אורט בראודה
ORT Braude College

Software Engineering Department

ORT Braude College

Capstone Project Phase A

Investigate Handwriting for Identification Using Deep Learning 10-D-2-2

In Partial Fulfillment of the Requirements for

Final Project in Software Engineering (Course 61998)

June 2021 - Karmiel

Doron Tuchman 201094505 Doron.tuchamn@e.braude.ac.il

Adi Lampert 315963280 Lampert.adi@e.braude.ac.il

Supervisor:

Mr. Illya Zeldner

Table of Contents

Abstract 1

1. Introduction 1

 1.1. Organization of the paper 2

2. Background and Related Work 2

 2.1. Programing languages 2

 2.2. Deep Learning and Machine Learning Libraries 3

  2.2.1. PyTorch 3

  2.2.2. TensorFlow 3

  2.2.3. Keras 3

  2.2.4. PyTorch VS TensorFlow VS Keras 3

 2.3. Algorithms 3

  2.3.1. Convolutional neural network (CNN) 3

  2.3.2. Autoencoder 4

  2.3.3. Scale-Invariant Feature Transform (SIFT) 4

  2.3.4. FLANN (Fast Library for Approximate Nearest Neighbors) 4

  2.3.5. Max pooling 4

  2.3.6. Harris Corner Detector 5

  2.3.7. Softmax 5

  2.3.8. Rectified linear activation function (ReLU) 5

 2.4. Requirements 5

3. Expected Achievements 5

4. Research Process 6

 4.1. Process 6

 4.2. Product 6

  4.2.1. Multi-stream Deep CNN Algorithm 6

  4.2.1.1. Multi-stream input and patch scanning 7

  4.2.1.2. Convolutional layers and Max-pooling 8

  4.2.1.3. Fully connected layer 8

  4.2.1.4. Sum 8

  4.2.1.5. Softmax classifier and Average 9

  4.2.1.6. Testing details 9

  4.2.1.7. Pseudo code 10

  4.2.2. Two channel AutoEncoder with SIFT 10

4.2.2.1. Two channel AE 10
4.2.2.2. Pseudo code 11
4.2.2.3. SIFT 11
4.2.2.4. Pseudo code 12
4.2.2.5. Summery 12
4.2.3. Activity Diagram 13
4.2.4. Use Case 14
4.2.5. Class Diagram 16
.4.2.6 User interface 17
5. Evaluation/Verification Plan 20
References 22

Abstract

Handwriting is a unique thing that produced differently for each person, therefore there are different characteristics and huge variation in individual writing styles so handwriting can be used as a variable in biometric systems. In our paper we propose a deep multi-stream CNN, combined with SIFT algorithm, to learn deep powerful representation for recognizing writers. First, we use an optimize multi-stream structure for writer identification. Second, we use SIFT for feature detection on the most common word to yield better results.

Keywords: Deep learning, DeepWrite, Multi-stream, CNN, Mini-batch gradient, SIFT, Softmax, Max pooling, AutoEncoder.

1. Introduction

Our project addresses the problem of automatic writer identification using scanned image of the author handwrite. Writer can be recognized by capturing specific characteristics of handwriting habits of one author, which differ from other authors.
[1] The main difference between two handwritten images is dominated by the text contents. For writer identification, one needs to extract abstractive written style features and fine details which reflect personal writing habits.
The purpose of our project and the automatic writer identification is recognizing a person based on his handwrite text.
There are many areas where handwritten text detection has key impact, such as insurances, healthcare and pharmaceuticals, educational institutions and banking.
The solution to the writer identification can solve many problems related to these areas: Signature verification and forgery, and determination whether the signature is real or not; Detect copies in tests and written papers by matching the writer handwriting to the submitter; Read and enter the information present on a cheque and also verify the entries like signature and date ;Comparison between a person's writing when he is healthy versus his writing when he is not feeling well, and more.
Today, there are various solutions regarding handwriting recognition such as signature verification [2], and converting between handwrite text to computer text, but in our project, we are going to combine between two different solutions for identification by writer handwrite.
We will present the solution for this challenging problem by leverages deep CNNs (Convolutional Neural Network) as a powerful model. Deep CNNs have demonstrated its effectiveness in various computer vision problems by improving state-of-the-art results with a large margin, including image classification [3–5], object detection [6, 7], face recognition [8, 9], handwriting recognition [10] etc.
We propose a two-part solution. Part 1 is DeepWriter, a multi-stream CNN, for extracting writer sensitive features that takes multiple local regions as input and is trained with softmax loss on identification. Part 2 is a feature detection algorithm in computer vision called SIFT, to detect and describe local features in the most common word in the text.
We are going to use IAM dataset, CEDAR letter dataset, and we are going to make our own dataset in Hebrew.

1.1. Organization of the paper

In section 2, background and define basic concepts. In section 3 we describe our goal in the project. In section 4 we describe the work process we performed during the semester in order to achieve our goal in the project and the final product we deliver. And last, section 5 consists of preliminary the software engineering test method we will perform on our final product.

2. Background and Related Work

Our project based on the papers “Hybrid Feature Learning for Handwriting Verification” that propose an effective Hybrid Deep Learning (HDL) architecture for the task of determining the probability that a questioned handwritten word has been written by a known writer. The second article is on “DeepWriter: A Multi-Stream Deep CNN for Text-independent Writer Identification” which proposes DeepWriter, a deep multi-stream CNN to learn deep powerful representation for recognizing writers. DeepWriter takes local handwritten patches as input and is trained with softmax classification loss.

2.1. Programing languages

Python

For our main aim to create hand write recognitions we are planning to implement it in Python language.

Python is a script language, support object-oriented approach. Being easy to use, Python attracts many programmers to create open-source libraries. In addition, Python build in a way that it can run on any platform.

We chose Python for several reasons:

(i) In Python there are already many builds in libraries for Machine Learning (ML) and Deep Learning (DL).

(ii) The articles we are rely on are already implemented their research in Python.

C++

C++ is an object-oriented programming (OOP) language, developed by Bjarne Stroustrup, and is an extension of C language. It is therefore possible to code C++ in a "C style" or "object-oriented style." In certain scenarios, it can be coded in either way and is thus an effective example of a hybrid language.

The main highlight of C++ is a collection of pre-defined classes, which are data types that can be instantiated multiple times. The language also facilitates declaration of user defined classes. Classes can further accommodate member functions to implement specific functionality. Multiple objects of a particular class can be defined to implement the functions within the class. Objects can be defined as instances created at run time. These classes can also be inherited by other new classes which take in the public and protected functionalities by default.

2.2. Deep Learning and Machine Learning Libraries

2.2.1. PyTorch

PyTorch is an open-source machine learning library based on the Torch library used for applications such as computer vision and natural language processing. PyTorch provides two high-level features:

- Tensor computing (like NumPy) with strong acceleration via graphics processing units (GPU).
- Deep neural networks built on a type-based automatic differentiation system.

2.2.2. TensorFlow

TensorFlow is a free and open-source software library for machine learning. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks. TensorFlow can run on multiple CPUs and GPUs.

2.2.3. Keras

Keras is an open-source software library that provides a Python interface for artificial neural networks, it contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code. Further-more, Keras has the ability to run on GPU.

2.2.4. PyTorch VS TensorFlow VS Keras

	PyTorch	TensorFlow	Keras
API Level	Low	High and Low	High
Architecture	Complex, less readable	Not easy to use	Simple, concise, readable
Popularity	Third most popular	Second most popular	Most popular
Speed	Fast, high-performance	Fast, high-performance	Slow, low performance
Written In	Lua	C++, CUDA, Python	Python

Similar to the working method presented in the articles on which our project based on, we decided as well to use the Keras library.

