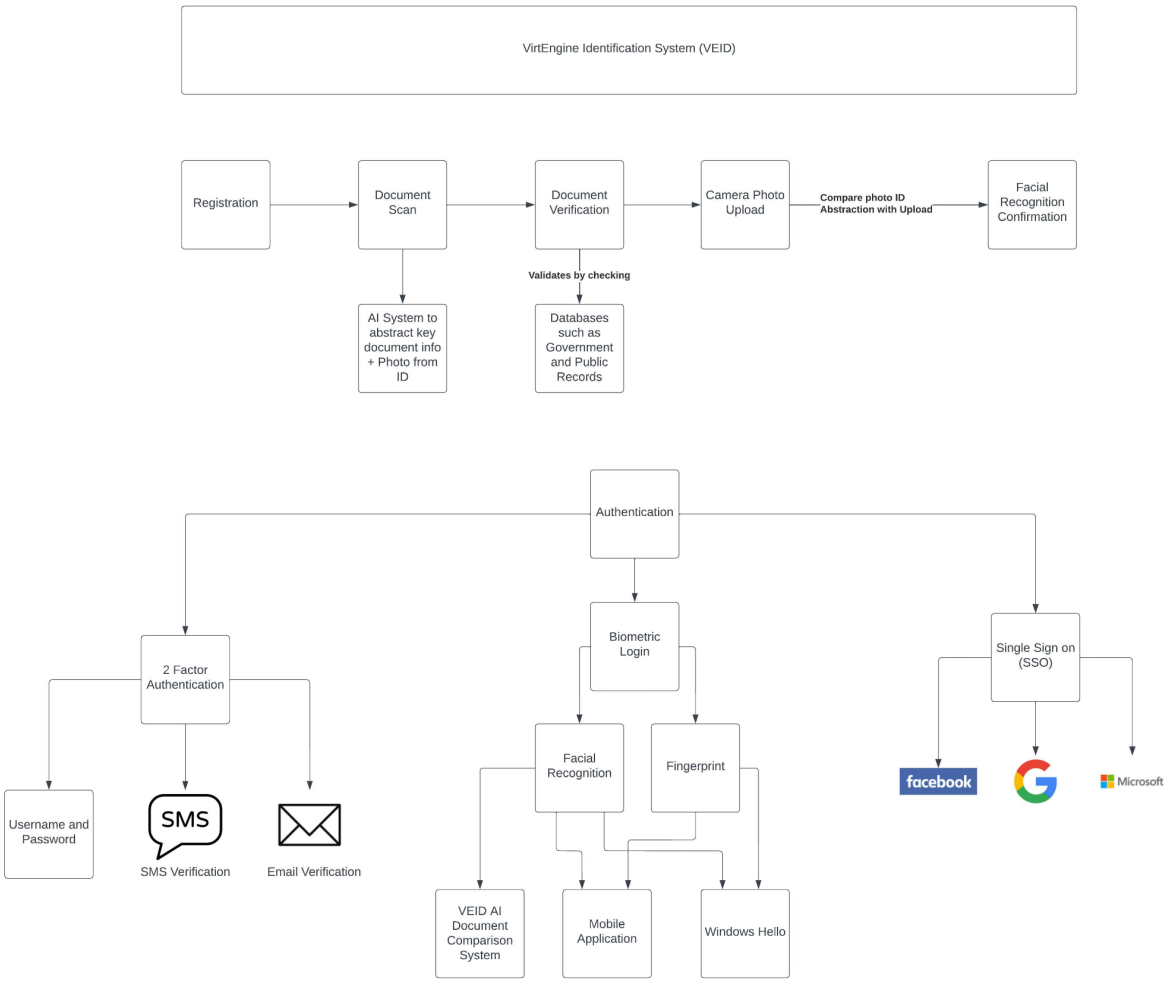
Registration:
Upon joining the VirtEngine platform, users have multiple avenues to register. The standard method involves a two-factor registration process where users input their email, password, and subsequently verify through either mobile or email. Alternatively, users have the option to employ Single Sign-On (SSO) using credentials from major platforms such as Google, Microsoft, or Facebook. It's worth noting that this initial registration grants users a basic, unverified account. Such accounts might face limitations, especially when interacting with vendors on the marketplace. This is because vendors, in their pursuit to ensure transparency and avoid potential illicit activities, often prefer to deal with verified users.

Authentication:
To bridge this gap between basic registration and a fully verified account, the VirtEngine Identification System (VEID) introduces an advanced Authentication layer. Through VEID, users can undergo a robust verification process. Here, they are required to upload

identification documents, which the system rigorously examines using machine learning algorithms. Not only does VEID verify the authenticity of the document, but it also extracts pertinent data, contrasting it against established government database records for validation. An integral part of this process involves image comparison. For instance, if a user uploads a scanned driver's licence, the system will extract the photo from the document and then request real-time images of the user. Using facial recognition technology, VEID compares the scanned photo with the real-time images to ascertain the user's identity. Upon satisfactory validation, users are assigned a score out of 100 and subsequently achieve a verified account status. This verification elevates their access privileges within the VirtEngine platform, ensuring a more enriching user experience.

Authorization:
While Registration and Authentication lay the groundwork for identity establishment, Authorization ensures the ongoing security of sensitive actions on the platform. For high-stakes operations, such as purchases that surpass a user-defined threshold, biometric verifications like fingerprint scans or facial recognition are necessitated. Without these added layers of security, users are restricted to merely browsing the marketplace. This stringent approach to Authorization is designed to fortify the safety of both providers and users, emphasising that in VirtEngine's decentralised landscape

1.6 Overall Flow of VirtEngine Identification System



2. Facial Recognition - VirtEngine Identification System

Facial recognition is a key element in the VirtEngine Identification System. This section outlines a proposed technical design for implementing a comprehensive architecture that provides seamless and accurate facial recognition. This section will delve into the system's components, detailing each state's role and operation, starting from image acquisition to the final decision-making engine as well as the technologies to implement this feature.

2.1 Image Acquisition Module

1. Device Cameras
Facilitate real-time image capturing to ensure up-to-date facial data.
Use OpenCV's VideoCapture() method to initialise and access the user's device camera.
Handle different camera resolutions by dynamically adapting to the camera's default settings.

Implement exception handling to manage cases where the camera isn’t accessible or malfunctions. Offer user feedback mechanisms, suggesting remediation steps.
Extract individual frames from the live feed using the read() function. This ensures granularity in image selection, allowing users to pick the most suitable frame for facial recognition.

2. Storage Retrieval
Enable retrieval of previously stored facial images for verification or recognition.
Centralised Cloud Solution:
Utilise Waldurs Backend Database
Programmatically retrieve stored images via API. Implement secure authentication protocols (like OAuth 2.0) to ensure data protection and ensure end to end encryption.
Implement caching solutions (e.g., Redis) to temporarily hold frequently accessed images, decreasing retrieval latency.
Decentralised Cosmos SDK Blockchain Solutions:
Utilise IPFS (InterPlanetary File System) integrated with the Cosmos SDK. IPFS provides decentralised storage solutions, ensuring data isn't centrally controlled, promoting data sovereignty.
Once facial data is stored on the blockchain, its integrity is maintained as any change results in a new block creation.
Implement Cosmos SDK-based smart contracts to retrieve facial image data. Ensure cryptographic mechanisms validate all retrieval transactions, affirming the legitimacy of requestors.
Given blockchain's size constraints, images might be fragmented into smaller chunks and distributed across nodes. A retrieval protocol should, therefore, reconstruct the original image by consolidating these chunks.

