A Comparative Study on Handwritten Digit Recognition using Supervised Learning and Deep Learning with The Application based on Website Application

Fadly Haikal Fasya
Computer Science Department
(School of Computer Science)
Bina Nusantara University
Jakarta, Indonesia
fadly.fasya@binus.ac.id

Gerry William Nanlohy
Computer Science Department
(School of Computer Science)
Bina Nusantara University
Jakarta, Indonesia
gerry.nanlohy@binus.ac.id

***Abstract* — Handwritten digit recognition technology refers to the automatic identification of handwritten numbers through computers or other equipment, and it has a greater application prospect in letter postal identification and financial statements and bank bill processing. This paper proposes a method of handwritten digit recognition using Supervised Learning Algorithm are KNN Algorithm (K-Nearest Neighbors Algorithm) such as KNN with OpenCV Library and KNN with Scikit-Learn library, Random Forest Classification Algorithm and SVM Algorithm (Support Vector Machine Algorithm) and with Deep Learning Algorithm, CNN (Convolutional Neural Network) with TensorFlow. The aim of this paper is to analyze efficiency of all method which will be experimented and compared to find out the best techniques. We experimented on MNIST and our handmade handwritten digit as dataset. This paper has results where the SVM (Support Vector Machine Algorithm) algorithm is the best algorithm that can perform digit recognition accurately when compared to other algorithms. Where in the first trial when using the MNIST Dataset an accuracy of 86.59% was obtained and when using a handmade digit dataset an accuracy of 73.58%. Then in the second trial an accuracy of 88.09% was obtained from the handmade digit dataset.**

Keywords — Handwritten - Digit - Recognition - Machine Learning, Supervised Learning, KNN Algorithm (K-Nearest Neighbors Algorithm), Random Forest Classification, SVM Algorithm (Support Vector Machine Algorithm), Deep Learning - CNN (Convolutional Neural Network) - TensorFlow.

I. INTRODUCTION

Making machines being capable to perform pattern recognition has always been an important direction of computer research. Character recognition is a typical pattern recognition, through which handwritten numeral recognition is derived. Handwritten digit recognition is one of the oldest and a very important topic in the field of pattern recognition. Handwritten digit recognition poses different problem because of different writing styles, differences in structure and angle of orientation. Therefore, it is very important to find effective method for recognition and classification of digit. Handwritten digit recognition has various applications such as number plate recognition, extracting business card information, bank check processing, postal address processing, passport processing, signature processing etc[1].

Handwriting is a common practice globally, and hence, it is important to select a sample from library for systematic training and testing. At present, a relatively large number of handwritten digital samples are available in the following libraries: (1) the MNIST database, compiled collected by the national bureau of standards and technology of the United States; (2) the CEDAR database, a zip code sample library prepared by the department of computer science, state university of New York, Buffalo; (3) the ETL database, compiled by Japanese institute of electrical technology; (4) the ITPT database, compiled by Japan post and telecommunications policy research institute[2]. Various algorithms have been developed for classification and recognition of handwritten digits and letter. Recognition of handwritten digits and letter has become harder due to different writing styles, angle of orientation and thickness of the digit and letter. Therefore, there is a need to develop algorithm that provides very high recognition accuracy and a very high speed of computation.

Moreover, the time complexity of traditional algorithm or model becomes very high due to the continuous enrichment of handwritten digital sample library, and the gradual improvement of recognition accuracy[3]. For example, the KNN handwritten numeral classification algorithm has been widely used. However, the traditional model calculation is quite laborious for calculating the similarity of sample data and training set data and searching for the first K values. At present, machine learning is also facing similar problems of large training data sets and computing bottlenecks[4]. In view of the above problems, this paper proposes a method of handwritten digit recognition using Supervised Learning Algorithm are KNN Algorithm (K-Nearest Neighbors Algorithm) such as KNN with OpenCV Library and KNN with Scikit-Learn library, Random Forest Classification and SVM Algorithm (Support Vector Machine Algorithm) and with Deep Learning Algorithm, CNN (Convolutional Neural Network) with TensorFlow.