2.3. Algorithms

2.3.1. Convolutional neural network (CNN)

A convolutional neural network consists of an input layer, hidden layers and an output layer. In any feed-forward neural network, any middle layers are called hidden because their inputs and outputs are masked by the activation function and final convolution. In a convolutional neural network, the hidden layers include layers that perform convolutions. Typically, this includes a layer that performs a dot product of the convolution kernel with the layer's

input matrix. This product is usually the Frobenius inner product, and its activation function is commonly ReLU. As the convolution kernel slides along the input matrix for the layer, the convolution operation generates a feature map, which in turn contributes to the input of the next layer. This is followed by other layers such as pooling layers, fully connected layers, and normalization layers.
Convolutional layers convolve the input and pass its result to the next layer. This is similar to the response of a neuron in the visual cortex to a specific stimulus. Each convolutional neuron processes data only for its receptive field. Although fully connected feedforward neural networks can be used to learn features and classify data, this architecture is generally impractical for larger inputs such as high-resolution images.

2.3.2. Autoencoder

Autoencoder is an unsupervised artificial neural network that learns how to efficiently compress and encode data then learns how to reconstruct the data back from the reduced encoded representation to a representation that is as close to the original input as possible. Autoencoder, by design, reduces data dimensions by learning how to ignore the noise in the data.
The structure of the autoencoder is:

1. Encoder: In which the model learns how to reduce the input dimensions and compress the input data into an encoded representation.
2. Bottleneck: which is the layer that contains the compressed representation of the input data. This is the lowest possible dimensions of the input data.
3. Decoder: In which the model learns how to reconstruct the data from the encoded representation to be as close to the original input as possible.

2.3.3. Scale-Invariant Feature Transform (SIFT)

SIFT is a feature detection algorithm in computer vision to detect and describe local features in images. SIFT key points of objects are first extracted from a set of reference images and stored in a database. An object is recognized in a new image by individually comparing each feature from the new image to this database and finding candidate matching features based on Euclidean distance of their feature vectors.

2.3.4. FLANN (Fast Library for Approximate Nearest Neighbors)

FLANN is a library for performing fast approximate nearest neighbor searches in high dimensional spaces. It contains a collection of algorithms that found to work best for nearest neighbor search and a system for automatically choosing the best algorithm and optimum parameters depending on the dataset.

2.3.5. Max pooling

Max pooling function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, it reduces the computational cost by reducing the number of parameters to learn and provides basic translation invariance to the internal representation.

2.3.6. Harris Corner Detector

Harris Corner Detector is a corner detection operator that is commonly used in computer vision algorithms to extract corners and infer features of an image. Harris' corner detector takes the differential of the corner score into account with reference to direction directly, instead of using shifting patches for every 45-degree angle and has been proved to be more accurate in distinguishing between edges and corners.

2.3.7. Softmax

The softmax function is a function that turns a vector of K real values into a vector of K real values that sum to 1. The input values can be positive, negative, zero, or greater than one. Softmax function transforms them into values between 0 and 1, so that they can be interpreted as probabilities. If one of the inputs is small or negative, the softmax turns it into a small probability, and if an input is large, then it turns it into a large probability.

2.3.8. Rectified linear activation function (ReLU)

In artificial neural networks, the activation function of a node defines the output of that node given an input or set of inputs, ReLU is an activation function, a piecewise linear function, that will output the input directly if it is positive, otherwise, it will output zero.

2.4. Requirements

Our general purpose is to create an application to identify whether is a handwrite is forged or not, which using a trained Convolution Neural Network who will be the main system for the application. In addition, foreach tested person there will be a previous store data to compare with.

3. Expected Achievements

Our project depends on recent findings that used for identification handwrite, "Multi-Stream Deep CNN" (MSDC) and "Two-Channel AutoEncoder with SIFT" (TCAES). We saw that MSDC is more autonomic and gives better results but not absolute. In order to receive more reliable result, we will combine these methods. For start we will run MSDC to get the preliminary result. Next, we will run TCAES and get the degree of similarity, afterward we take both results and see if the difference between them is small, that will help us achieve more accurate results. Our project contains several unique features: (i) the use of Two Channel AutoEncoder (AE) with SIFT. Help us to determine the distances between two given inputs images. And (ii) the use of Multi-Stream with Deep CNN, this algorithm helps us to get a predict of the writer.

4. Research Process

In this section we explain about the research process we have gone through. We describe the work process we performed during the semester to address the goal set for our project.

4.1. Process

During our research, we searched for articles about handwriting recognition using Deep Learning. In order to get into the depth of the problem we had to do a comprehensive study of the field itself, understand the problem, understand the benefits in solving it, understand the existing algorithms, and what steps did he have to take in order to reach our goal.

After reviewing several articles, we chose two main articles. Once we understood the background to the problem we wanted to solve, and were motivated to do so, we began the learning process of the solution proposed in the articles. We have combined the two solutions proposed in the articles and made changes to suit our purpose.

During the study we encountered many difficulties, such as concepts and algorithms we did not know. In order to deal with those difficulties, we built a good knowledge base, and after verifying with our supervisor that the knowledge we had acquired was correct, we proceeded to the next step.

One of the things that made it especially difficult for us in the research process is that there is a lot of information on Machine Learning and CNN, a lot of different algorithms, and a lot of ways to plan the steps of the algorithm and the convolutional layers, and we had to find the most effective way with the best results.

The next steps in our research process are the design of the application and the implementation of the algorithm.

4.2. Product

In this section, we provide a high-level description of the algorithm we will implement. First, we will explain the first algorithm we relied on, called "multi-stream deep CNN". Second, we explain the second algorithm, called "Two channel AutoEncoder with SIFT".

We will present a comprehensive explanation of each of the two algorithms, pseudo-code of the algorithms, Use-case chart, and GUI.

4.2.1. Multi-stream Deep CNN Algorithm

The basic network structure is Half DeepWriter. Half DeepWriter takes as input a 113 × 113 image patch. Input handwritten text images for identifying author are with various height and width.

We are going to do the training process using Mini-Batch Gradient Descent. Gradient descent is an optimization algorithm often used for finding the weights or coefficients of machine learning algorithms. It works by having the model make predictions on training data and using the error on the predictions to update the model in such a way as to reduce the error. Mini-batch gradient descent is a variation of the gradient descent algorithm that splits the training dataset into small batches that are used to calculate model error and update model coefficients.

```
model = initialization(...)
n_epochs = ...
train_data = ...
for i in n_epochs:
    train_data = shuffle(train_data)
    X, y = split(train_data)
    predictions = predict(X, model)
    error = calculate_error(y, predictions)
    model = update_model(model, error)
```

First step, we are going to do the training process on "Half DeepWrite": The batch size was set to 256, momentum to 0.9, and weight decay to $5 * 10^{-4}$. The learning rate was initialized at 10^{-2}, and then decreased by a factor of 10 every 10^5 iterations. The learning was stopped after 400K iterations.

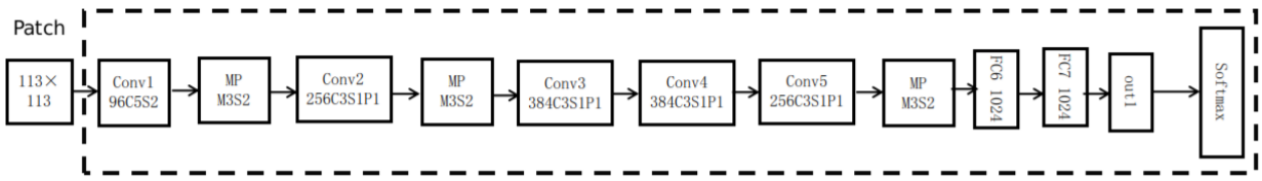


Fig1

Step two, we leverage relationship between two adjacent image patches, leading to DeepWriter structure. Now we do the training process on full DeepWriter, batch size was set to 256, momentum to 0.9, and weight decay to $5 * 10^{-4}$. The base learning rate was initialized at 10^{-3}, and then decreased by a factor of 10 every 20K iterations. The learning was stopped after 40K iterations.