3. Buffering
Offer transient storage, acting as a staging area for images before they undergo subsequent processing stages.
Methodology:
Use Python's io.BytesIO() to create an in-memory binary stream object. This acts as a temporary receptacle for images.
Write image data to the buffer using the write() method and access it using the read() or getvalue() methods.
Periodically clear or reset the buffer using the truncate() and seek() methods to manage memory usage efficiently.

2.2 Image Preprocessing

Given we are using a facial recognition library such as DeepFace only specific preprocessing measures are required as detailed in the preprocessing plan below:

1. Image Acquisition:
Resolution Consistency: Consistency in resolution is pivotal to ensure compatibility with the comparison module. Utilise Python's OpenCV library to read images and determine an optimal resolution, for instance, 256x256 pixels. Subsequent to loading, employ the resize

function in OpenCV using interpolation techniques like INTER_LINEAR or INTER_CUBIC to conform all images to this uniform specification.

Format Consistency: Images should be standardised to a common format, such as JPEG or PNG, to reduce format-induced discrepancies. This uniformity can be achieved using Python's Pillow library, which offers the Image class to handle and convert image formats.

2. Grayscale Conversion:
While platforms like DeepFace can process RGB images, instances may arise where colour information varies across comparison samples. In these scenarios, converting the RGB image to grayscale becomes essential. This not only simplifies subsequent operations but also mitigates potential colour disparities, retaining vital facial structures. The transformation can be accomplished using OpenCV's cvtColor function combined with the COLOR_BGR2GRAY argument.

3. Histogram Equalization:
Images with diverse lighting conditions can benefit from Adaptive Histogram Equalization (AHE), a technique that accentuates image contrast, making facial attributes more pronounced. This step is paramount for clear comparisons. Achieve this using OpenCV's createCLAHE function, tailoring the clipLimit and tileGridSize parameters for dataset-specific optimization.

4. Noise Reduction:
External factors often introduce noise to images. To accentuate genuine facial features without noise interference, apply filtering techniques such as:
Gaussian Blur: Use OpenCV's GaussianBlur function.
Median Filtering: Employ medianBlur for a more aggressive noise reduction.
Bilateral Filtering: Opt for bilateralFilter which accounts for both spatial and colour differences, safeguarding edges during noise elimination.

5. Face Detection:
To emphasise facial characteristics, it's imperative to discern and extract the face from the image. This task can be executed using Deep Learning-based detectors or traditional Cascade Classifiers available in OpenCV or Dlib. These tools help in pinpointing the face, setting the stage for accurate comparisons.

6. Face Alignment: Alignment of crucial facial landmarks like eyes and mouth assures standardisation among images. Harness techniques that exploit facial landmarks, such as Dlib's 68-point landmarks model, to realise a consistent facial orientation, paving the way for precise comparison.

7. Face Cropping: Following the steps of detection and alignment, it's essential to segregate the face from any potential background distractions. By cropping to the face, the comparison operation's attention remains strictly on facial features. Achieve this cropping by employing Python's slicing techniques on the detected bounding box or landmarks.

8. Normalisation: Before channelling images into comparison tools like DeepFace, standardise pixel intensities to a consistent range, ideally [0,1]. This normalisation ensures

that images are on a unified intensity scale, primed for effective feature extraction and comparison. Divide the grayscale image values by 255 for normalisation.

By diligently executing this preprocessing sequence, facial images are aptly prepared for comparison. This thorough preparation minimises potential variances, enhancing the efficacy of recognition tools like DeepFace in delivering accurate results.

2.3 DeepFace

2.3.1 Introduction to DeepFace:

DeepFace is a Python framework for deep learning-based facial recognition. One of its core strengths is the encapsulation of several state-of-the-art models such as VGG-Face, Google FaceNet, OpenFace, Facebook DeepFace, and DeepID. Built on the Keras framework with TensorFlow and Theano backends, DeepFace abstracts many complex tasks and provides an out-of-the-box facial recognition solution.

2.3.2 Benefits of DeepFace:

No Need for Training: DeepFace comes pretrained on an extensive dataset, eliminating the need for additional training. This is especially beneficial for systems looking to implement facial recognition without investing time and resources in training a model from scratch.

Automated Preprocessing: The framework handles essential preprocessing tasks internally. This includes face detection, alignment, and normalisation, reducing the manual effort significantly.

Diverse Model Support: DeepFace supports multiple architectures, allowing users to choose the one most suited for their application.

Simplified API: The API abstracts underlying complexities, offering straightforward methods for tasks like representation and verification.

2.3.3 How DeepFace fits into the System Architecture:

Feature Extraction Module:
DeepFace Models: Leveraging one of the toolkit's architectures, such as FaceNet, can directly extract facial features without any manual intervention.

Automated Feature Extraction: By utilising DeepFace's representation method, it's possible to obtain a feature vector for an image seamlessly.

Training Module:
Eliminating Training Overhead: As DeepFace models come pretrained, there's no need for further training. This reduces computational costs and speeds up system deployment.

Data Augmentation: With the pretrained nature of the models, data augmentation steps become redundant, simplifying the pipeline further.

Batch Processing: Without the need for training, batch processing for optimised training is also unnecessary.

Recognition Module:
DeepFace Verification: The framework's verification function offers a direct way to compare two images and ascertain their similarity, bypassing manual handling of feature vectors or distance metrics.

Automated Distance Metrics: DeepFace employs model-specific distance metrics to provide similarity scores, negating the need for custom implementations.

2.3.4 Technical Implementation with DeepFace for VEID:

Installation: DeepFace can be easily installed via pip.

Face Representation: Using DeepFace's find function, extract the feature vector of an image. This step encompasses face detection, alignment, and feature extraction without requiring any manual setup.

Face Verification: To compare two images, the verify function returns a similarity score, determining if they originate from the same individual.

Database Integration: The feature vectors derived from the find method can be stored in the VEID database. For real-time recognition, compare the live image's vector with the stored vectors using DeepFace's built-in methods.

By adopting DeepFace, the VEID Facial Recognition System can bypass many traditional steps like preprocessing, training, and manual feature extraction. The framework's automation and abstraction simplify the recognition pipeline while ensuring accuracy and robustness. It is also based on tensorflow and open-source making it an optimal choice for VEID's requirements.

2.4 Decision Making Engine

Please note that this section is purely theoretical due to lack of development and testing of this system. Creating a decision making engine is complex and requires lots of testing. However the following can be used as a guide and best practices.

A Decision-making Engine is the logical computational system designed to automate the process of making decisions based on predefined criteria, algorithms, or patterns discerned from data, in this case the metrics of the similarity between the two faces compared.