The process of handwritten digit recognition is divided into 4 phases. These phases are Data Collection, Pre-processing, Feature Extraction and Classification. Data collection is collecting the required dataset for the experiment. Preprocessing deals with normalization of dataset, binarization and noise removal. In feature extraction, ideal number of features are extracted using various methods. These features are passed to the classifier for classification.

II. Related Works

1. Handwritten Digit Recognition Using K-Nearest Neighbour Classifier (U. R. Babu, Y. Venkateswarlu and A. K. Chintha, 2014).

   This study presents a new approach to off-line handwritten digit recognition based on structural features which is not required thinning operation and size normalization technique. The main objective of this paper is to provide efficient and reliable techniques for recognition of handwritten digits. A Euclidean minimum distance criterion is used to find minimum distances and k-nearest neighbor classifier is used to classify the digits. A MNIST database is used for both training and testing the system. 5000 images are used to test the proposed method a total 5000 numeral images are tested and got 96.94% recognition rate.

2. Comparisons on KNN, SVM, BP and the CNN for Handwritten Digit Recognition (W. Liu, J. Wei, and Q. Meng, 2020).

   This study takes the MNIST handwritten digit database as samples, discusses algorithms KNN, SVM, BP neural network, CNN, and their application in handwritten digit recognition. In the training process, this work rewrites KNN with Python, SVM with scikit-learn library, and BP, CNN with TensorFlow, and fine-tunes the algorithm parameters to get the best results for each algorithm. Finally, by comparing the recognition rate and recognition duration of the four algorithms, the advantages, and disadvantages of the four algorithms in handwriting recognition are analyzed.

3. Handwritten Digit and Letter Recognition Using Hybrid DWT-DCT with KNN and SVM Classifier. (P. Ghadekar, S. Ingole and D. Sonone, 2018).

   This study proposes a method of handwritten digit and letter recognition using feature extraction based on hybrid Discrete Wavelet Transform (DWT) and Discrete Cosine Transform (DCT). These extracted features are passed to K-Nearest Neighbor (KNN) and Support Vector Machine (SVM) classifiers for classification. Standard MNIST and EMNIST letter dataset are used for this experiment. Firstly, MNIST digit and EMNSIT letter dataset are binarized and later stray pixels are removed. Features are extracted using hybrid Discrete Wavelet Transform and Discrete Cosine Transform. KNN and SVM classifiers are used for classification purpose. The proposed method was able to obtain a highest accuracy of 97.74% for digit and 89.51% for letter using SVM classifier.

4. Analogizing time complexity of KNN and CNN in recognizing handwritten digits (T. Makkar, Y. Kumar, A. K. Dubey, Á. Rocha and A. Goyal, 2017).

   The study analyzes the time complexity in two different algorithms KNN and CNN. The K-Nearest Neighbor Algorithm is used as a classifier capable of computing the Euclidean distance between data set input images. The dataset is fetched for training through neural network contains various (28 × 28) pixel size images and therefore, our first layer of neural network contains 784 neurons as input. We will analyze these images by varying values to obtain output layer of our network 10 neurons, each neuron if fixed gives any output between 0 to 9. After reading in data appropriately from MNIST and testing it on Gaussian distribution, KNN classifier then presented the result with Python tool. So, to avoid such a long waiting another algorithm Convolutional Neural Networks has been used. CNN has been implemented on Keras including TensorFlow and produces accuracy. It is then shown that KNN and CNN perform competitively with their respective algorithm on this dataset, while CNN produces high accuracy than KNN and hence chosen as a better approach.

III. Tools, Framework, And Method

To find effective method for recognition and classification of digit, the author uses a machine learning and deep learning method including KNN Algorithm (K-Nearest Neighbors Algorithm) such as KNN with OpenCV Library and KNN with Scikit-Learn library, Random Forest Classification, SVM Algorithm (Support Vector Machine Algorithm) and CNN (Convolutional Neural Network) with TensorFlow. The author also creates a web interface so that users can see how the product works. To accomplish this, the author employs the following tools and algorithms are used to develop this product:

A. Python

Python is one of many programming languages that are widely used by programmers. The popularity of this programming language is not without reason, python is known as a programming language that is very close to human language (high-level programming language), making it easier for developers and even average people to understand. Furthermore, Python is well-known as one of the languages widely used in the application of machine learning, data analysis, and deep learning.