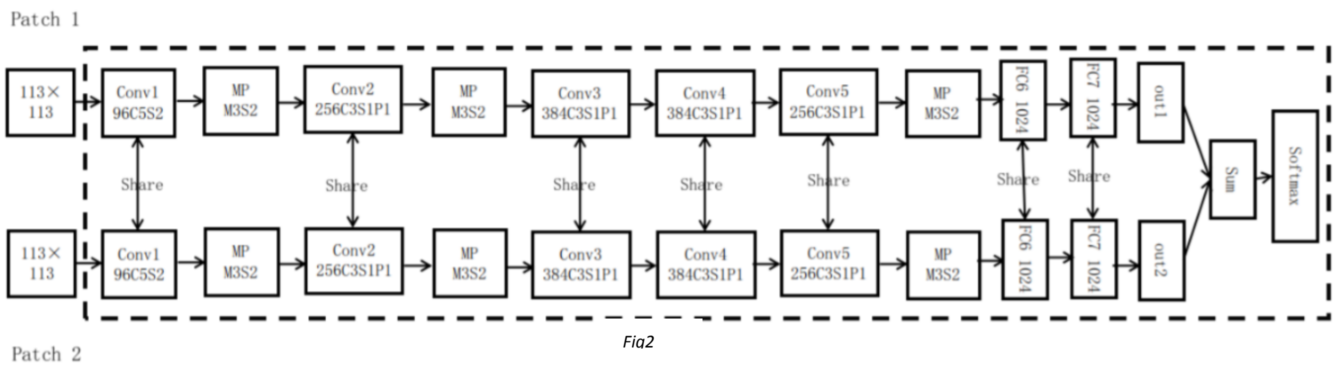


Fig2

The learning rate of softmax layer correlated to specific dataset was set to tenfold larger than base learning rate.
After the training process, we will continue to the **testing process** which we will explain about in the next paragraphs.

4.2.1.1. Multi-stream input and patch scanning

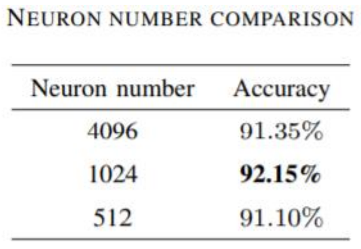
DeepWriter takes as input a pair of 113×113 image patches. Patch 2 is adjacent to Patch 1. As a result, the first step we do is resize the scanned image so that min(w,h)=113 while maintaining its aspect ratio. Secondly, 113×113 image patches are cropped randomly from the input image. Finally, image patches for testing are uniformly sampled from these cropped 113 × 113 image patches with a specific ratio.

4.2.1.2. Convolutional layers and Max-pooling

The convolution and pooling layers perform feature extraction. Given an image, the convolution is using a 'kernel' to extract certain 'features' from an input image. Each convolutional layer has some number of filters that we define with a specified dimension and that these filters convolve our image input. All convolutional layers are followed by Rectified Linear Unit layer (RELU).
In our algorithm, we have 5 convolutional layers, the αCβSσP θ like notation specifies that the convolutional layer filters the input with α kernels of size β × β with a stride of σ pixels and a padding of θ pixels. Conv1 and Conv2, layers of DeepWriter and Half DeepWriter, filter their input with smaller kernels with smaller stride compared to that of AlexNet [3]. Therefore, we decrease the kernel size and stride step of Conv1 and Conv2 layers to handle more image details.
Max pooling reduces the dimensionality of images by reducing the number of pixels in the output from the previous convolutional layer. In our algorithm, we use it 3 times, the MβSσ like notation specifies that the max-pooling layer performs max-pooling operation in a neighborhood of size β × β with a stride of σ pixels.

4.2.1.3. Fully connected layer

In our algorithm, we have two fully connected layers with 1024 neurons. All fully connected layers are followed by Rectified Linear Unit layer (RELU).
We believe that appropriate neuron number reduces the risk of overfitting. We chose the number of neurons of FC6 and FC7 through contrast experiment on validation set that made in the article we based on. We set the neuron number of FC6 and FC7 layers of DeepWriter and Half DeepWriter to 1024.
Experiment result is shown in this table image.


NEURON NUMBER COMPARISON

Neuron number	Accuracy
4096	91.35%
1024	**92.15%**
512	91.10%

Fig3

Out1 and out2, output vectors of FC7 of DeepWriter.

4.2.1.4. Sum

The vectors output, Out1 and Out2, are merged by element-wise sum operation. An element-wise operation allows you to distribute the operation over the elements of a data container, and to apply a function to the elements of a data container.

4.2.1.5. Softmax classifier and Average

Softmax function outputs a vector that represents the probability distributions of a list of potential outcomes.
Softmax turns a vector of K real values into a vector of K real values that sum to 1, the input values can be positive, negative, zero, or greater than one.
Last thing we do is average score vectors of all image patch pair.

4.2.1.6. Testing details

Given a scanned handwritten image, the testing procedure follows this pipeline: (1) scan the image to generate image patches following the strategy presented above; (2) input ith image patch pair or image patch into DeepWriter or Half DeepWriter to compute score vector F_i (3) compute final score of jth writer $F_j = \frac{1}{N}\sum_{i=1}^{N} F_{ij}$, where N denotes the number of image patches; (4) return the writer with highest score.
Noting that the score vector outputted by DeepWriter can be treated as a probability distribution over all writers, we thus average score vectors of all image patch pairs or image patch to construct the final prediction of input image.

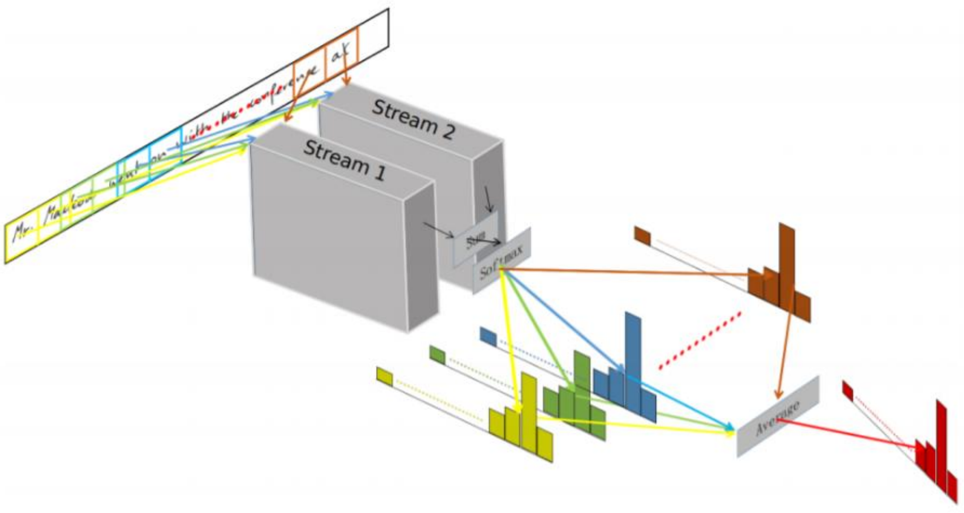


Fig4

4.2.1.7. Pseudo code

```
                DeepWriter(textImage)
1. Resize textImage so that min(w,h)=113
2. Crop textImage to 113X113 image patches.
3. For each patch i to N do :
     3.1 Convolutional layer 1 – 96 kernels of size 5X5 and stride 2
     3.2 Max Pooling – size 3X3 with stride 2
     3.3 Convolutional layer 2 – 256 kernels of size 3X3 with stride 1 and 1 pixels padding
     3.4 Max Pooling – size 3X3 with stride 2
     3.5 Convolutional layer 3 – 384 kernels of size 3X3 with stride 1 and 1 pixels padding
     3.6 Convolutional layer 4 – 384 kernels of size 3X3 with stride 1 and 1 pixels padding
     3.7 Max Pooling – size 3X3 with stride 2
     3.8 Fully Connected layer – with 1024 neurons
     3.9 Fully Connected layer – with 1024 neurons
     3.10      Save output vectors as out_i
4. For each pair of patches (adjacent) do :
     4.1 Merge by element wise sum operation
5. Softmax classification loss
6. Compute final score - $f_j = \frac{1}{N}\sum_{i=1}^{N} f_{ij}$
7. Average score vectors
8. Return the writer with highest score
```

