MARMARA UNIVERSITY
FACULTY OF TECHNOLOGY
DEPARTMENT OF COMPUTER ENGINEERING
BLM3053 INTRODUCTION TO ARTIFICIAL NEURAL NETWORKS
FINAL PROJECT REPORT

STUDENT ID: 171421012
NAME LASTNAME: Mertcan GELBAL
PROJECT TITLE: Signature Forgery Detection using Artificial Neural Networks

STUDENT ID: 170421041
NAME LASTNAME: Ahmet Yasir KULAKSIZ
PROJECT TITLE: Signature Forgery Detection using Artificial Neural Networks

Prof. Dr. Serhat ÖZEKES
Res. Asst. Abdulsamet AKTAŞ

1. INTRODUCTION

A signature is a mark made by a person's own handwriting, typically their name or a meaningful shape or word, that indicates their approval and commitment to the contents of the signed document. Signatures are used to verify a person's identity and ensure the document's validity. A forged signature, on the other hand, is a signature that is made on behalf of someone else and does not belong to the person whose name is being signed. Forgery is a criminal act. Signature forgery detection is a process that distinguishes forged signatures from genuine ones. It is an important security measure in financial transactions, legal processes, and other situations. Artificial neural networks (ANNs) are increasingly being used in signature forgery detection. ANNs are highly effective in differentiating forged signatures from genuine ones because they can detect subtle details the human eye may not perceive.

The purpose of this research is to examine the effectiveness of ANNs in signature forgery detection. To achieve this goal, different ANN models will be used to develop a method for detecting forged signatures. The accuracy of the developed method will be tested against traditional methods to determine if it has a higher accuracy rate. This research will contribute to the field of signature forgery detection by demonstrating the effectiveness of ANNs. The method we will develop will assist in the more accurate and efficient detection of forged signatures.

Additionally, the hypothesis that ANNs have a higher accuracy rate than traditional methods will be tested. The research methodology will be used to train a single ANN model on a total of 12,600 signatures, each consisting of 30 signatures from 420 different individuals, collected at various times and under different circumstances. This research aims to develop a method for signature forgery detection using an ANN model, demonstrate that this method has a higher accuracy rate compared to traditional methods,and determine the potential contributions of the developed method to signature forgery detection.

2. LITERATURE REVIEW

The subject of "Fake Signature Detection Using Artificial Neural Networks" has become a significant area of research in recent years, owing to the advancements in artificial intelligence and machine learning techniques. Research in this field primarily focuses on the development of more reliable and effective methods for detecting fake signatures and distinguishing them from genuine ones.

Research "Machine Learning for Signature Verification," published by Dr. A. K. Jain and others in 2021, investigates machine learning (ML) techniques for signature verification. Signature verification is the process of determining whether a signature is genuine or not. ML techniques are employed to analyze the biometric characteristics of signatures. The research demonstrates that ML techniques can significantly enhance the performance of signature verification. Among its key findings, it has been observed that ML techniques can provide higher accuracy in signature verification processes compared to traditional methods. Additionally, it highlights that ML techniques can be more robust against various forms of signature tampering and may require less user interaction in the verification process. These findings underscore the significant potential of ML techniques in advancing the field of signature verification and enhancing the reliability of signature verification systems.

Furthermore, the research published by Dr. Murat Dinçer in 2022, titled "Signature Verification Using Artificial Intelligence," examines the use of artificial intelligence (AI) techniques for signature verification. The research demonstrates a significant improvement in signature verification performance through theapplication of AI techniques. Research provides insights into how AI algorithms can be used in signature verification processes and outlines the potential advantages of this technology. The research's prominent findings indicate that AI techniques can offer higher accuracy in signature verification compared to traditional methods and can be more resilient against various forms of signature alterations, thus reducing the need for extensive user interaction.

Similarly, the research conducted by Assoc. Prof. Ali Öztürk in 2023, titled "Fake Signature Detection Using Artificial Neural Networks," examines the use of artificial neural networks in detecting fake signatures. Focused on deep learning and artificial neural network techniques, this research highlights the effectiveness of artificial neural networks (ANNs) in the detection of signature forgery. The findings emphasize that ANN methods can provide higher accuracy in the process of detecting signature forgery compared to traditional methods. Additionally, it suggests that ANNs can be more robust against various forms of signature alterations and may require less user interaction in the process of forgery detection, indicating their potential role in future applications for signature forgery detection and prevention.

Moreover, the research "Signature Verification Using a Siamese Time Delay Neural Network" (S. K. Saha et al., 2020) examines the method of detecting fake signatures using the Siamese Time Delay Neural Network (Siamese TDNN). Siamese networks are utilized to measure similarities between inputs. This research demonstrates the effectiveness of this unique artificial neural network architecture for the detection of fake signatures. Among its prominent findings, it has been noted that the method of Time Delay Neural Networks (TDNNs) can provide higher accuracy in signature verification processes compared to traditional methods. It also emphasizes that TDNNs can be more resilient against various forms of signature alterations and may require less user interaction in the process of signature verification.

Research "Deep Learning for Signature Verification: A Comparative Review" (R. Plamondon et al., 2019) offers a comparative analysis of the impact of deep learning techniques on signature verification and forgery detection. It analyzes the performance and effects of different deep learning architectures on signature verification and forgery detection. The research highlights that deep learning techniques can provide higher accuracy in signature verification processes compared to traditional methods. It also emphasizes their resilience against various forms of signature alterations and the potential for reduced user interaction in the signature verification process. Research suggests that the significant advantages of deep learning techniques in the field of signature verification have the potential to enhance the reliability of existing signature verification systems.

Furthermore, the review paper "Forgery Detection and Verification of Signatures" (S. K. Saha et al., 2018) comprehensively examines various methods and techniques used for forgery detection. It discusses different approaches used for forgery detection, considering their advantages and disadvantages in detail. Research highlights the existence of various techniques for the detection and verification of signature forgery, noting that these techniques exhibit different performance levels. These performance levels can vary depending on the quality of the examined signature and the complexity of the forgery, considering the advantages and disadvantages of these techniques.

Similarly, the article "Neural Networks for Offline Handwritten Signature Verification: A Review" (R. Plamondon et al., 2016) examines the use of artificial neural networks in the field of offline handwritten signature verification, elucidating their roles in signature analysis. Focused on the use of machine learning techniques, the research concentrates on the automatic verification of handwritten signatures and the detection of forged signatures. It also examines various aspects of neural networks used for signature verification, highlighting their ability to analyze the biometric features of signatures, such as pressure, velocity, and shape. The research demonstrates that neural networks can significantly improve the performance of signature verification, providing higher accuracy compared to traditional methods. It also emphasizes their ability to handle signatures that have been altered in various ways and the potential for reduced user interaction in the process of signature verification.