The author also uses the python programming language in this study because the library required to make stock predictions with random forest regression has already been provided by python. The following libraries were used in this study:

1. Streamlit

Streamlit is an open-source Python library that makes it easy to create and share beautiful, custom web apps for machine learning and data science using the functions it provides[5]. This library is intended specifically for machine learning and data science purposes, as most machine learning and data science libraries can be used in streamlit to quickly build, deploy, or host responsive web sites[5].

2. PyNgrok

PyNgrok is a Python wrapper for ngrok that manages its own binary, making ngrok available via a convenient Python API. Ngrok is a reverse proxy tool that opens secure tunnels from public URLs to localhost, perfect for exposing local web servers, building webhook integrations, enabling SSH access, testing chatbots, demoing from your own machine, and more, and it's made even more powerful with native Python integration through PyNgrok[6].

3. UUID

UUID (Universal Unique Identifier) is a module that provides immutable UUID objects (the UUID class) and the functions uuid1(), uuid3(), uuid4(), uuid5() for generating version 1, 3, 4, and 5 UUIDs as specified in RFC 4122 [https://docs.python.org/3/library/uuid.html]. UUID helps in generating random objects of 128 bits as ids. Advantages of UUID can be used as general utility to generate unique random id, can be used in cryptography and hashing applications and useful in generating random documents, addresses etc[7].

4. Localtunnel

Localtunnel allows you to easily share a web service on your local development machine without messing with DNS and firewall settings[8]. Localtunnel will assign you a unique publicly accessible url that will proxy all requests to your locally running webserver. Localtunnel is great for working with browser testing tools like browserling or external api callback services like twilio which require a public URL for callbacks[9].

5. Matplotlib

Matplotlib is a Python-based data visualization library[10] as a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible. Matplotlib, unlike Seaborn, can only perform basic data visualization. Matplotlib can create publication quality plots; make interactive Figures that can zoom, pan, update; customize visual style and layout; export to many file formats; embed in JupyterLab and Graphical User Interfaces; use a rich array of third-party packages built on Matplotlib, etc[11].

6. IO

IO is a module provides Python's main facilities for dealing with various types of I/O and allow us to manage the file related input and output operations. There are three main types of I/O: text I/O, binary I/O and raw I/O. These are generic categories, and various backing stores can be used for each of them. A concrete object belonging to any of these categories is called a file object. Other common terms are stream and file-like object. Independent of its category, each concrete stream object will also have various capabilities: it can be read-only, write-only, or read-write[12].

7. Scikit-learn

Scikit-learn is a Python library for making predictions based on data analysis. This library was created by combining the NumPy, SciPy, and matplotlib libraries[13]. Furthermore, scikit-learn includes a wide variety of libraries related to machine learning algorithms for both supervised and unsupervised learning[14].

8. NumPy

NumPy is a Python library that is widely used for scientific computing and offers many functions related to arrays, mathematics, and logic[15]. A NumPy array is a numerical representation of data that can be used to perform numerical computations effectively and efficiently[16].

9. OpenCV

OpenCV (Open-Source Computer Vision Library) is an open-source library that includes several hundreds of computer vision algorithms. OpenCV has a modular structure, which means that the package includes several shared or static libraries. The following modules are available:

- Core functionality (core) - a compact module defining basic data structures, including the dense multi-dimensional array Mat and basic functions used by all other modules.

- Image Processing (imgproc) - an image processing module that includes linear and non-linear image filtering, geometrical image transformations (resize, affine and perspective warping, generic table-based remapping), color space conversion, histograms, and so on.

- Video Analysis (video) - a video analysis module that includes motion estimation, background subtraction, and object tracking algorithms.

- Camera Calibration and 3D Reconstruction (calib3d) - basic multiple-view geometry algorithms, single and stereo camera calibration, object pose estimation, stereo correspondence algorithms, and elements of 3D reconstruction.

- 2D Features Framework (features2d) - salient feature detectors, descriptors, and descriptor matchers.