4.2.2. Two channel AutoEncoder with SIFT

In the part of the Two channel Autoencoder (AE) with SIFT we use to find the similarity between two writers; the algorithm contains two phases that operating in parallel, in one part we operate the two channel AutoEncoder and in the second part we will operate the SIFT part. Both parts get as input image in grayscale.

4.2.2.1. Two channel AE

Since the input image is in grayscale it is necessary to get more robust features that will describe the image better; this we will be doing by reconstructing the image using an AE. The AE contains an encoder and a decoder and gets an output of vector in size of 128.
The encoder architecture contains two Convolution layers with 64 and 1 kernels respectively, after them there are 3 fully connected layers with 4096, 1024 and 128 neurons respectively.
The decoder architecture it's simply reversed to maintain balance between the encoder and the decoder. So, we first pass through the Fully connected layer with 128, 1024, 2048 neurons respectively and then through Convolution layers with 1 and 64 kernels, although here we append Cubic Spline Interpolation after the Convolution layers in order to resize the output array to size of 64x64.

4.2.2.2. Pseudo code

```
Fully Connected layer (FC):
Input: vector
Output: matrix

    1.    Init fully connected layer
          a.  Layer1 ← 4096 neurons
          b.  Layer2 ← 1024 neurons
          c.  Layer3 ← 128 neurons
```

```
Encoder:
Input: image (shape 64X64)
Output: vector
1    K ← Kernel size: (3X3)
2    run convolutional (64 · K)
3    resC ← run convolutional (1 · K)
4    resFC ← FC(resC)
5    resD ← Dense(resFC)
6    vector ← flatten(resD)
```

```
Decoder:
Input: vector
Output: vector (of 128)
1    K ← Kernel size: (3X3)
2    run convolutional (1 · K)
3    resC ← run convolutional (64 · K)
4    resCI ← CI(resC)
5    vector ← FC(resCI)

CI = cubic interpolation
```

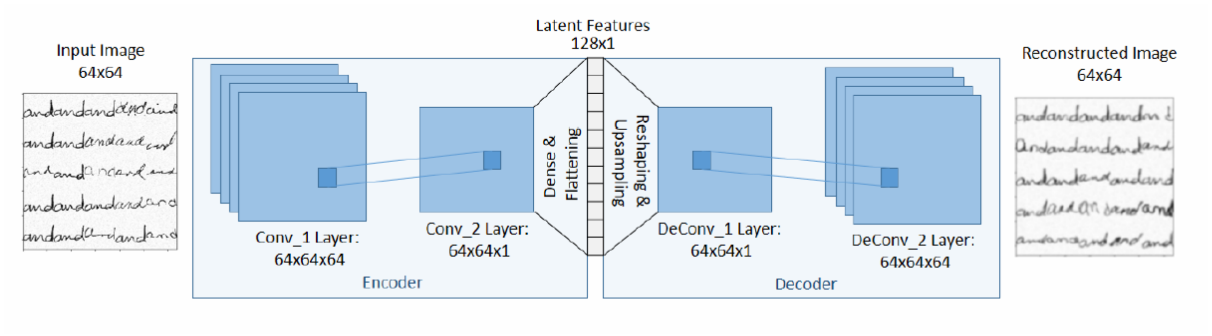


Fig5

4.2.2.3. SIFT

In this part we use the algorithm SIFT, the output is variable number of key points. Each key point is a result of a stable maxima across all the differences of blurred scales. For the blur produce we are convolving the image with a gaussian kernel. Then every outlier from the resulting of the key points are removed by using low contrast detector and Harris corner detector. For each key point SIFT creates scale of rotational and orientational invariant image descriptors. The output of each descriptor is 128. After we have all the key points from our input images, we use the nearest neighbor mapping algorithm FLANN to match n key points. The matching process runs on the basis of strokes (connecting, beginning, ending), slantness, flourishments and embellishments. Eventually we get a feature vector of fixed size n*128 which we return.

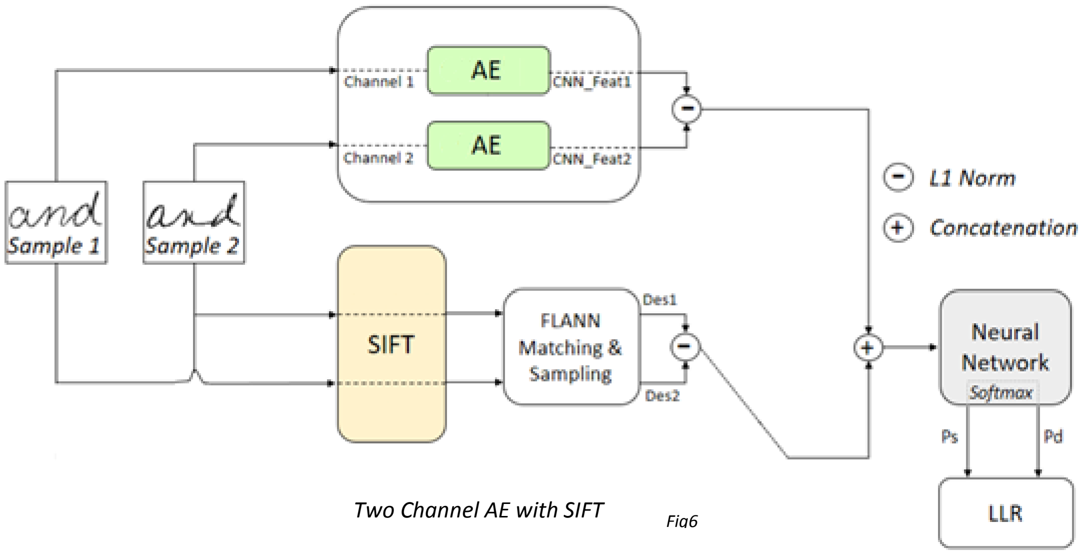
4.2.2.4. Pseudo code

```
SIFT
Input: image1, image2 (shape 64X64)
Output: vector (of n*128)
    1.  X ← keypoints(image1)
    2.  Y ← keypoints(image2)
    3.  Return X-Y

keyPoints:
Input: image (shape 64X64)
Output: vector (of n*128)
    1       res ← run gaussian kernel (input)
    2       add to list keypointList ( SIFT(res) ) // SIFT return variable number of keypoints
    3       foreach keypoint in keypointList:
                a.  run low contrast detector
                b.  run Harris corner detector
               //length of each keypoint is 128
    4       X ← on keypointList run FLANN //output: n points from keypointList
    5       return X
// FLANN algorithm uses to find nearest neighbour mapping
// FLANN finds n points
```