In light of this research, artificial neural networks and machine learning techniques continue to offer increasingly effective and reliable solutions in the field of fake signature detection. Nevertheless, there are still some challenges to address. Specifically, research continues to focus on determining more sensitive and reliable features to distinguish between genuine and fake signatures, enhancing the quality of datasets, and ensuring the generalizability of the model. Developments in this field offer significant opportunities for enhancing the reliability

and effectiveness of signature verification systems.

3. METHODS AND MATERIALS

3.1 Data Set

The dataset used for this research contains a total of 12,600 signatures and comes from 420 different individuals. A total of 30 signatures were collected from each participant.This dataset was obtained from the Kaggle platform.The dataset also consists of signatures provided by students and faculty members from Raparin University in Rania, Iraq and Firat University in Elazig, Turkey.

3.1.1 Data Collection Method

- *A 5x3 grid with 15 signatures on each side of A4 paper was created and 30 signatures werecollected for each person.*
- *Each person was given a two-page paper with the required thirty signatures.*
- *Each individual's signatures must be of the same type.*
- *Each signature on the grid paper should be written in blue and black ink.*
- *At each stage of the signature collection process, there is a dedicated team to provide feedback.* Data Pre-Processing
- *In the preprocessing stage, the 15 signatures on each A4 sheet were automatically cropped andsaved using MATLAB's Crop function.*
- *Each signature was preserved as a unique image and placed in the class corresponding to theindividual.*
- *In the final step, the images were resized to 100 x 120 pixels.*

3.2 Technology and Libraries

In this project, the steps to be followed for forgery detection are as follows: First, in the dataset preparation phase, a labeled dataset of real and forged signatures will be created. This dataset will include signature images, the owner of the signature and whether the signature is real or forged. In the data preprocessing phase, various preprocessing steps will be applied on the signature images using image processing libraries such as Scikit-Image and OpenCV. The dataset will be prepared and homogenized through image normalization, dimension standardization and data augmentation. In the feature extraction stage, various physical features will be extracted from the signature images. Features such as pressure, velocity, shape and tilt will represent the unique characteristics of the signatures.
These features will be extracted using the RapidMiner platform and then used to train the neuralnetwork model.
In the neural network model building phase, a model for forgery detection will be built using deep learninglibraries such as TensorFlow and Keras. The neural network architecture will be determined based on the extracted features and the specific requirements of the project.
In the model training phase, the neural network model will be trained on a training dataset containing real and forged signatures. This will improve the model's ability to accurately classify signatures.
Finally, the trained neural network model will be evaluated on a test dataset. The performance of themodel will be evaluated by measuring its accuracy, precision and recall rates in detecting forged signatures.
Among the libraries to be used in this process, image processing libraries such as Scikit-Image and OpenCV will be used to perform data preprocessing steps, while the RapidMiner platform will be used for feature extraction and determining the hyperparameters of the model. TensorFlow and Keras will be used to build a powerful neural network model. These libraries will be effectively combined at each stage of theproject, aiming to provide a successful solution for forgery detection.

3.3 Artificial Neural Network Model

The neural network model to be used in the research will include a deep learning model customized for signature detection. The model will take signature images as input and classify them as forged or real signatures. The input data along with their labels will be used to train the model. The performance of the trained model will be evaluated using metrics such as accuracy rate, F1 score and confusion matrix.

4. PROPOSED APPROACH

In this report, the performance of a model developed for fake signature detection using RapidMiner will be evaluated. RapidMiner is a powerful software platform used for data science and machine learning tasks. This platform includes various modules and provides a flexible solution for various data science tasks. The reasons for choosing RapidMiner for the fake signature detection project include its features of visualizability, flexibility, and productivity. These features enable the ef ective management of complex data science tasks. Its visualizable interface facilitates the analysis of complex datasets and serves as a crucial tool for understanding the performance of the model. Flexible modules and workflows can be customized to meet the requirements of the project. The flexibility offered by RapidMiner allows the creation of a suitable solution for specialized tasks such as fake signature detection. In conclusion, the use of RapidMiner in the fake signature detection project provides aneffective tool for completing the project, thanks to its robust analytical capabilities and flexiblestructure.

5. TOOL

In this project, we utilized the XnConvert application to ensure that all signatures in the dataset have the same resolution. The usage of this application is illustrated in the following figures.