- Object Detection (objdetect) - detection of objects and instances of the predefined classes (for example, faces, eyes, mugs, people, cars, and so on).

- High-level GUI (highgui) - an easy-to-use interface to simple UI capabilities.

- Video I/O (videoio) - an easy-to-use interface to video capturing and video codecs.

- And some other helper modules, such as FLANN and Google test wrappers, Python bindings, and others[17].

10. Pillow

Pillow is the Python Imaging Library adds image processing capabilities to your Python interpreter. This library provides extensive file format support, an efficient internal representation, and powerful image processing capabilities. The core image library is designed for fast access to data stored in a few basic pixel formats. It should provide a solid foundation for a general image processing tool[18].

11. TensorFlow

TensorFlow is an open-source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The flexible architecture allows you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API. The TensorFlow API is composed of a set of Python modules that enable constructing and executing TensorFlow graphs. The tensorflow package for R provides access to the complete TensorFlow API from within R[19].

12. Keras

Keras is a high-level neural networks library, written in Python and capable of running on top of either TensorFlow or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research. Use Keras if you need a deep learning library that:

- Allows for easy and fast prototyping (through total modularity, minimalism, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Supports arbitrary connectivity schemes (including multi-input and multi-output training).
- Runs seamlessly on CPU and GPU[20].

B. Visual Studio Code

Visual Studio Code is a code editor that supports a variety of programming languages, including html, CSS, and JavaScript. Furthermore, this tool is compatible with all PC operating systems, including Windows, macOS, and Linux. To use Python in Visual Studio Code, users must first download the python extension, which is one of the most popular extensions in the program[21]. Before starting to code the program itself, users must first create their own folder or workspace in Visual Studio Code[22]. The authors of this study use these tools to accommodate projects and build drawable digit recognition programs in combination with all the methods. When using Visual Studio Code to build projects, the author achieves many advantages, including the ability to connect directly to GitHub, countless extensions that make it easier for writers to write programs, a cleaner project workspace, and much more.

C. Jupyter Notebook

Jupyter Notebook is a web-based notebook application that is mainly used to compute a language. Jupyter notebook supports over 40 different programming languages, including Python. This notebook is also commonly used for big data computing[23]. This is the main reason that drives the authors to use these tools for data processing tasks like exploratory data analysis, data cleaning, data transformation, and even the development of their own machine learning models. The.ipynb file (python notebook) that can be downloaded when using this notebook can provide a more interactive report because it can provide a more interesting explanation than using ordinary comments[24].

D. GitHub

GitHub is a hosting service that allows you to manage software versions. Furthermore, GitHub is widely used due to its collaboration features, which enable many developers to collaborate in a GitHub environment with many other supporting features wherever and whenever they are[25]. The author uses this tool because the author wants all of the project members to collaborate in one environment so that project processing development takes less time and is more efficient.

E. Google Colab

Google Colab or Colab notebooks allow you to combine executable code and rich text in a single document, along with images, HTML, LaTeX and is especially well suited to machine learning, data analysis and education. When you create your own Colab notebooks, they are stored in your Google Drive account. You can easily share your Colab notebooks with co-workers or friends, allowing them to comment on your notebooks or even edit them[26]. The author uses this tool because the author wants all of the project members can looking to back their work up to the cloud and to sync their notebooks across multiple devices so that project processing development takes less time and is more efficient[24].

F. Random Forest Classification Algorithm

Random Forest is a machine learning algorithm that utilizes averaging to improve accuracy and resolve overfitting, and it also uses several decision trees for each sample, which is divided into sub-samples[27].

Because the author's dataset contains thousands of rows of data, the author requires a machine learning algorithm that can manage very large amounts of data. The Random Forest algorithm is well suited to overcoming these issues because it has been shown to provide better accuracy results when dealing with large datasets than traditional decision tree algorithms[28].

The author uses this algorithm to classify certain diseases based on their symptoms. The algorithm will divide the dataset into sub-samples in several trees randomly[28], depending on the number of estimators and max depths entered by the user. Then a vote will be taken for each class, which will later be combined to find the highest vote. The highest vote will be the final prediction.