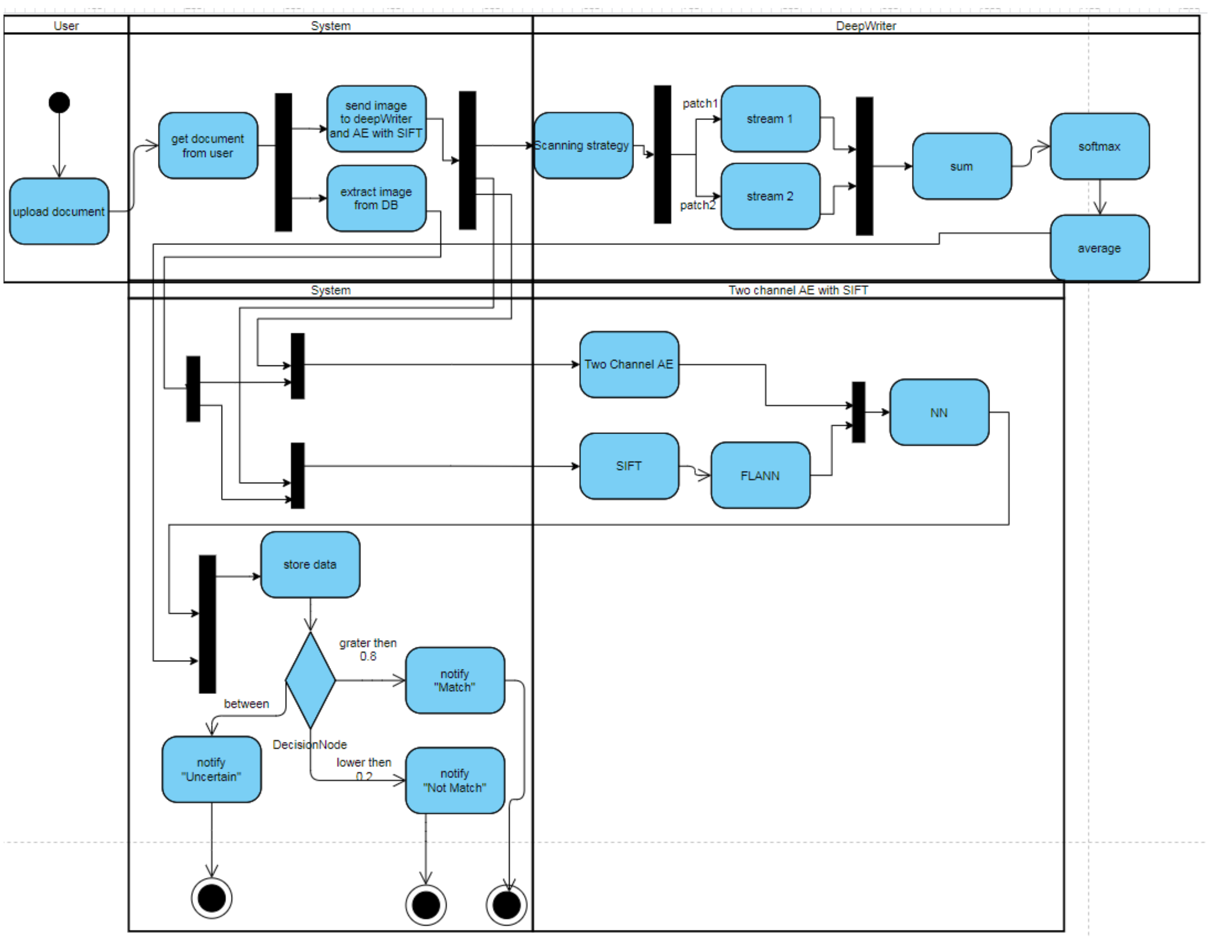
4.2.2.5. Summery

After we activate AE and SIFT algorithms, we get two feature vectors, vector of 128 and vector of n*128 respectively, those vectors are now chained and then fed as input to a two-class neural network classifier. For the activation function we use softmax. The softmax provide us the degree of similarity between the pair of input samples.