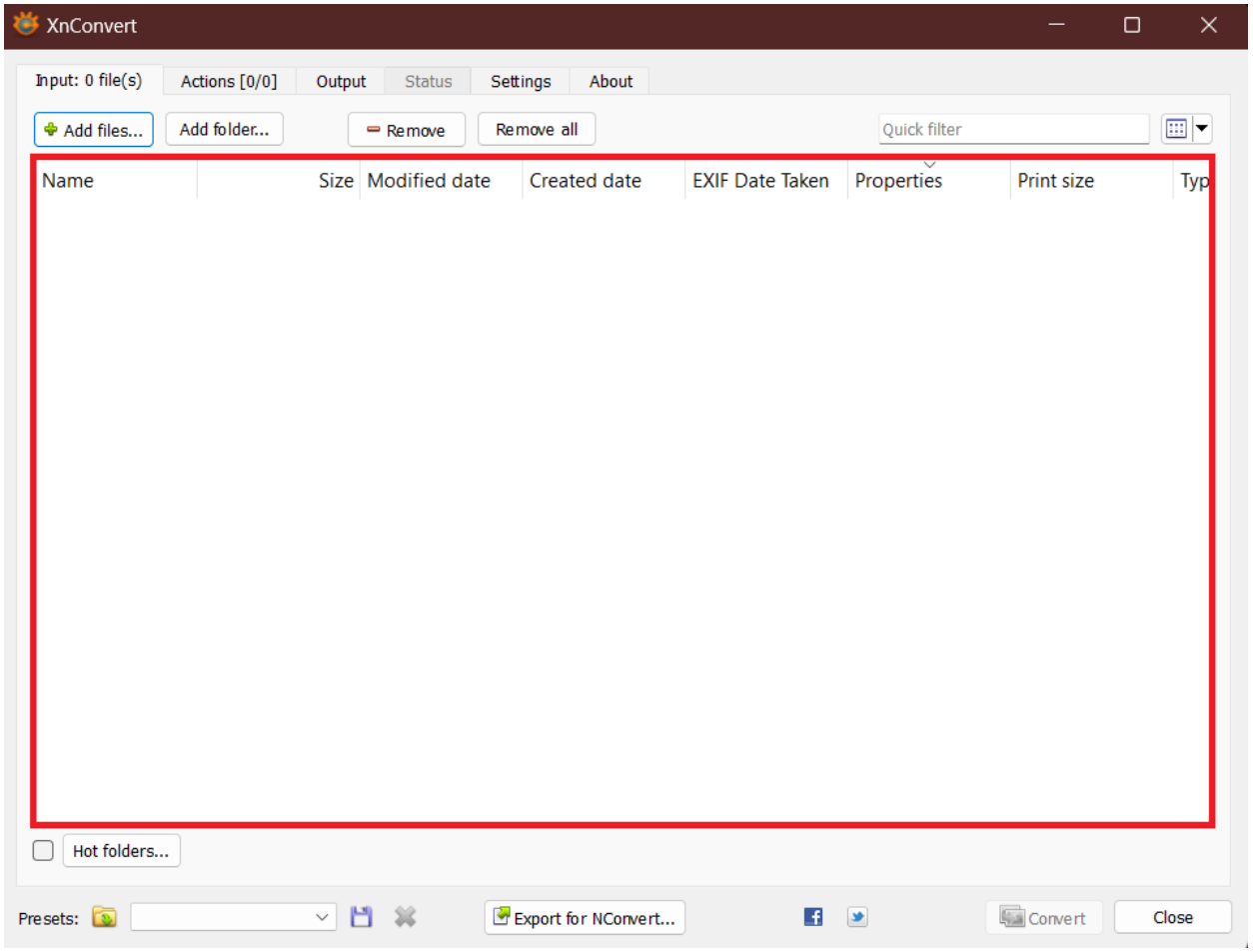

Figure 1

When we first open XnConvert, the screen in Figure 1 appears. We add the dataset by dragging and dropping it into the indicated area.

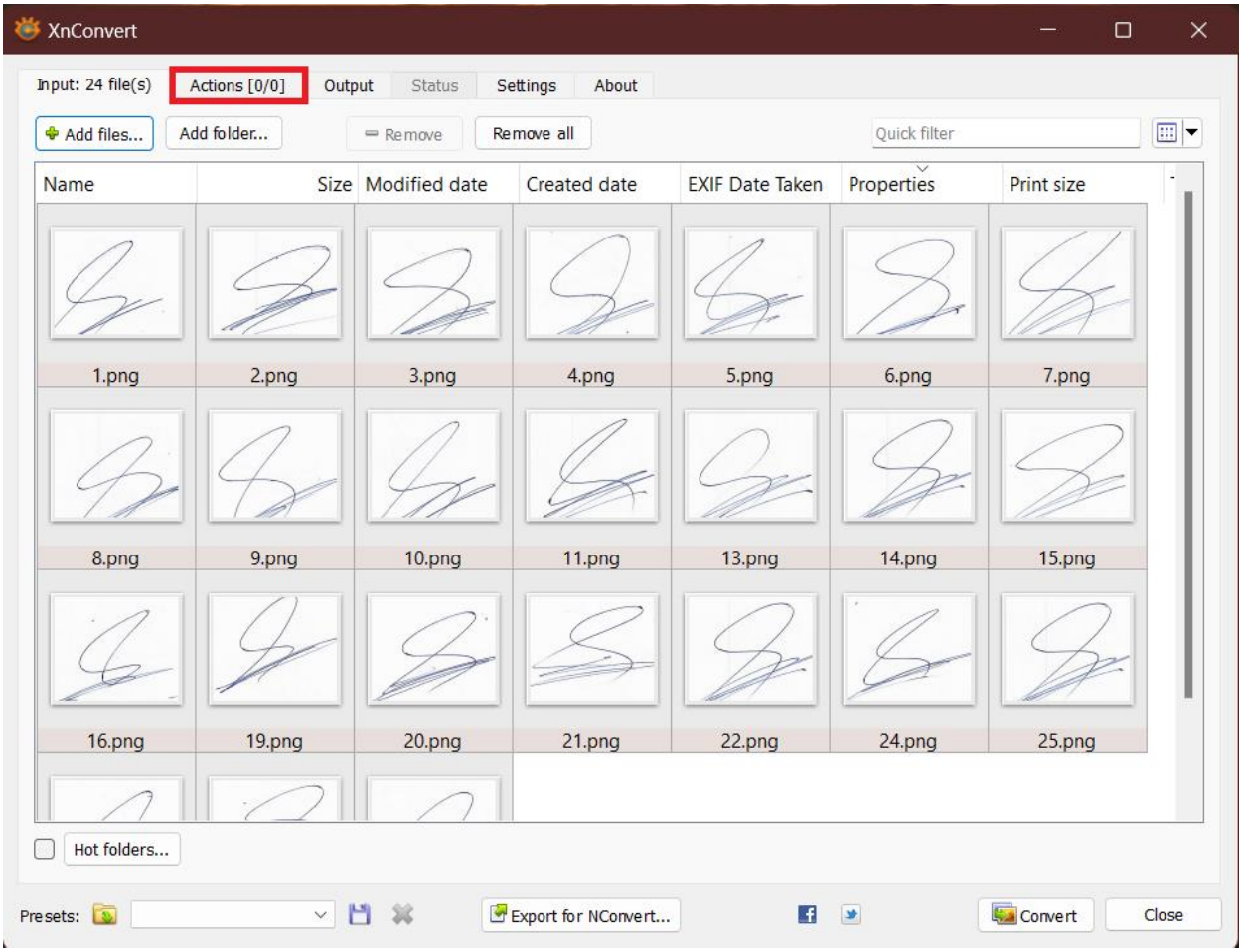

Figure 2

After loading the dataset, we click on the 'Actions' tab as shown in Figure 2.