Threshold Setting:

In any facial recognition system, the key determinant that propels a match between a new input and the reference dataset is a measure of confidence. This confidence is essentially a quantifiable metric that articulates the likelihood that the input image matches a particular face in the dataset.
To ensure the robustness and reliability of the system, it is paramount to establish a predetermined confidence threshold. For instance, a threshold might be set at 90%. This implies that for the system to acknowledge and validate a face match, the generated confidence score must equal or exceed this 90% mark.
From a technical standpoint, after the recognition algorithm (often a deep learning model in contemporary systems) processes an input image, it produces a confidence score. This score, typically scaled between 0 to 100%, represents the probability of a match. By comparing this score against the predefined threshold, the system instantaneously discerns whether the match is sufficiently reliable to be endorsed.
Setting this threshold is a balancing act: too low, and the system risks false positives; too high, and genuine matches might be disregarded. The optimal threshold is often identified through rigorous testing and validation on diverse datasets.

Fallback Mechanisms:
Given the variability of real-world conditions and the intrinsic complexities associated with facial structures, there will invariably be instances where the system's confidence wavers, and the resultant score hovers near the threshold but doesn't surpass it. In such borderline cases, relying solely on the algorithm's output could jeopardise the system's integrity.
To counteract these uncertainties, it's prudent to institute fallback mechanisms. These are secondary verification procedures triggered when the system's confidence doesn't meet the stipulated threshold.

For example, if a user's facial recognition score is 88%—close yet below the 90% threshold—the system could prompt the user to undergo an additional layer of verification. This could take the form of:
Secondary Biometric Checks: Requesting a fingerprint
One-time Password (OTP): Sending an OTP to the user's registered mobile number or email, which they then need to input for verification.
Both of these methods are supported by VirtEngines Identification System.

By implementing these fallback mechanisms, the system not only fortifies its defence against erroneous identifications but also ensures that genuine users, who might have been erroneously flagged due to minor discrepancies, still have a pathway to access, which is important as VirtEngine aims to have as little maintenance and manual support cases as possible, designed to be a self resilient system.

3. Identity Document Extraction and Verification Design - VirtEngine Identification System

3.1 Image Acquisition Module

Ensure the acquisition of a high-definition and clear digital representation of the user's identity document, paving the way for successful text detection and extraction.

3.1.1 Key Requirements

High-Resolution Image Capture: The resolution must be high enough to discern minute details of the text, which might include smaller fonts or embossed letters.

Consistent Image Orientation: The document must be positioned such that it is right-side up and horizontally aligned.

Minimal Distortion: The document should be photographed head-on, avoiding angles that could introduce distortions.
Implementation:

3.1.2 User Interface Design and User Guidance

Interface Design: Implement a user-friendly upload interface within the Waldur front end, allowing users to upload their identity document images easily.

User Guidance:
Direct users to position their document on a flat surface.
Recommend a well-lit environment but avoiding direct light sources that might introduce glare.
Advise users to hold the camera parallel to the document, ensuring an even distance from each corner, to prevent perspective distortions.
Allow users to preview the uploaded image and provide options to crop, rotate or retake the photo if necessary.

3.1.3 Image Quality Validation

Using Python's OpenCV library, upon image upload:
Resolution Check: Ensure the uploaded image meets a specified minimum resolution, e.g., 300 DPI (dots per inch). If the resolution is too low, request the user to re-upload or capture the image with higher settings.

Brightness & Contrast Check: Analyse the image for brightness and contrast values. If they fall outside optimal ranges (indicating potential underexposure or overexposure), prompt the user for a re-upload or offer automated adjustments.

Perspective Distortion Check: Using the Hough Line Transform in OpenCV, detect if the edges of the document are parallel with the image boundaries. If a significant skew is

detected, the user can be prompted to retake the photo or the system might offer to auto-correct the skew.

3.1.4 Metadata Stripping and Image Format Standardisation

Metadata Removal: Since images often come with metadata (like GPS location, camera model, etc.), use Python's Pillow library to strip away this data to ensure user privacy.

Format Standardisation: Convert all uploaded images to a standardised format (e.g., PNG) to ensure consistency across all processes. The Image class from Pillow allows easy conversion between image formats.

3.1.5 Feedback Loop & Error Handling

Feedback Mechanism: Once the image passes all quality checks, confirm successful upload to the user. If issues are detected, provide actionable feedback so the user understands how to rectify them.

Error Handling: In case of failed uploads due to network issues or unsupported file formats, offer error messages guiding the user towards successful image submission.

3.2 Image Preprocessing

Refine and standardise the captured identity document image, priming it for efficient and accurate text detection and subsequent extraction. A high degree of fidelity in the preprocessing phase is vital to achieve high accuracy in text extraction.

3.2.1 Key Requirements

Uniform Image Properties: Image properties such as brightness, contrast, and sharpness must be standardised. Fluctuations in these attributes can detrimentally influence the performance and accuracy of text detection models.

Optimal Image Resolution: While resolution significantly impacts clarity, it's a double-edged sword. Excessively high resolutions can lead to increased computational demand without a proportionate enhancement in extraction accuracy.

3.2.2 Brightness & Contrast Adjustment

Counteract uneven illumination and optimise the text's visibility and clarity. This is particularly vital when the original image has shadows or is under/overexposed due to suboptimal lighting during capture.
Technique:
Histogram Equalization: This process aims to distribute the intensity of the pixels of the image uniformly across the histogram. By applying OpenCV's equalizeHist() function to the image, one can improve contrast and brightness distribution.
Adaptive Histogram Equalization: This is a refined form of histogram equalisation. Using OpenCV's createCLAHE() function, this technique divides the image into smaller blocks and equalises them individually, thereby avoiding noise amplification.

3.2.3 Sharpness Enhancement

Emphasise the minutiae, predominantly focusing on text, particularly if the original image exhibits signs of blur, which is commonplace given handheld captures.
Technique:
Laplacian Sharpening: A method where a laplacian operator is used to compute the second derivative of the image, highlighting areas of rapid intensity change, which are primarily the edges in the image. Once computed, the resultant laplacian image can be subtracted from the original image, yielding a sharpened result.
Unsharp Masking: This method involves subtracting a blurred version of the image from the original, thereby amplifying the details. OpenCV's GaussianBlur() function can first create the blurred image, which is then subtracted from the original to generate the sharpened output.

3.2.4 Image Resizing

Attain a balance between computational efficiency and image clarity. As the subsequent text detection model may have an optimal input resolution, resizing becomes crucial.
Technique:
Bilinear and Bicubic Interpolation: While the resize() function in OpenCV performs resizing, the interpolation method selected deeply influences the quality. Bilinear (INTER_LINEAR) is computationally faster but might not preserve all details, whereas bicubic (INTER_CUBIC) offers a higher quality at the expense of computational time.