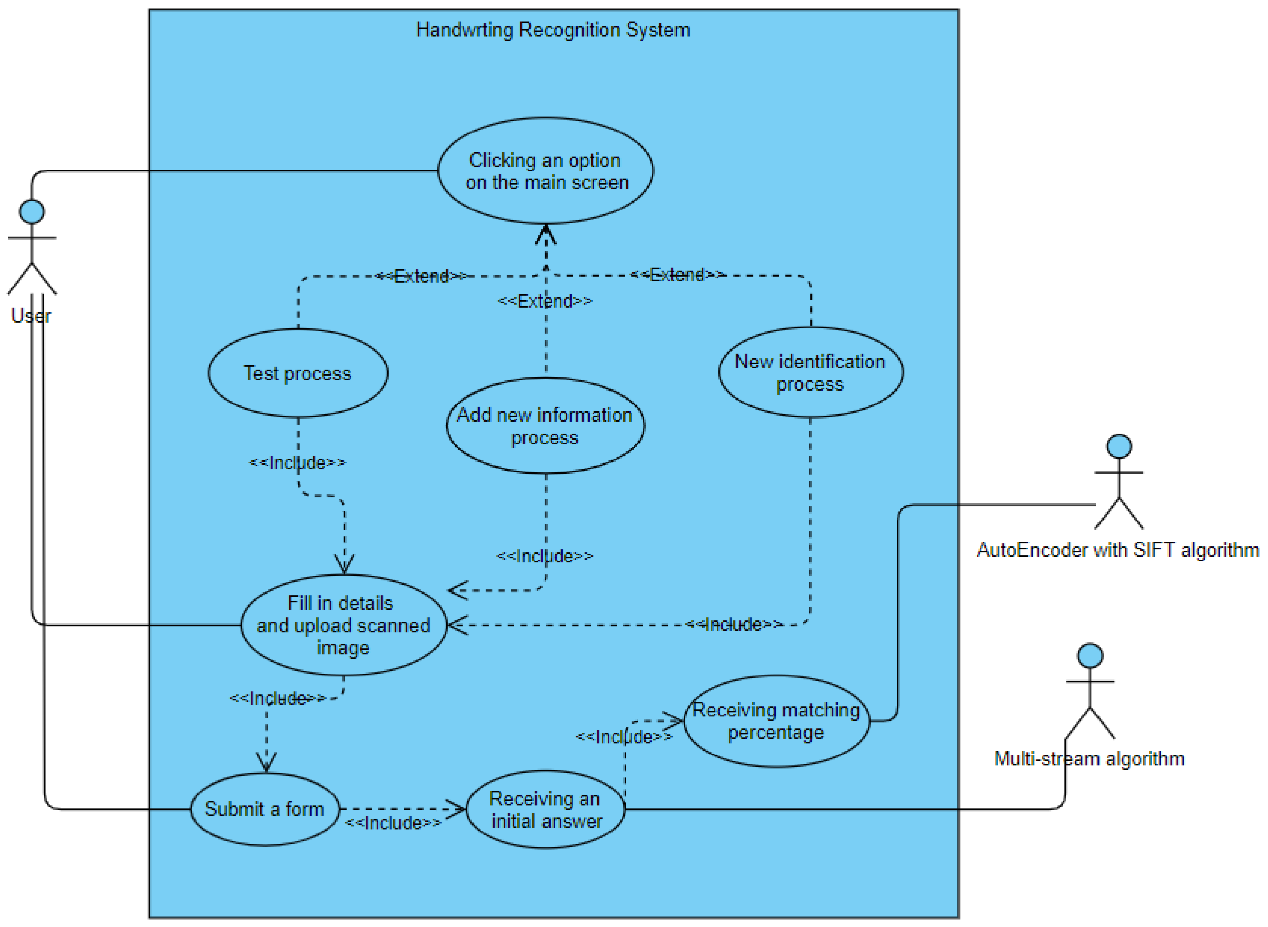


Two Channel AE with SIFT *Fig6*

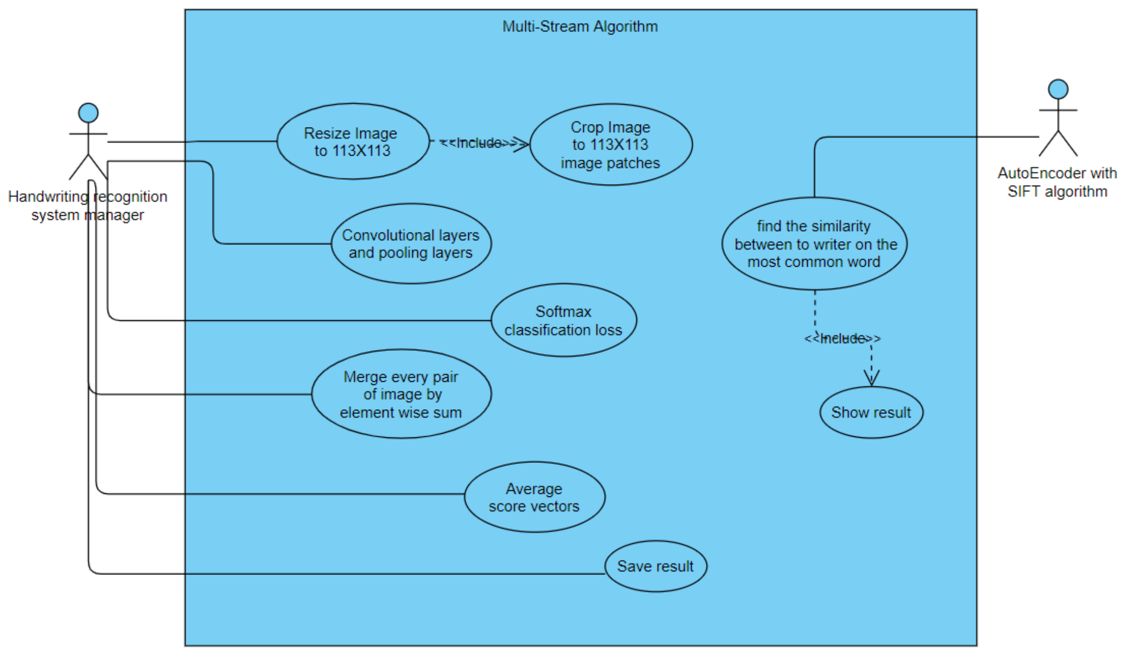
4.2.3. Activity Diagram



4.2.4. Use Case



Handwrting Recognition System
User
Clicking an option on the main screen
<<Extend>>
<<Extend>>
<<Extend>>
Test process
Add new information process
New identification process
<<Include>>
<<Include>>
<<Include>>
Fill in details and upload scanned image
<<Include>>
Receiving matching percentage
<<Include>>
Submit a form
<<Include>>
Receiving an initial answer
AutoEncoder with SIFT algorithm
Multi-stream algorithm



Multi-Stream Algorithm
Handwriting recognition system manager
Resize Image to 113X113
<<Include>>
Crop Image to 113X113 image patches
Convolutional layers and pooling layers
Softmax classification loss
Merge every pair of image by element wise sum
Average score vectors
Save result
AutoEncoder with SIFT algorithm
find the similarity between to writer on the most common word
<<Include>>
Show result

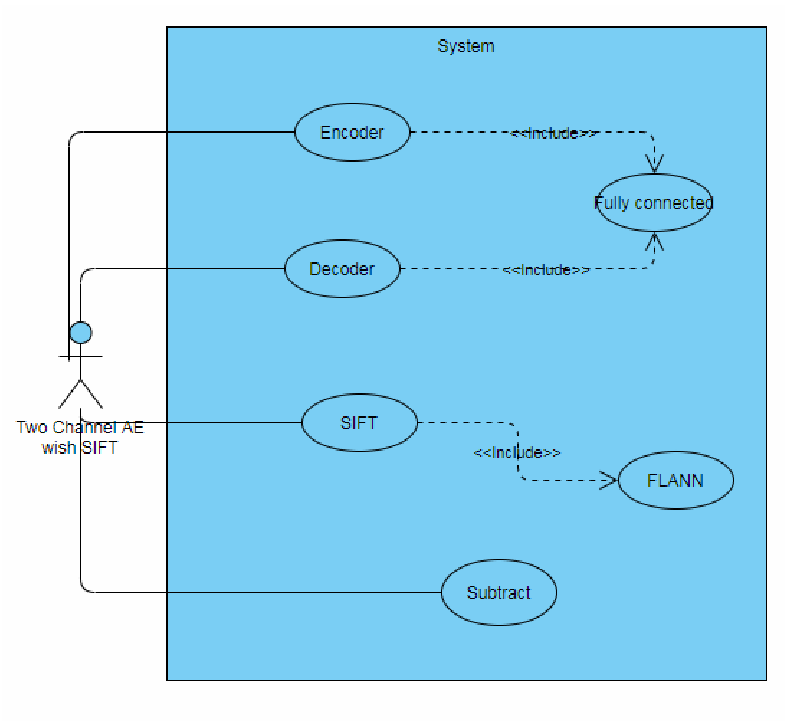
Use case 1: Convolutional layer and pooling layer.
Preconditions: Resize image to 113X113 include Crop image to patches.

Use case 2: Softmax classification loss.
Precondition: Use case 1

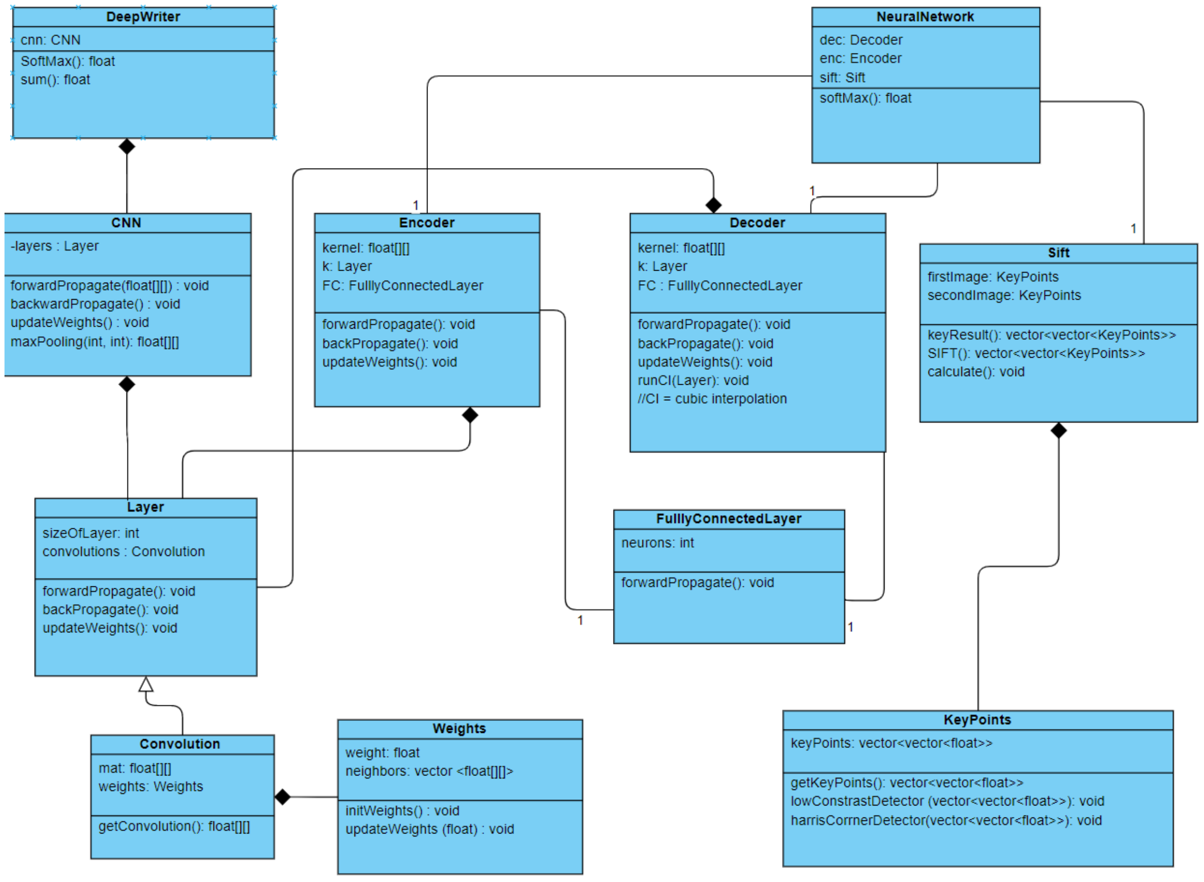
Use case 3: Merge every pair of images by element wise sum.
Precondition: Use case 2