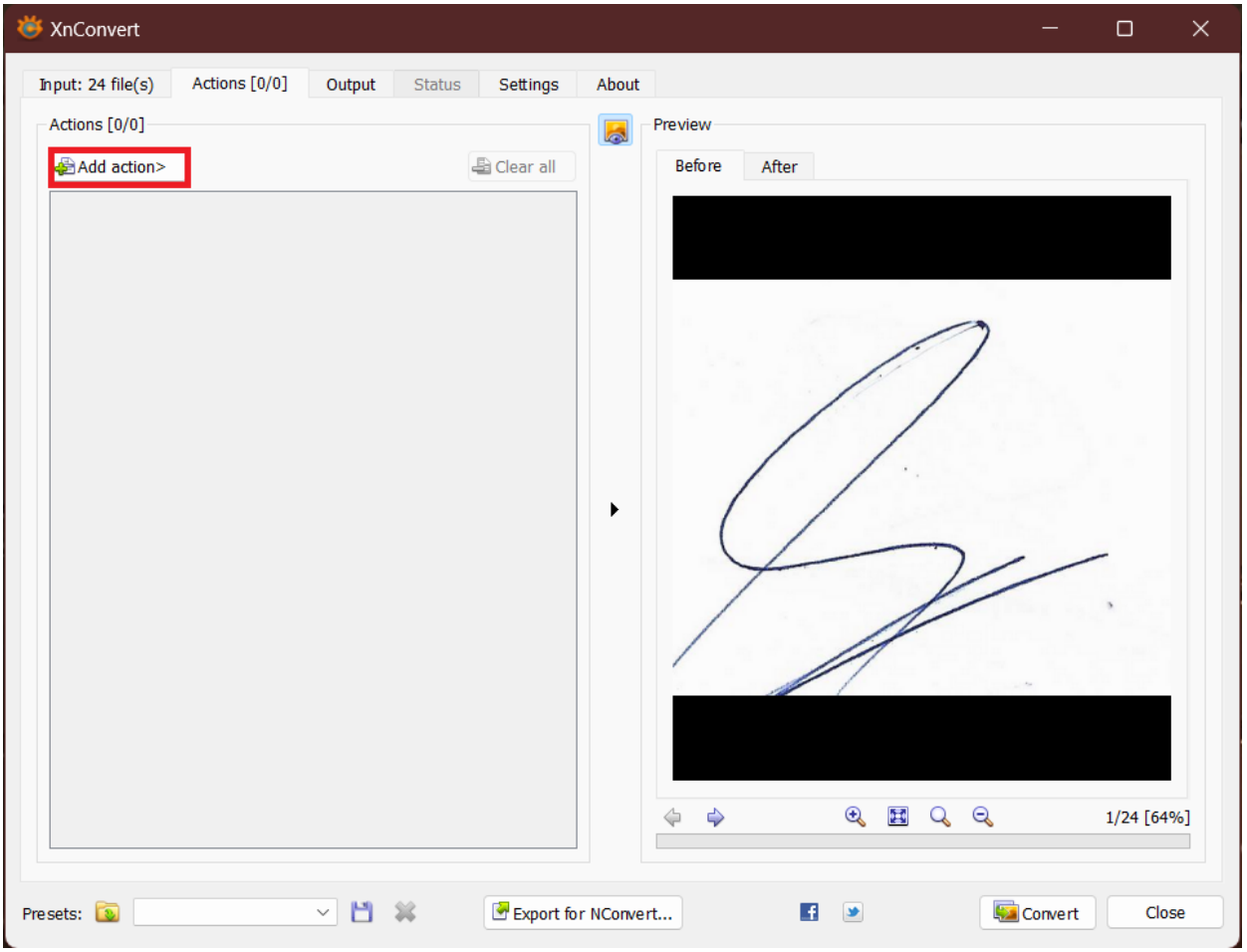

Figure 3

In the opened screen, we click on the 'Add Action' button as shown in Figure 3.

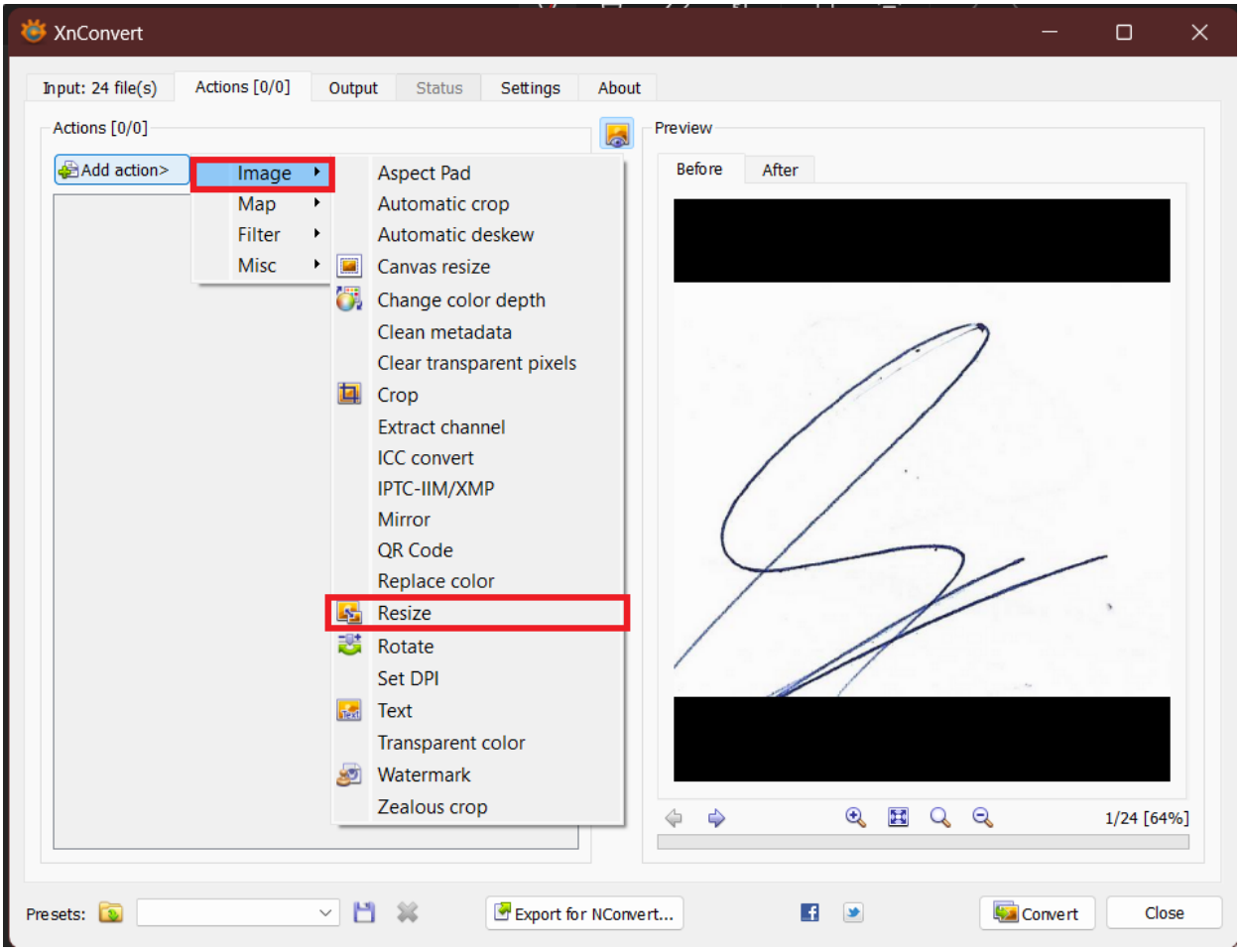

Figure 4

From the menu in Figure 4, we first click on 'Image' and then 'Resize'.