3.2.5 Noise Reduction

Cleanse the image of any spurious or random intensity variations, which can introduce inaccuracies in the text detection phase.
Technique:
Gaussian Blur: By convolving the image with a Gaussian function, this technique smoothens it, reducing high-frequency noise. However, it may also slightly blur the text.
Median Filtering: This nonlinear method replaces each pixel's value with the median value of its neighbours. It is especially adept at removing 'salt-and-pepper' noise without blurring the image.
Bilateral Filtering: A more sophisticated approach that preserves edges while reducing noise. It weighs both the spatial closeness and intensity similarity between pixels, ensuring that only those pixels with similar intensities to the central pixel are considered for blurring.

3.2.6 Perspective Correction:

As users may not always capture the identity document squarely, it might be skewed. Correcting such distortions aids in more accurate text detection.
Technique:
Four-point Transform: Identify the four corners of the document, either manually or via contour detection. Using these points, compute a perspective transform matrix. Apply this matrix to warp the image, ensuring the document appears as if captured squarely front-on.
This intricate and rigorous preprocessing pipeline establishes a strong foundation, significantly bolstering the accuracy and robustness of the ensuing text detection and extraction phases.

3.3 Text and Region of Interest Detection (ROI) using CRAFT

Identify and demarcate regions of interest (ROIs) containing text within the document using the CRAFT system. This will then allow these regions of interest to be passed into the OCR for text extraction.

3.3.1 Theoretical Overview of the CRAFT System

CRAFT, which stands for "Character-Region Awareness for Text detection," is a novel technique that focuses on detecting text from natural scenes, especially focusing on the detection at the character level. Unlike traditional methods that concentrate on word or line-level detection, CRAFT detects text by identifying individual characters and their regions. This level of granularity makes it adept at detecting text in documents with intricate layouts, varying fonts, or jumbled orientations.

3.3.2 How CRAFT Works:

Character-Level Text Detection: At its core, CRAFT is designed to detect individual characters in an image. By identifying each character's presence and its precise location, the model ensures high accuracy even in congested and complex textual patterns.

Affinity Score Prediction: Alongside character detection, CRAFT calculates 'affinity scores' between neighbouring characters. These scores indicate the likelihood that two characters belong to the same word or text line. This is a critical component that allows CRAFT to delineate distinct text regions and group characters together.

Use of Gaussian Heatmaps: For each detected character, CRAFT produces a Gaussian heatmap, emphasising the centre of the character. This approach provides a probabilistic sense of each character's presence and its precise region, allowing for clearer distinction even if characters are closely spaced.
Region Linking: With the character-level bounding boxes and affinity scores in place, the model links these individual detections to form cohesive word or line-level text regions.

Post-Processing: After detection, some post-processing steps, like non-maximum suppression and thresholding, refine the detected regions to ensure that only the most probable text regions are retained.

3.3.3 Implementation:

Obtain a pre-trained CRAFT model for Python. This model is typically trained on a vast dataset comprising various text types, orientations, and complexities, ensuring robustness.

Input the preprocessed image from the earlier stage into this model. As the model processes the image, it will generate character-level bounding boxes across the document.

Utilise the affinity scores produced by the model to group these character-level detections into word or line-level regions. This step ensures that text is recognized cohesively, considering the spatial relationships between characters.

Finally, store these delineated regions of interest (containing text) for the subsequent OCR phase. Each region can be considered a separate input for the OCR, ensuring that text extraction is performed with high precision.

3.3.4 Example CRAFT python library:

The CRAFT (Character Region Awareness for Text Detection) method is a state-of-the-art technique for text detection in images. It focuses on detecting individual characters and then grouping them to form words or text regions. The craft-text-detector library is an implementation of the CRAFT method, which can be used to identify regions of interest in an image before applying Optical Character Recognition (OCR) for text extraction. This has been tested and has worked successfully.

Code Flow and Implementation
1. Initialization
The library starts with the __init__.py file, which primarily imports necessary modules and functions for the library to function.
2. Utility Functions
craft_utils.py: This module provides utility functions specific to the CRAFT method. It includes functions for:

- Adjusting link and character scores.
- Getting prediction results from the model.
- Post-processing the predictions to get bounding boxes.

file_utils.py: Contains utility functions for file operations, such as:

- Loading and saving images.
- Reading text files.

image_utils.py: Provides utility functions for image processing, including:

- Resizing images while maintaining the aspect ratio.
- Loading images with the correct orientation.

3. Model Architecture
vgg16_bn.py (BaseNet): This module defines the VGG16 architecture with batch normalisation. VGG16 is used as the backbone or base network for the CRAFT method.
craftnet.py: Defines the CRAFT model architecture. It integrates the VGG16 base network and adds additional convolutional layers to detect character regions and affinity scores between characters.
refinenet.py: Implements the RefineNet, which refines the text detection results from the CRAFT model. It enhances the accuracy of the detected text regions.

4. Prediction
predict.py: This is the core module for text detection. It includes functions for:

- Loading the trained CRAFT model.
- Pre-processing the input image.
- Getting predictions from the model.
- Post-processing the predictions to obtain text regions.

5. Torch Utilities
torch_utils.py: Provides utility functions related to PyTorch, such as:

Checking and loading CUDA devices.
Loading trained model weights.

Implementation of the CRAFT Extraction Method

Image Pre-processing: The input image is first pre-processed using functions from image_utils.py. This includes resizing the image and ensuring it has the correct orientation.
Model Loading: The trained CRAFT model weights are loaded using functions from torch_utils.py.
Text Detection: The pre-processed image is passed through the CRAFT model (craftnet.py) to get character region scores and affinity scores.
Refinement: The initial text detection results are refined using the RefineNet (refinenet.py).
Post-processing: The raw predictions from the model are post-processed using functions from craft_utils.py to obtain bounding boxes around text regions.
Output: The final output consists of bounding boxes that highlight the text regions in the input image. These regions can then be passed to an OCR system for text extraction.

3.4 Text Extraction with Tesseract OCR:

The Tesseract OCR engine, developed by Google, has undergone significant improvements over the years and has positioned itself as a reliable open-source OCR solution. With its ability to recognize multiple languages and its deep learning capabilities, Tesseract has consistently shown impressive accuracy levels in text extraction tasks. Pytesseract is a Python wrapper for the Tesseract OCR engine, offering a simple means to extract text directly within Python applications.

Implementation Details:
ROI Cropping:
After text detection with CRAFT, each identified Region of Interest (ROI) must be carefully cropped to minimise noise and interference during the OCR phase.
Utilising OpenCV's cv2.getRectSubPix() function will ensure precise and efficient cropping, taking the bounding box coordinates of each ROI.
This results in a collection of cropped image segments, each containing individual text elements.

Image Preprocessing for OCR:
While Tesseract is robust, it's not infallible. Its performance can significantly benefit from preprocessing.
Convert each cropped image segment to grayscale using OpenCV’s cv2.cvtColor() function. A grayscale conversion is fundamental as it simplifies the image and reduces computational overhead.
Applying a binary threshold can further simplify the image, making text stand out more distinctly against the background. Adaptive thresholding methods can be considered for images with varying lighting conditions.

Noise removal might be beneficial in certain instances. Techniques like Gaussian blurring or median filtering can smooth out the image, but care must be taken not to over-smooth, as that might affect text legibility.