Use case 4: Average score vectors.
Precondition: Use case 3

Use case 5: Save result.
Precondition: Use case 4



4.2.5. Class Diagram



4.2.6. User interface

Main screen: In our main screen, we have three user options. "New identification process" button, "Add new information" button, and "Test" button. Each button directs the user to a new screen.

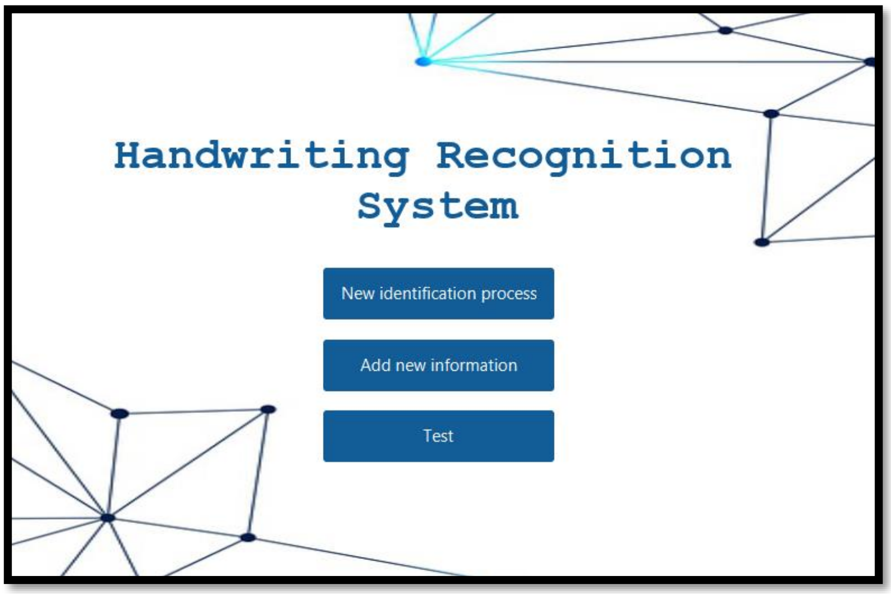


Fig7

New identification screen: In this screen, the user will enter ID, full name, and a scanned image of the person he wants to investigate. The scanned image will appear on the right.

By clicking "Check", the system will check if the handwriting in the scanned image belongs to the person whose details we entered. The results appear in a new screen called "Results".

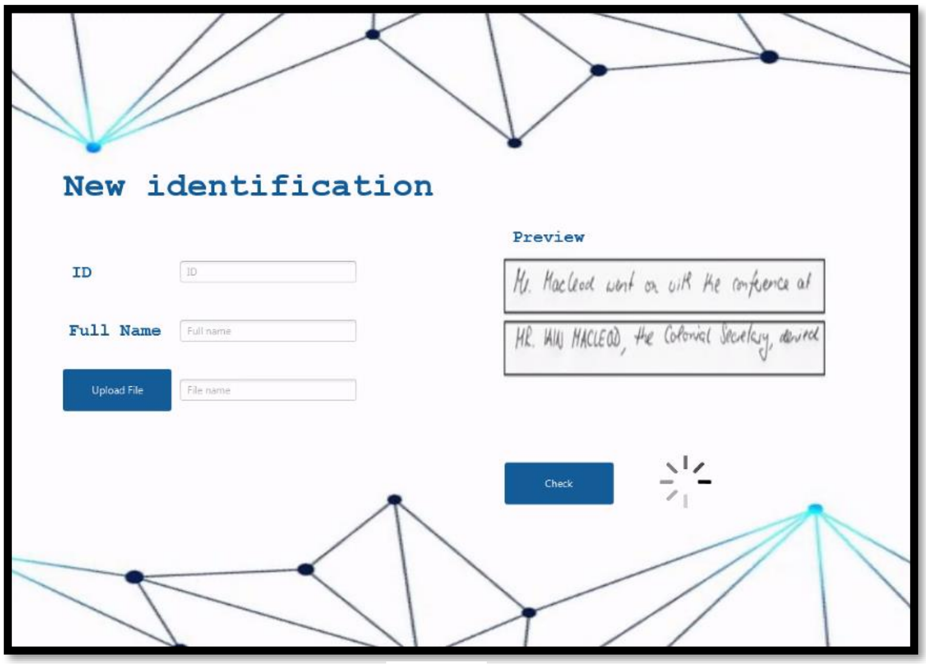

New identification
ID
ID
Full Name
Full name
Upload File
File name
Preview
Mr. Macleod went on with the conference at
MR. IAIN MACLEOD, the Colonial Secretary, denied
Check

Fig8

Add new information screen: In this screen, the user enters the ID number, full name, and scanned document of a new person who is not in the Dataset. In addition, there is an option to search by ID for an existing person and add a new document to it. By clicking "Save", the details are saved in our dataset and a success message will appear.

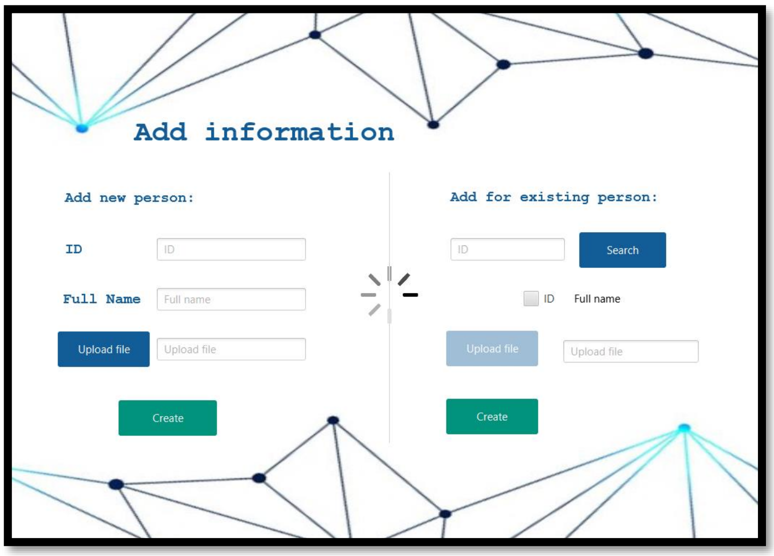


Fig9

Test screen: In this screen, we give the user an option to upload two scanned images, and check whether the two parts of the image belong to the same person. The results appear in a new screen called "Results".