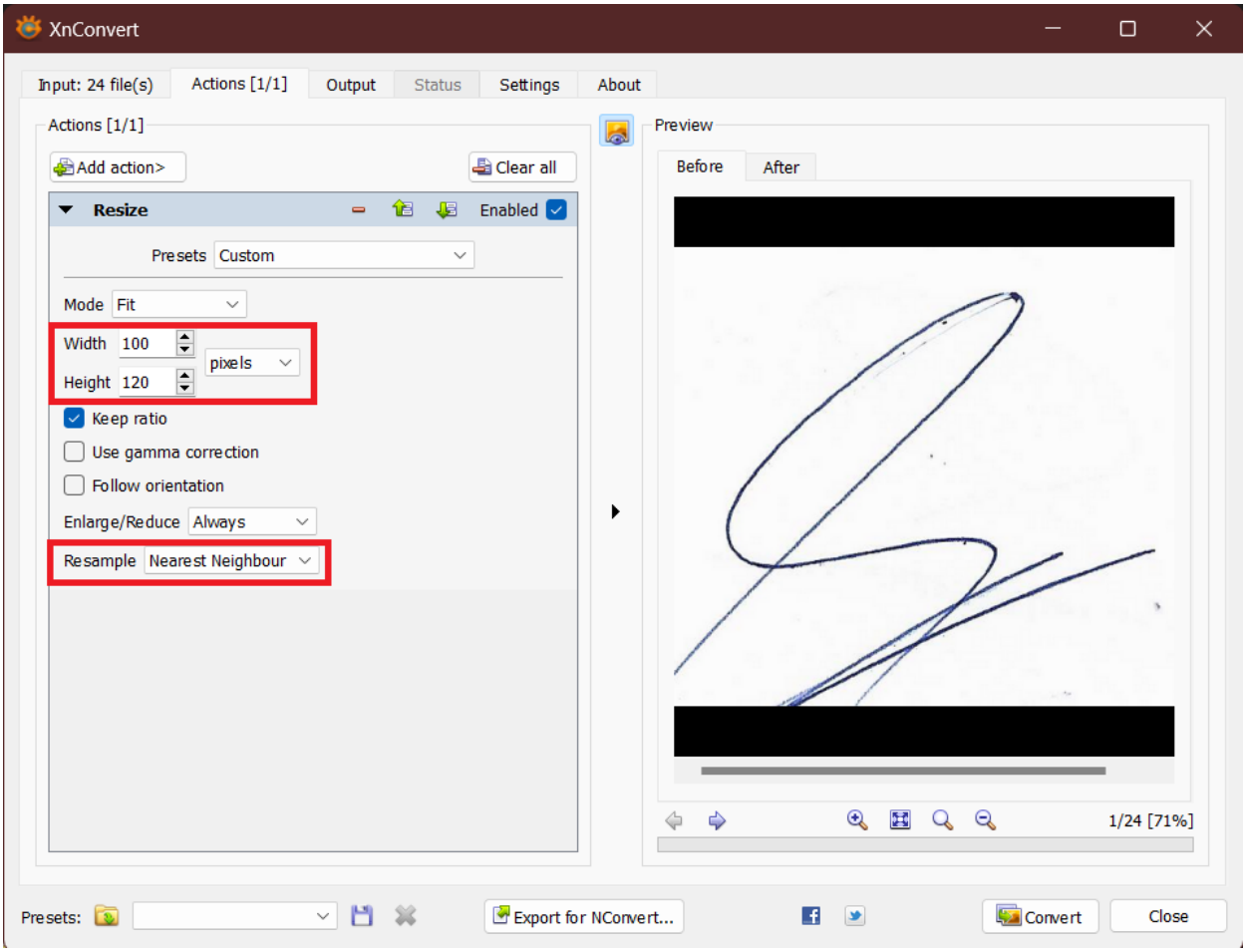

Figure 5

In the opened window, we enter the width and height values and choose the resample mode we want to use.

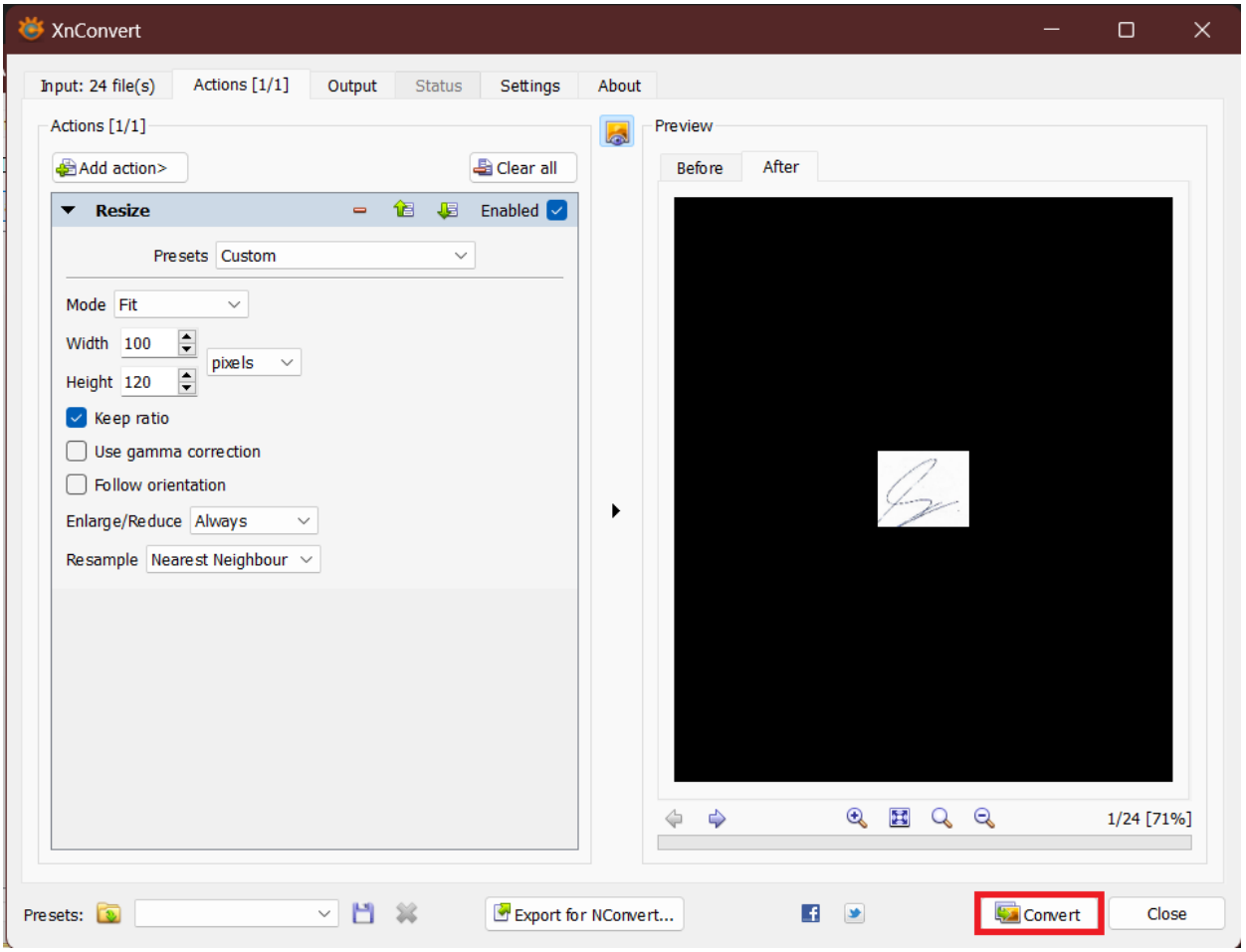

Figure 6

We click on the 'Convert' button shown in Figure 6 and complete the conversion.