Tesseract OCR Integration:
Pytesseract is an invaluable bridge between Python applications and the Tesseract OCR engine. Once the Python environment is set up with Tesseract and Pytesseract installed, text extraction becomes a straightforward task.
For each preprocessed ROI, pass it to the Tesseract engine using the pytesseract.image_to_string() function.
To improve accuracy, specific configuration parameters (like OCR language, whitelist/blacklist of characters, etc.) can be supplied to Tesseract via Pytesseract. This customization is especially useful when dealing with identity documents as they might have specific predictable formats or limited sets of characters that need to be recognized.

Post-processing:
After text extraction, there might be occasional OCR misinterpretations. While Tesseract's accuracy is high, certain edge cases or challenging image conditions can lead to minor errors.
Regular expressions or string manipulation techniques can be applied to correct predictable and recurrent mistakes, especially when the format of the text (like date formats, identification numbers) is known.
Additionally, metadata associated with each text extraction (like the position of the text in the original document) can be stored alongside the extracted text. This can be useful in reassembling the data in a structured format that mirrors the original document's layout.

3.5 Face Extraction from Identification Document

3.5.1 UNET

U-Net is a convolutional neural network (CNN) architecture specifically designed for biomedical image segmentation. It was introduced by Olaf Ronneberger, Philipp Fischer, and Thomas Brox in their 2015 paper titled "U-Net: Convolutional Networks for Biomedical Image Segmentation." The architecture is named "U-Net" because of its U-shaped structure when visualised.

Architecture
The U-Net architecture can be divided into two main parts: the contracting (downsampling) path and the expansive (upsampling) path, which together give it a symmetric U-shape.
2.1 Contracting Path
Layers: Consists of a series of convolutional layers followed by max-pooling layers.
Function: This path captures the context of the input image, reducing the spatial dimensions while increasing the depth (number of feature maps).
Details: Each step involves two 3x3 convolutions followed by a rectified linear unit (ReLU) activation, and then a 2x2 max pooling operation with stride 2 for downsampling.

2.2 Expansive Path
Layers: Comprises a series of up-convolutional layers followed by regular convolutional layers.
Function: This path allows for precise localization, restoring the spatial dimensions of the image.
Details: Each step involves an upsampling of the feature map followed by a 2x2 convolution (up-convolution), a concatenation with the corresponding feature map from the contracting path, and two 3x3 convolutions, each followed by a ReLU activation.

2.3 Skip Connections
Function: These are the connections between the contracting and expansive paths.
Details: At each level of the expansive path, feature maps from the contracting path are concatenated. This helps in retaining the high-resolution features, aiding in more accurate localization.

3. Importance in Image Segmentation
3.1 High Resolution
U-Net retains the boundary details of the segmented objects, which is crucial for tasks like medical image segmentation where precision is paramount.

3.2 Fewer Training Samples
U-Net is designed to work with fewer training samples and still produce accurate segmentations. This is beneficial, especially in biomedical applications where labeled data can be scarce.

3.3 Efficient Training
Due to its symmetric architecture and the use of skip connections, U-Net can be trained efficiently, converging in fewer epochs compared to other architectures.

4. Importance in the Identification Document Face Extraction Implementation

4.1 Precise Segmentation
Extracting faces from identification documents requires precise segmentation to ensure that only the face region is extracted without any surrounding noise. U-Net, with its ability to retain high-resolution features, is well-suited for this task.

4.2 Handling Variability
Identification documents can come in various formats, lighting conditions, and qualities. U-Net's architecture, which captures both global context and local details, can handle such variability effectively.

4.3 Robustness
The skip connections in U-Net ensure that even if certain features are lost during the downsampling process, they can be recovered in the upsampling process, making the network robust to variations in input data.

3.5.2 Implementation

Pre-processing
The pre-processing stage prepares the image for further analysis by enhancing its features and reducing noise.

Image Loading:

Function: To read the ID card image into the system.
Method: Utilise OpenCV's imread function.
Consistency: It's essential that all images processed have a uniform size. If they don't, use OpenCV's resize function with appropriate interpolation techniques (like INTER_LINEAR or INTER_CUBIC) to standardise the image dimensions.

Grayscale Conversion:

Function: Simplify the image by reducing its channels from three (RGB) to one.
Method: Use OpenCV's cvtColor function with the COLOR_BGR2GRAY flag.
Advantage: Reducing the image to grayscale minimises computational overhead in subsequent steps.

Noise Reduction:

Function: Eliminate minor disturbances or "noise" in the image.
Method: Apply a Gaussian blur using OpenCV's GaussianBlur function.
Advantage: A smoother image aids in better segmentation and feature detection.

Thresholding:

Function: Differentiate the regions of interest from the background.
Method: Implement adaptive thresholding using OpenCV's adaptiveThreshold function.
Advantage: This dynamic method adjusts the threshold value based on the local neighbourhood, ensuring better segmentation, especially in varying lighting conditions.

3.2 Face Detection
This stage identifies and isolates the face region in the ID card image.
Model Selection:

Choice: A pre-trained U-Net model is ideal due to its excellence in segmentation tasks.
Reason: U-Net's architecture, with its contracting and expansive paths, captures both contextual and detailed information, making it apt for precise segmentation tasks like face detection.

Model Prediction:

Normalisation: Before feeding the image to the model, normalise it to ensure all pixel values are in a consistent range, typically [0,1].
Prediction: Pass the normalised image through the U-Net model. The output will be a mask highlighting the face region.

Post-processing:

Binary Conversion: Transform the mask into a binary format where pixels with values above a certain threshold represent the face, and others represent the background.
Morphological Operations: Enhance the mask using dilation (to strengthen the face region) and erosion (to eliminate small noise). These operations can be performed using OpenCV's dilate and erode functions.

3.3 Face Extraction
This stage crops the detected face from the ID card.

Bounding Box Detection:
Function: Determine the rectangular region encompassing the face.
Method: Use the binary mask from the U-Net prediction to detect the bounding box coordinates.

Face Cropping:
Function: Isolate the face from the rest of the ID card.
Method: Use OpenCV's Rect class to define the bounding box and then extract the region of interest (ROI) from the original image.