G. K-Nearest Neighbor Algorithm

K-Nearest Neighbor, or KNN for short, is a machine learning algorithm that is commonly used for classification, but it may also be used for regression cases. It works by providing a training dataset with data points grouped into a specific class, so that data testing can predict that class[29]. It works by determining the number of neighbors to be used for classification. The distance from each data point in the dataset is then calculated. Finally, to determine the class prediction, determine K from data that has not seen during training[30].

This algorithm, like Random Forest, is used by the author to classify diseases based on their symptoms. Furthermore, the authors include this algorithm to compare with random forest to determine which classification algorithms produce the best accuracy.

H. Support Vector Machine Algorithm

In machine learning, support vector machines are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other[31]. The advantages of support vector machines are:

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

The disadvantages of support vector machines include:

- If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial.
- SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation (see Scores and probabilities, below)[32].

I. Convolutional Neural Network

A CNN is a kind of network architecture for deep learning algorithms and is specifically used for image recognition and tasks that involve the processing of pixel data. There are other types of neural networks in deep learning, but for identifying and recognizing objects, CNNs are the network architecture of choice[33]. Computer vision is evolving rapidly day-by-day, so CNN is surprisingly effective on computer vision and natural language processing tasks, it is widely used in real world applications[34].

IV. Result And Discussion

A. Data Collection

Data collection is the process of gathering, measuring, and analyzing accurate data from a variety of relevant sources to find answers to research problems, answer questions, evaluate outcomes, and forecast trends and probabilities. In this paper, the author used MNIST and Handmade Handwritten Digit dataset for training and testing in digit recognition.

1. MNIST Dataset

MNIST dataset is used for digit recognition in this experiment. MNIST dataset is a large database of handwritten digits. It is a subset of a larger set available from NIST. This dataset is publicly available. MNIST dataset has total 70000 images and respective labels. Out of this, 60000 images are used for training and 10000 are used for testing purpose. These images are in greyscale format. The images are size normalized. The digits are centered in image with respect to the background. There are total 784 pixels in each image. These 784 pixels are arranged in 28 x 28 plane. In this experiment, we are using all 60000 images for training and all 10000 images for testing. Figure 1 shows some random images from the MNIST dataset.

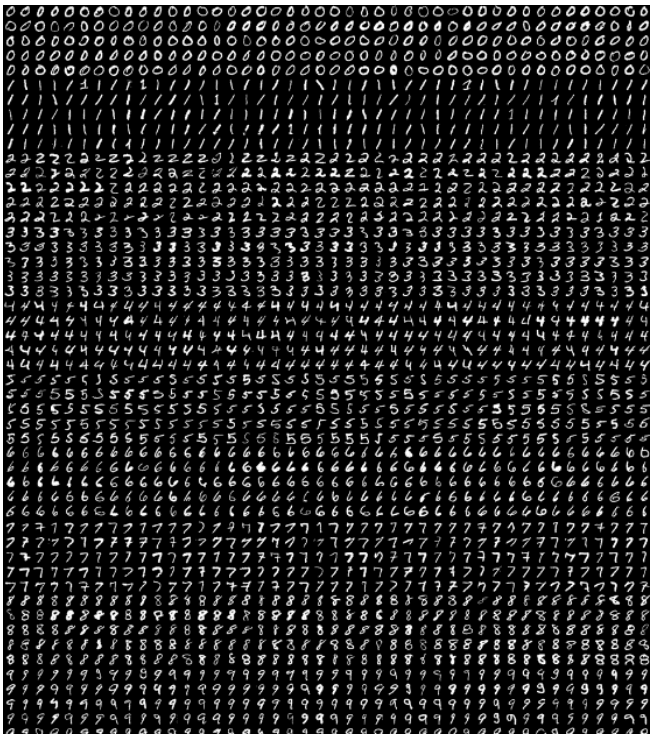

Fig 1. *MNIST dataset sample image*

2. Handmade Handwritten Digit Dataset

The author also created a digits dataset from the author's handwriting, named DigitRecognition_Dataset. This dataset is created using Drawable Digit Recognition which is already in the program by carrying out several stages, as follows:

1. First, type and run the code "*pip install pillow*" as shown in Figure 2, to import the pillow library into the program.