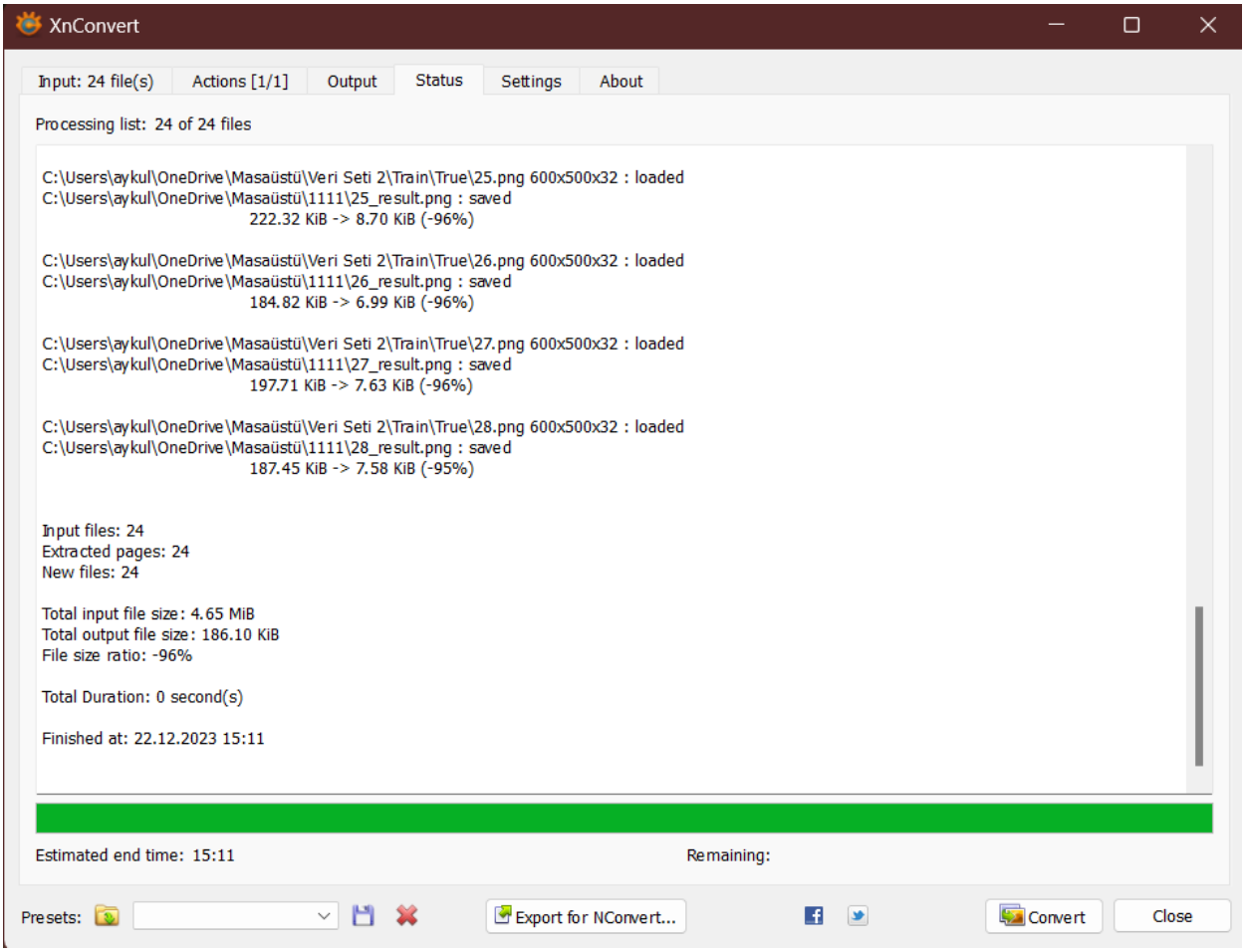

Figure 7

6. CODES

In addition to the tools used, the learning process has been implemented using the Python programming language.

```
import os
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.image import imread
```

```
my_data_dir = 'C:\\Users\\aykul\\OneDrive\\Masaüstü\\Veri Seti'
```

```
os.listdir(my_data_dir)
```

```
['Test', 'Train']
```

```
test_path = my_data_dir+'\\Test'
train_path = my_data_dir+'\\Train'
```

Figure 8

The code block in Figure 8 is used for importing the necessary libraries and assigning the file paths of the 'train' and 'test' folders in the dataset to variables.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
image_gen = ImageDataGenerator(rotation_range=20, # rotate the image 20 degrees
                               width_shift_range=0.10, # Shift the pic width
                               height_shift_range=0.10, # Shift the pic height
                               shear_range=0.1, # Shear means cutting away part of the image
                               zoom_range=0.1, # Zoom in by 10% max
                               horizontal_flip=True, # Allo horizontal flipping
                               fill_mode='nearest' # Fill in missing pixels with the nearest filled value
                              )
```

Figure 9

The code block in Figure 9 creates an ImageDataGenerator object that can be used to augment images. This is employed to increase the diversity of our training data and enhance the performance of our model.

```
image_gen.flow_from_directory(train_path)
```

```
Found 480 images belonging to 2 classes.

<keras.preprocessing.image.DirectoryIterator at 0x2379f613a90>
```

```
image_gen.flow_from_directory(test_path)
```

```
Found 120 images belonging to 2 classes.

<keras.preprocessing.image.DirectoryIterator at 0x2379f6136d0>
```

Figure 10

In Figure 10, the number of classes and the number of data points for training and testing are presented.

```
model = Sequential()

# It is a convolutional neural network (CNN) layer used for extracting features from images.
model.add(Conv2D(filters=32, kernel_size=(3,3),input_shape=image_shape, activation='relu'))
# The Conv2D layer reduces the size of its output while preserving important features.
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(filters=64, kernel_size=(3,3),input_shape=image_shape, activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(filters=128, kernel_size=(3,3),input_shape=image_shape, activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# It combines input data into a vector. This is used with fully connected layers such as Dense layers.
model.add(Flatten())

# We used this layer to classify the data
model.add(Dense(128))
model.add(Activation('relu'))


# Dropouts help reduce overfitting by randomly turning neurons off during training.
# Here we say randomly turn off 50% of neurons.
model.add(Dropout(0.5))

# Last layer, remember its binary so we use sigmoid
model.add(Dense(1))
model.add(Activation('sigmoid'))


model.compile(loss='binary_crossentropy',
              optimizer='Adam',
              metrics=['accuracy'])
```

Figure 11

It is the code in Figure 11 defines and configures a CNN model suitable for binary image

classification tasks. It extracts features from images using convolutional layers, reduces dimensionality using pooling layers, and applies fully connected layers for classification. Dropout is used to prevent overfitting, and the sigmoid activation function ensures the output is a probability value.