3.4 Utility Functions
These are supplementary functions aiding in various stages of the process.

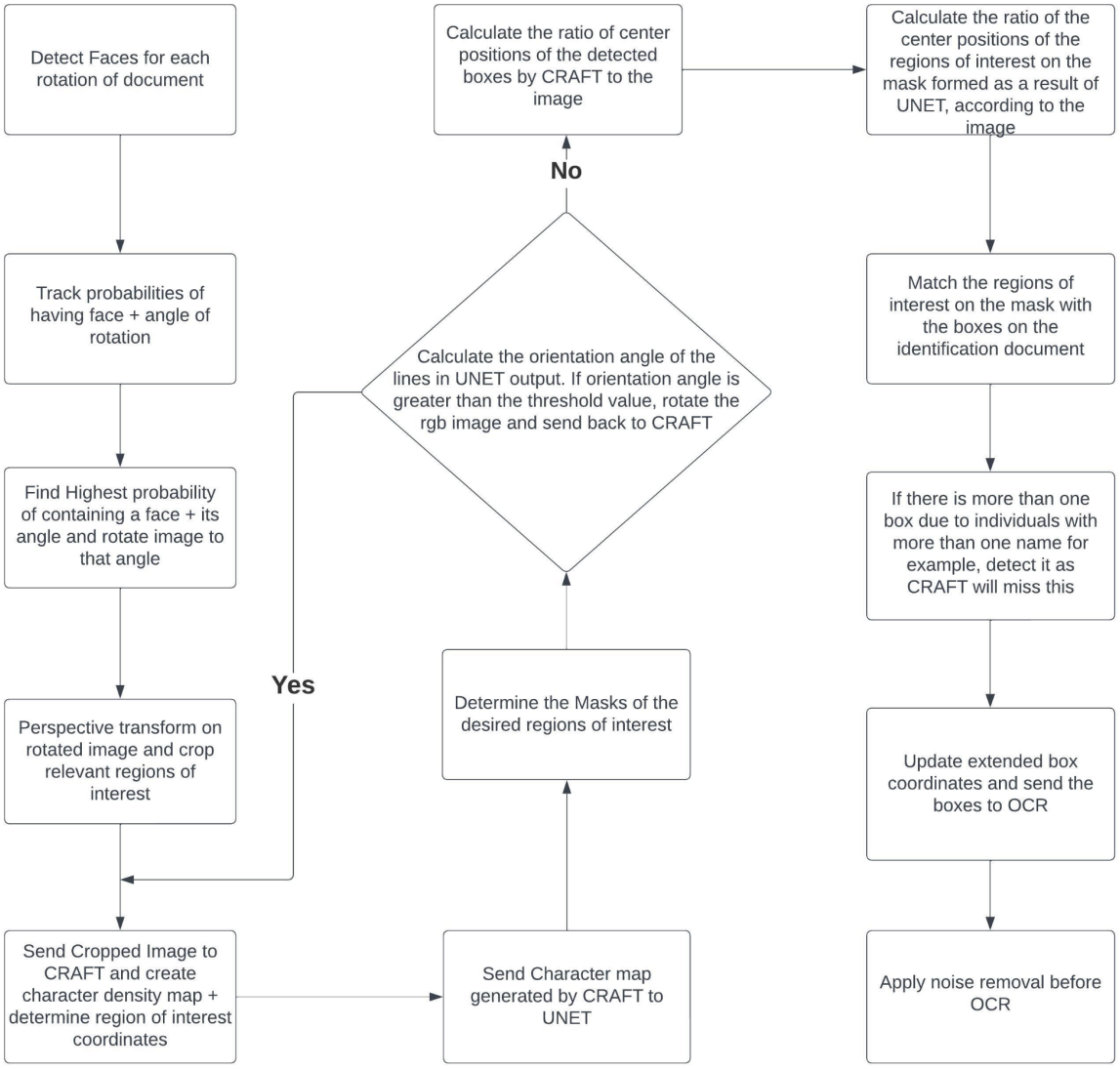
Distance Calculation:
Function: Compute the distance between two points.
Method: Implement the Euclidean distance formula.

Nearest Box Detection:
Function: Identify the closest bounding box to a given point or region.
Application: Useful in associating words with their respective fields based on proximity.

Image Display:
Function: Visualise images during the development phase.
Method: Use OpenCV's imshow function.

4. Tools and Libraries
OpenCV: A comprehensive library offering tools for various image processing tasks, from basic operations like loading and transformation to advanced functions like contour detection.
PyTorch: A deep learning framework that facilitates the training, evaluation, and deployment of models like U-Net.
U-Net Architecture: A specialised CNN architecture designed for image segmentation tasks, particularly effective in scenarios requiring precise boundary detection.

3.6 Example Algorithm Flow



Face Detection:
Utilising advanced facial recognition algorithms, the system first scans the input image at varying rotation angles. The objective here is twofold: to detect the presence of a face, indicative of the orientation of the ID, and to record the rotation angle at which this face was detected. This probabilistic approach ensures that the ID is oriented correctly regardless of its initial position.

Image Rotation:
Post face detection, the system identifies the rotation angle where the probability of a face being detected is maximised. Using this optimal angle, the original image undergoes a rotation transformation. This ensures that any subsequent processing occurs on an image that is oriented in a standardised fashion.

Perspective Transformation & Cropping:

With the image now appropriately oriented, it is paramount to adjust for any inherent skew or tilt. A perspective transformation is applied, ensuring the ID's text is perfectly horizontal, which is a prerequisite for optimal text detection. Once the image is aligned, the section specifically containing the identification details is isolated and cropped, reducing computational overhead in subsequent steps.

Text Detection with CRAFT:
The cropped section of the image is relayed to the CRAFT model. Renowned for its text detection capabilities, CRAFT produces a detailed character density map. This map elucidates the coordinates of various text boxes, demarcating regions of textual data.

UNET Processing for Box Detection:
The character density map is not the terminus. It serves as an input to the UNET model, a neural network known for its segmentation capabilities. UNET diligently processes this map to pinpoint masks corresponding to four pivotal data regions: Identity Number, First Name, Last Name, and Date of Birth.

Centre Position Ratio Calculation:
Akin to a calibration step, the system calculates the ratio of the centre positions of the boxes detected by CRAFT in relation to the overall image dimensions. This computational step is pivotal as it ensures a standardised and precise alignment of detected text boxes, a foundation for accurate text extraction.

Orientation Adjustment with UNET Output:
Harnessing the UNET output, the system discerns the orientation angle of the detected lines. If this angle exceeds a predefined threshold, it suggests an orientation anomaly. The system promptly rotates the RGB image to rectify this, ensuring textual data is horizontally aligned. This corrected image is then looped back to CRAFT for refined text detection.

Box Matching:
Precision is the linchpin here. The algorithm matches the four boxes identified on the segmentation mask with the analogous boxes on the ID card. This validation step ensures data fidelity, guaranteeing that each box truly represents its designated section.

Multiple Box Detection:
CRAFT's granularity can lead to the detection of multiple boxes, especially for individuals with extended names or surnames. The system is designed to accommodate this. It detects and segregates these additional boxes, ensuring that each name component is accurately represented and extracted.

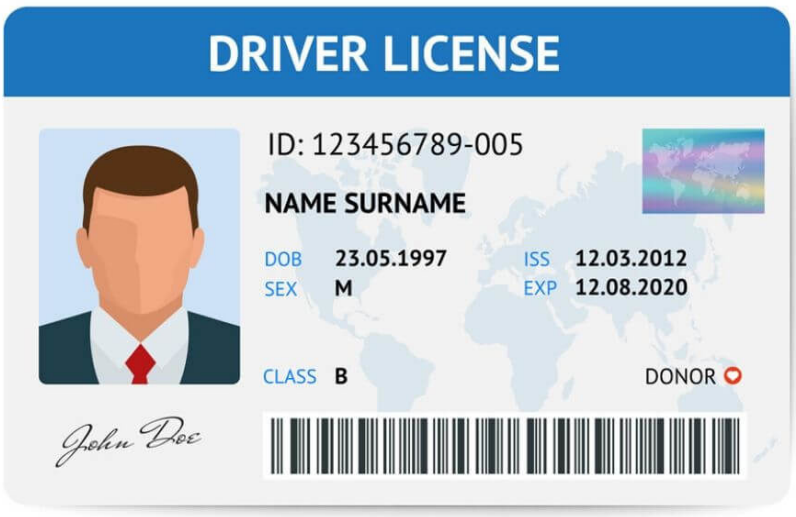
Box Coordinate Update:
Post detection, there's an imperative to ensure the entirety of the text is encapsulated within the detected boxes. To achieve this, the box coordinates are meticulously extended. These refined boxes, now encompassing all textual details, are ushered into the OCR stage.