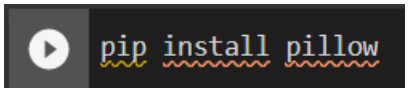

Fig 2. Install Pillow

2. Second, type and run the code "*!pip install streamlit==1.13.0*" to import the streamlit framework and "*!npm install -g localtunnel*" to call the localtunnel module into the program as shown in Figure 3.

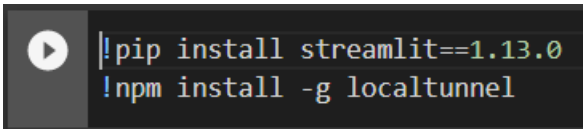

Fig 3. *Install Streamlit & Localtunnel*

3. Third, type and run the code "*!pip install streamlit-drawable-canvas*" to import the streamlit framework which will help us draw on a canvas as shown in Figure 4.

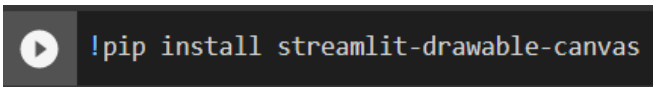

Fig 4. *Streamlit Drawable Canvas*

4. Fourth, run the code in the "*Streamlit App*" column to run the streamlit application framework as shown in Figure 5.

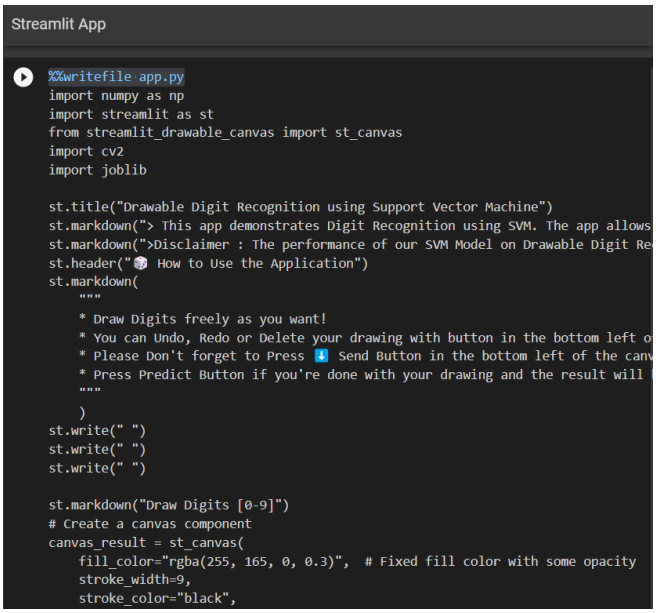

Fig 5. *Streamlit Application*

5. Fifth, run the code in the "*Streamlit Create Dataset*" column to create a dataset through the streamlit framework as shown in Figure 6.

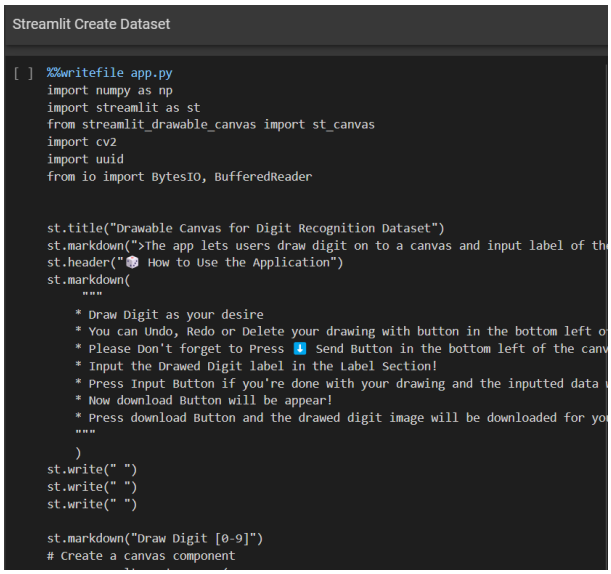

Fig 6. *Streamlit Create Dataset*

6. Finally, type and run the code "*!streamlit run app.py & npx localtunnel --port 8501*" in the "Local Tunnel" column to run the streamlit application on the local web in the browser as shown in Figure 7.