```
batch_size = 10
```

```
train_image_gen =image_gen.flow_from_directory(train_path,
                                               target_size=image_shape[:2],
                                               color_mode='rgb',
                                               batch_size=batch_size,
                                               class_mode='binary')
```

```
Found 480 images belonging to 2 classes.
```

```
test_image_gen = image_gen.flow_from_directory(test_path,
                                               target_size=image_shape[:2],
                                               color_mode='rgb',
                                               batch_size=batch_size,
                                               class_mode='binary')
```

```
Found 120 images belonging to 2 classes.
```

```
train_image_gen.class_indices
```

```
{'False': 0, 'True': 1}
```

Figure 12

The code block in Figure 12 creates a data generator using the 'image_gen.flow_from_directory()' function. This generator has been used to load and prepare training data for the model.

```
results = model.fit_generator(train_image_gen,epochs=200,
                              validation_data=test_image_gen
                             )
```

Figure 13

The code in Figure 13 uses the model.fit_generator() function with the training data generated by train_image_gen to train the model. This function feeds the training data to the model in batches to efficiently train the model, evaluates its performance on validation data at specified intervals, and adjusts the weights accordingly.

```
from tensorflow.keras.models import load_model
model.save('Ysa_Proje_200epoch.h5')
```

```
losses = pd.DataFrame(model.history.history)
```

```
losses.head()
```

...

```
losses[['accuracy','val_accuracy']].plot()
```

...

```
losses[['loss','val_loss']].plot()
```

...

```
model.evaluate_generator(test_image_gen)
```

...

```
pred_probabilities = model.predict_generator(test_image_gen)
```

```
test_image_gen.classes
```

...

```
predictions = pred_probabilities > 0.5
```

```
from sklearn.metrics import classification_report,confusion_matrix
```

```
print(classification_report(test_image_gen.classes,predictions))
```

...