Text Extraction with Tesseract OCR:
The pièce de résistance is the Optical Character Recognition (OCR) stage. But prior to text extraction, the system undergoes a noise removal phase. This phase, employing advanced

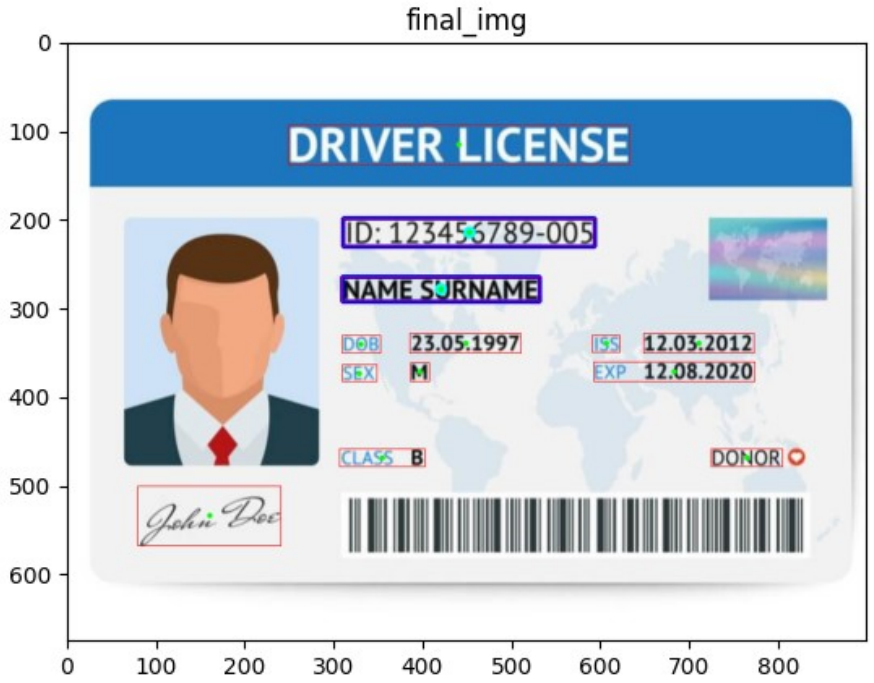
digital filters, ensures noise-free, clear textual regions. Post this, Tesseract OCR, a state-of-the-art text recognition engine, is summoned to extract pertinent details like name, surname, identification number, and date of birth with unparalleled accuracy.

3.7 Test Implementation Graph Results

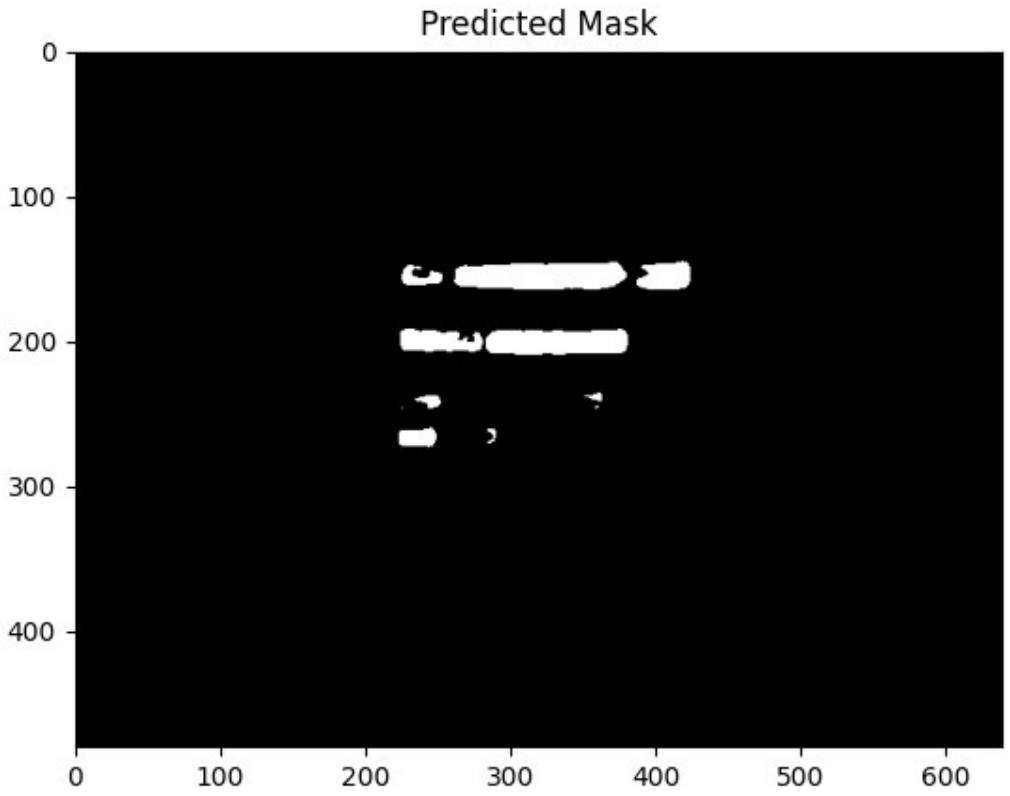
3.7.1 Original Identification Card Image



3.7.2 Regions of Interest



3.7.3 Predicted Mask using UNET



4. Security Considerations

4.1 Security Strategies

Biometric data, including facial features, and sensitive document data are paramount datasets requiring meticulous security. In the context of a system like VEID, ensuring their safety is essential, both from ethical, privacy, and legal standpoints. Below are some examples of strategies as well as implementation on how this risk can be mitigated. All of these are open-source. Since the database solution has not been finalised there are strategies for both decentralised blockchain data storage as well as centralised and decentralised database storage.

1. Advanced Data Encryption:
At Rest: For data at rest the open-source libsodium library can be used. It offers AES-256-GCM, ensuring the highest security for stored data.

In Transit: For data in transit, the OpenSSL toolkit provides a full-strength general-purpose cryptography library, which implements the Transport Layer Security (TLS) protocols, ensuring that data remains confidential and integral during transmission.

2. Data Tokenization and Pseudonymization:
Tokenization: By using tools like TokenD, actual biometric and document data can be replaced with non-descriptive tokens for any external operations, retaining the genuine data in a secured environment.