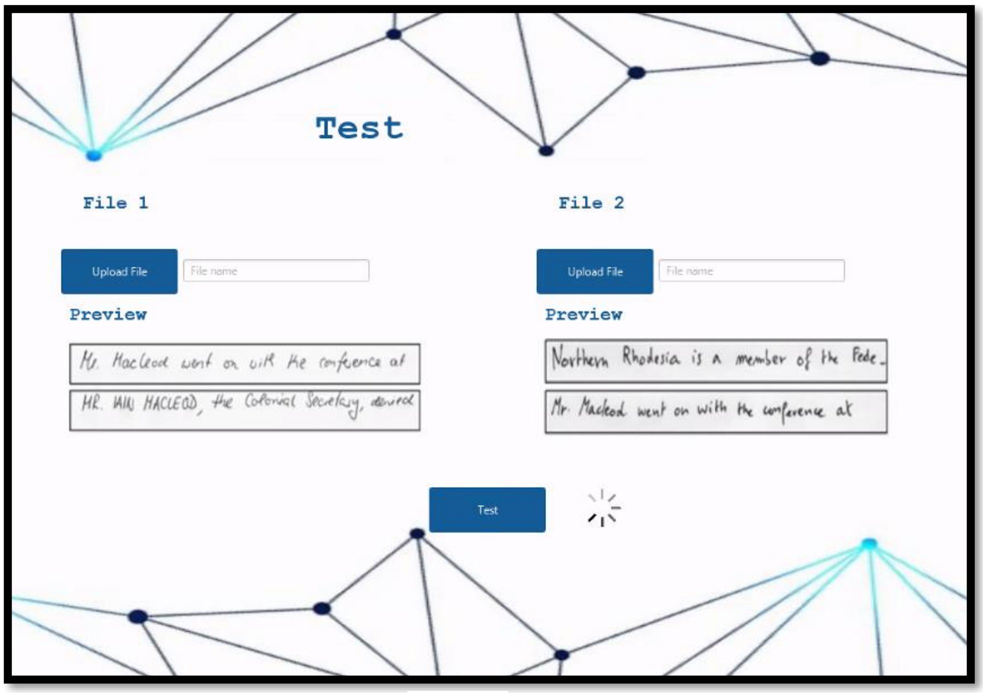
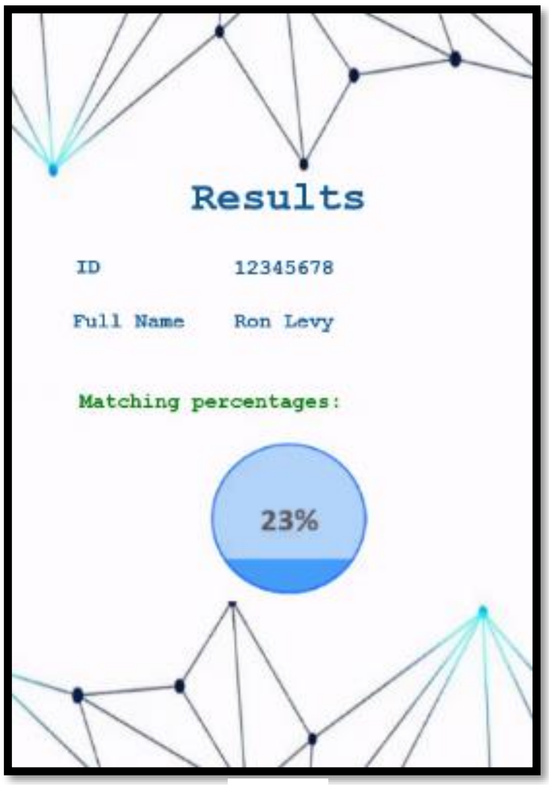

Test
File 1
File 2
Upload File
File name
Upload File
File name
Preview
Preview
Mr. MacLeod went on with the conference at
MR. IAIN MACLEOD, the Colonial Secretary, denied
Northern Rhodesia is a member of the Fede-
Mr. Macleod went on with the conference at
Test

Fig10

Results screen: In this screen, we display the full name and ID number of the person the user entered and show this person's matching percentage compared to the handwriting in the uploaded image.



Results

ID	12345678
Full Name	Ron Levy

Matching percentages:

23%

Fig11

5. Evaluation/Verification Plan

In order to test the correctness desired operation of our system, we chose to take a system testing approach. System testing is a black box testing method used to evaluate the completed and integrated system, as a whole, to ensure it meets specified requirements. In this testing method the functionality of the software is tested from end-to-end. We chose this type of testing because we want to make sure our system does what it is designed to do. The table below (table1) shows the tests we will perform on the system.

No.	screen	Input	Description	Expected result
1	New identification	ID: not-exist Full name: not-exist Document: uploaded	The user tries to start a new identification process to non-exist person	The system will notify to the user that there are no documents for this person
2	New identification	ID: 123456 Full name: "Yosi Balol" Document: uploaded	The user tries to start a new identification process to exist person	The system will open the "Result" window and display the degree of similarity
3	New identification	ID: 123456 Full name: blank Document: uploaded	The user tries to start a new identification process without fill one of the fields	The system will notify the user to fill all the fields
4	Test	Document1: file1 Document2: file1	The user used the 'Test' function and enter two identical documents	The system will open the "Result" window and display high degree of similarity
5	Test	Document1: file1 Document2: blank	The user used the 'Test' function and upload only one document	The system will notify the user to fill all the fields
6	Test	Document1: file1 Document2: file2	The user used the 'Test' function and enter two unidentical documents	The system will open the "Result" window and display low degree of similarity
7	Add information	ID:123456 Full name: "Yosi Balol" Document: uploaded	The user tries to upload new document to exist person in the system with legal inputs	The system updates the database
8	Add information	ID: blank Full name: blank Document: uploaded	The user upload document to test without insert ID and name	The check button will be disabled
9	Add information	ID:123456 Full name: "Yosi Balol" Document: blank	The user tries to test document without upload document to compare to	The check button will be disabled
10	Add information	ID: not exist	The user tries to search a non-exist person	The system will display a message "the person is not-existing"
11	Add information	ID:123456 Full name: "Yosi Balol" Document: uploaded	The user tries to create new person with details of existing person	The system will notify the user that the person is already exist

12	Add information	ID:123456 Full name: "Yosi Balol" Document: uploaded	The user tries to create new person.	The system display message "The new person created"

Table1

References

[1] Offline signature verification using DAG-CNN. Javier O. Pinzón-Arenas, Robinson Jiménez-Moreno, César G. Pachón-Suescún.

[2] Bulacu M, Schomaker L. Text-independent writer identification and verification using textural and allographic features[J]. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 2007, 29(4): 701-717.

[3] Krizhevsky, A., Sutskever, I., Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 10971105).

[4] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition[J]. arXiv preprint arXiv:1409.1556, 2014.

[5] He K, Zhang X, Ren S, et al. Deep Residual Learning for Image Re

[6] Girshick R. Fast r-cnn[C]. Proceedings of the IEEE International Conference on Computer Vision. 2015: 1440- 1448.

[7] Ren S, He K, Girshick R, et al. Faster R-CNN: Towards real-time object detection with region proposal networks[C]. Advances in Neural Information Processing Systems. 2015: 91-99

[8] Sun Y, Wang X, Tang X. Deep learning face representation from predicting 10,000 classes[C]. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2014: 1891-1898.

[9] Sun Y, Liang D, Wang X, et al. Deepid3: Face recognition with very deep neural networks[J]. arXiv preprint arXiv:1502.00873, 2015.

[10] Ciresan D, Meier U, Schmidhuber J. Multi-column deep neural networks for image classification[C]. Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on. IEEE, 2012: 3642-3649