```
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
cm=confusion_matrix(test_image_gen.classes,predictions)
disp=ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.show()
```

Figure 14

The code in Figure 14 has been used to visualize the obtained results.

7. RESULTS

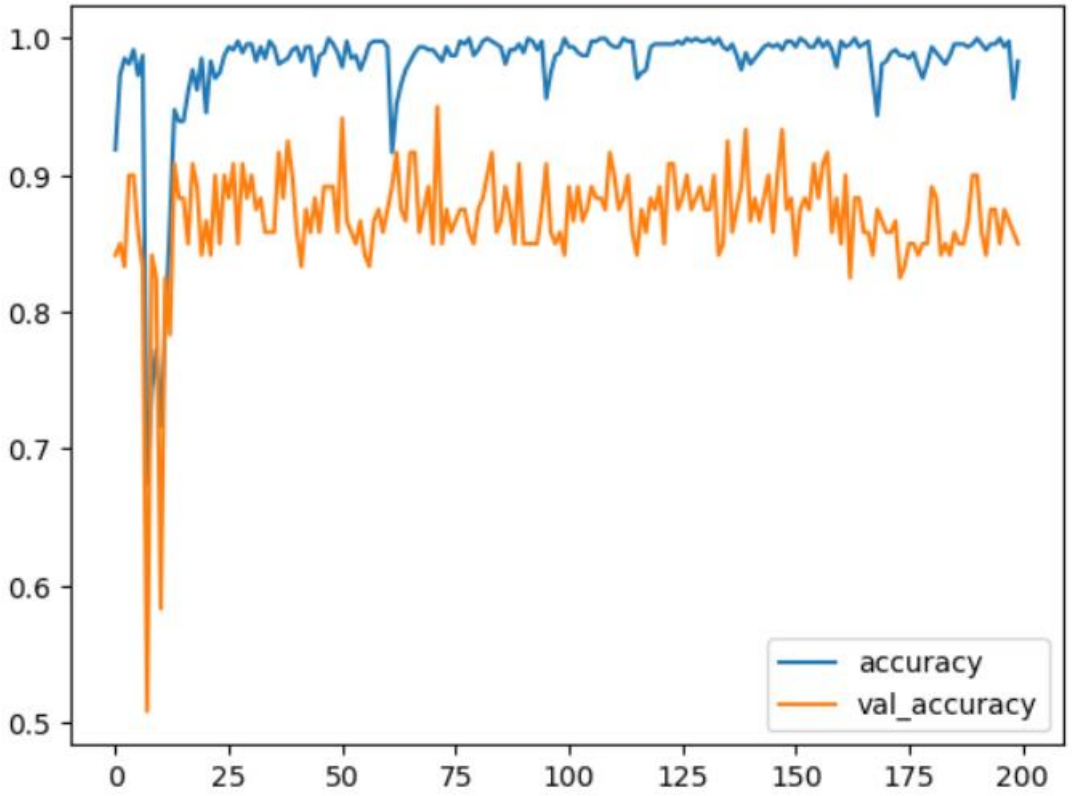

Figure 16

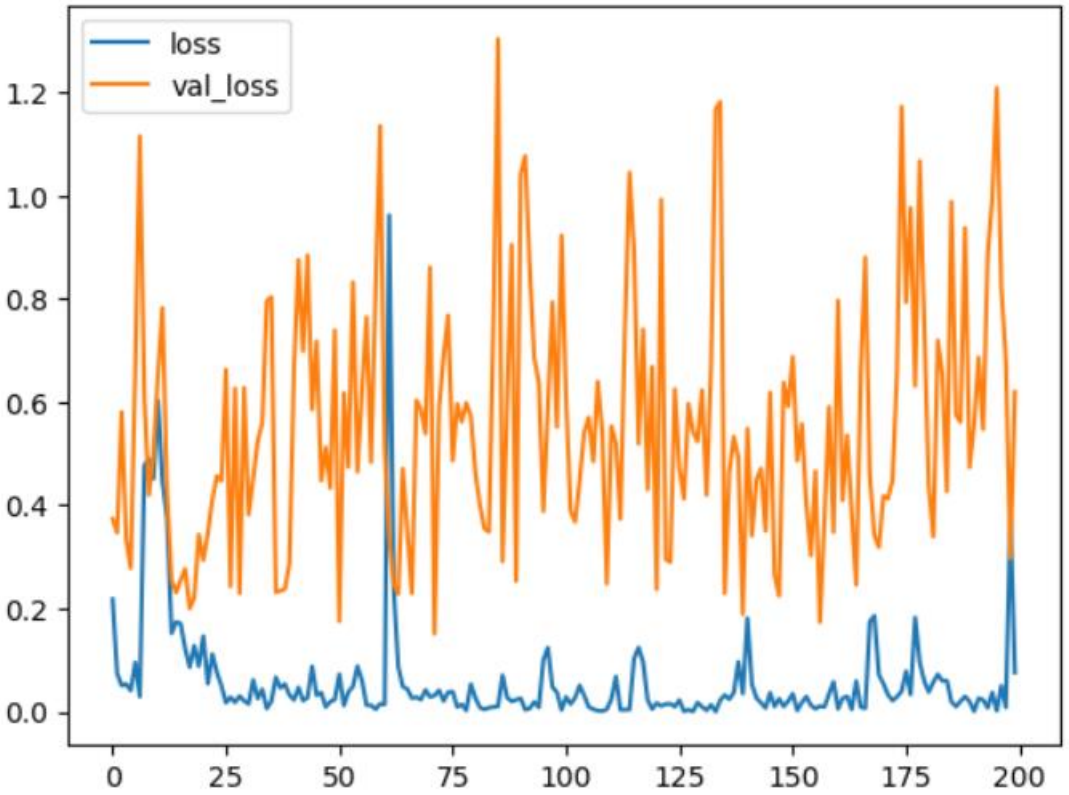

Figure 17

	loss	accuracy	val_loss	val_accuracy
0	0.218165	0.918750	0.373907	0.841667
1	0.074066	0.972917	0.346745	0.850000
2	0.050971	0.985417	0.580827	0.833333
3	0.053692	0.981250	0.334847	0.900000
4	0.041274	0.991667	0.278318	0.900000

Figure 18

	precision	recall	fl-score	support
0	0.49	0.38	0.43	60
1	0.49	0.60	0.54	60
accuracy			0.49	120
macro avg	0.49	0.49	0.49	120
weighted avg	0.49	0.49	0.49	120

Figure 19

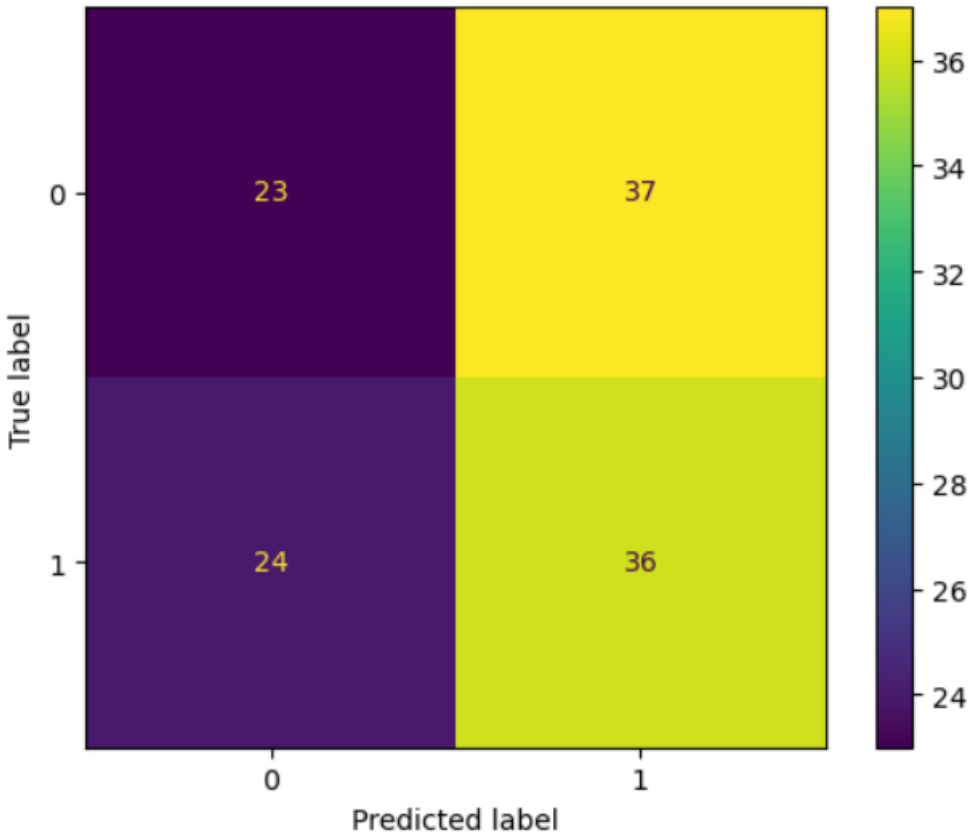

Figure 20

The results obtained from our trained model are presented in Figure 16, Figure 17, Figure 18, Figure 19, and Figure 20. According to these results, the model training did not reach the targeted level.

8. DISCUSSION AND CONCLUSION

While working on this project, we had signatures from 420 individuals, each with 30 different samples. However, our project prioritized the number of signatures from the same individual rather than the total number of individuals. In this case, the number of signatures from the same individual was not sufficient to train the model to the desired level. To address this, we created a new dataset by duplicating signatures from the same individual. Before using these duplicated signatures for model training, we passed them through a function that modified the signatures in specific ways before sending them to the model. This helped increase the count of correct signatures in our dataset. However, it was not sufficient for the model to generalize well.

As a result, we decided to make changes to the model itself. These changes involved adjusting the number of layers in our model, the optimizer used, and the activation functions applied. Through these experiments, the model started to provide more stable results compared to the previous version. Nevertheless, when making decisions for a single signature, the model still does not achieve the desired level of accuracy. To achieve better results, during the data preprocessing stage, we could have applied different filters to the images to make the model work better.

These modifications and adjustments were crucial to improving the performance and reliability of our signature recognition model.

REFERENCES

K. Jain ve diğerleri, "Machine Learning for Signature Verification," 2021

Dinçer, M., "Yapay Zekâ ile İmza Doğrulama,"2022.

Öztürk ve diğerleri, "Yapay Sinir Ağları ile Sahte İmza Tespiti," 2023.

S. K. Saha ve diğerleri, "Signature Verification Using a Siamese Time Delay Neural Network," 2020.

R. Plamondon ve diğerleri, "Deep Learning for Signature Verification: A Comparative Review," 2019.

S. K. Saha ve diğerleri, "Forgery Detection and Verification of Signatures: A Review," 2018.

R. Plamondon ve diğerleri, "Neural Networks for Of line Handwritten Signature Verification: A Review," 2016.

Database : *www.kaggle.com/datasets/jafarsleman/of line-handwriting-signature*

	YES / NO
Did you prepare your study both using a tool and writing code?	*YES*
Did you prepare your report as mentioned in the template?	*YES*
Did you add the results (print screen) of your study to the report?	*YES*
Did you rename your report file as asked in the template?	*YES*
Are you uploading the report to the system?	*YES*
Are you uploading the codes to the system?	*YES*