Pseudonymization: This process converts stored data into artificial, unrelated identifiers. It makes the data nonsensical outside the system's context, ensuring enhanced privacy without sacrificing the utility of the data.

3. Multi-factor Authentication (MFA):
For robust security beyond just passwords, integrate MFA solutions. VirtEngine works with a range of different MFA options, including email/SMS OTP, OAuth and biometric authentication.

4. Enhanced Database Security:
Database Auditing: Tools such as pgAudit for PostgreSQL enable detailed auditing of database activities, ensuring accountability and tracking of any unauthorised or suspicious activities.
Network Segregation: Implementing pfSense, an open-source firewall and router software, ensures that the biometric database is segregated and protected from potential vulnerabilities in other zones.

5. Blockchain Storage with Cosmos:
Immutable Logs: Cosmos inherently ensures data immutability. Any access or change is permanently recorded, providing a traceable and accountable record.
Smart Contracts for Access Control: Cosmos allows the creation of smart contracts, ensuring biometric and document data is accessed only under strict, predefined conditions.
Distributed Ledger Technology (DLT): With Cosmos' DLT, data is stored across multiple nodes, providing redundancy and ensuring high resilience against malicious attacks.

6. Data Minimization and Feature Extraction:
By employing open-source computer vision libraries like OpenCV or Dlib, it's feasible to process facial data, extract vital features, and then discard or mask the original image. This ensures that only minimal, necessary data is retained.

7. Periodic Security Evaluations:
Tools like OpenVAS or OWASP ZAP provide comprehensive vulnerability assessments and penetration testing capabilities. They help in identifying potential security loopholes and aid in maintaining the system's robustness against emerging threats.

8. Backup Mechanisms:
Open-source tools such as BorgBackup or Duplicati can be employed for encrypted backups. These tools allow for deduplication, compression, and encryption, ensuring optimal and secure backup storage.

9. Data Lifecycle Management:

With tools like MongoDB's TTL indexes, biometric data can be set to be automatically removed post the defined retention period, making sure that no data is retained beyond its intended lifecycle.
10. User-centric Transparency:
Open-source consent management platforms can be integrated to transparently handle user consents. Furthermore, educational platforms like Moodle can be employed to inform users about data usage policies and the security measures in place.

4.2 Future Risk Consideration - Quantum Computing

Quantum computing, with its potential to process complex calculations exponentially faster than today's classical computers, stands poised to disrupt many fields, including cryptography and data security. Given the sensitive nature of the VEID facial recognition feature and its reliance on cryptographic measures, identifying the potential threats posed by quantum computing is paramount.

1. Threat to Cryptographic Systems:
Breaking Encryption: Quantum computers can potentially run algorithms like Shor's algorithm, which can factorise large numbers more efficiently than classical computers. This could undermine encryption algorithms like RSA (Rivest–Shamir–Adleman) and ECC (Elliptic Curve Cryptography) which rely on the difficulty of factorization. The AES encryption, though robust in its classical sense, may require more extended key lengths to remain secure against quantum attacks.

Compromising Digital Signatures: The integrity of digital signatures, which authenticate and verify the legitimacy of data, could be jeopardised. Algorithms that verify digital signatures might be vulnerable to quantum algorithms, making unauthorised data alterations harder to detect.

2. Rapid Data Analysis:
Quantum computers' capability to handle and analyse vast datasets swiftly might make it feasible to reverse-engineer tokenization or masking techniques, leading to potential data exposure.

3. Threat to Blockchain:
Double Spending: Quantum computers might increase the risk of double-spending in blockchain, where the same cryptocurrency is spent more than once, undermining the chain's integrity.
Breaking Public Key Cryptography: Blockchain's security relies heavily on public key cryptography. Quantum computers could potentially derive private keys from their public counterparts, giving unauthorised access to data blocks.

4. Mitigation Strategies:
While quantum computers pose threats, there are steps that can be taken to mitigate these risks:

Post-Quantum Cryptography: Research is ongoing to develop cryptographic algorithms that are inherently resistant to quantum attacks. Transitioning to such algorithms as they mature will be essential.
Quantum Key Distribution (QKD): This is a method wherein cryptographic keys are generated and distributed using quantum mechanics principles, ensuring any eavesdropping attempts can be detected.

Hybrid Systems: Using a combination of classical and quantum-resistant algorithms might offer a balanced approach, providing security while quantum-resistant measures continue to develop.

References

1. Nielsen, M. A., & Chuang, I. L. (2010). Quantum computation and quantum information. Cambridge University Press.
2. Bernstein, D. J., & Lange, T. (2017). Post-quantum cryptography. Nature, 549(7671), 188-194.
3. Shor, P. W. (1999). Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM review, 41(2), 303-332.
4. Gottesman, D., & Lo, H. K. (2003). Proof of security of quantum key distribution with two-way classical communications. IEEE Transactions on Information Theory, 49(2), 457-475.
5. Zohar, A., & Lindell, Y. (2018). Bitcoin and cryptocurrency technologies. Princeton University Press.
6. Bradski, G., & Kaehler, A. (2008). Learning OpenCV: Computer vision with the OpenCV library. "O'Reilly Media, Inc.".
7. Gonzalez, R. C., & Woods, R. E. (2002). Digital image processing (2nd ed.). Prentice Hall.
8. Baek, Y., Lee, B., Han, D., Yun, S., & Lee, H. (2019). CRAFT: Character-Region Awareness for Text detection. Retrieved from https://arxiv.org/abs/1904.01941
9. Smith, R. (2007). An Overview of the Tesseract OCR Engine. In Proceedings of the Ninth International Conference on Document Analysis and Recognition - Volume 02 (ICDAR '07). IEEE Computer Society.
10. Lidar, D. A., & Brun, T. A. (2016). Quantum Error Correction. Cambridge University Press.
11. Bradski, G. (2000). The OpenCV Library. Dr. Dobb's Journal of Software Tools.
12. Baek, Y., Lee, B., Han, D., Yun, S., & Lee, H. (2019). Character Region Awareness for Text Detection (CRAFT). Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).
13. CRAFT: Character-Region Awareness For Text detection GitHub Repository. https://github.com/clovaai/CRAFT-pytorch
14. Tesseract OCR Official GitHub Repository. https://github.com/tesseract-ocr/tesseract
15. Pytesseract GitHub Repository. https://github.com/madmaze/pytesseract
16. OpenCV Documentation. https://docs.opencv.org/4.x/
17. Bernasek, L., Wiesner, M., Zendulka, J., & Polydoros, A. S. (2021). Blockchain and its role in the digital identity. Information Systems.

18. Preskill, J. (2018). Quantum Computing in the NISQ era and beyond. Quantum, 2, 79.
19. Tc_ID_Card_OCR Github Repository https://github.com/musimab/Tc_ID_Card_OCR
20. Craft-text-detector Github Repository https://github.com/fcakyon/craft-text-detector
21. Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. Medical Image Computing and Computer-Assisted Intervention (MICCAI).
22. Ronneberger, O., Fischer, P., & Brox, T. U-Net Official Website. Laboratory for Medical Image Computing, University of Freiburg.