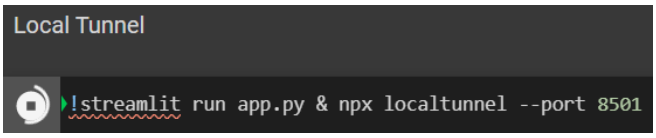

Fig 7. *Localtunnel*

When the code is executed, wait, and make sure the URL appears as shown in Figure 8, which will redirect to a local website in the browser when clicked, as shown in Figure 9.

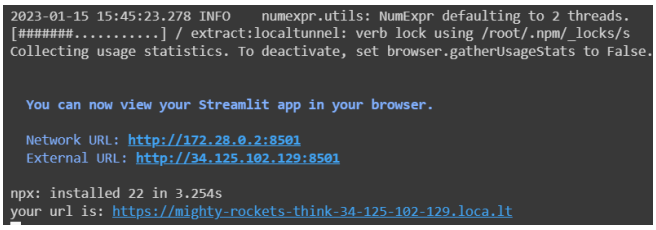

Fig 8. *Localtunnel URL*

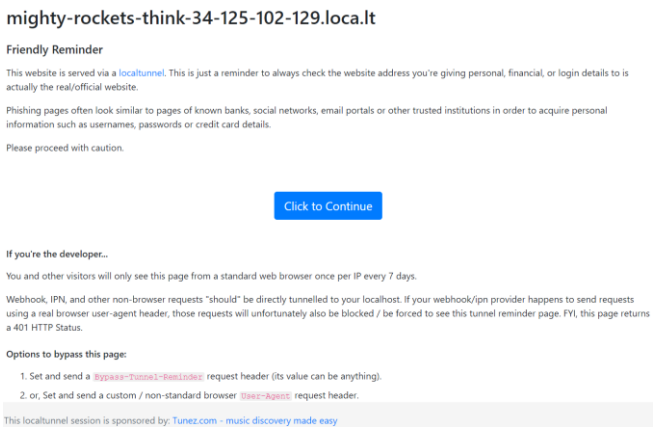

Fig 9. *Localtunnel Local Website*

Then click the *Continue* column located on the local website as shown in Figure 10.

Fig 10. *Continue Column*

Later, you will be redirected to a streamlit application with a drawable canvas, as shown in Figure 11.

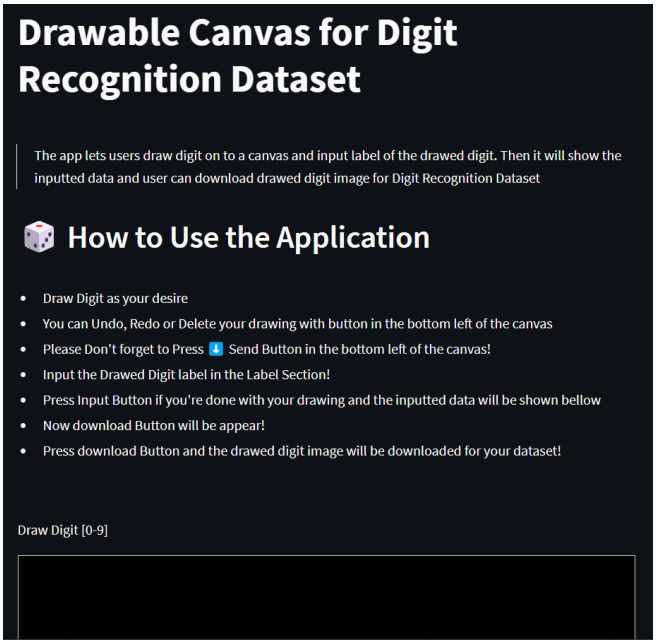

Fig 11. *Drawable Canvas on Streamlit Application*

If all the steps above have been executed but the localtunnel module does not want to connect or the streamlit application only gives a "please wait" display for a long time, then use the PyNgrok module step as a solution, as follows:

1. Repeat the entire steps above from step one to step five. After that, proceed to the next step.

2. The next step is type and run the code "!pip install pyngrok" in the Ngrok column to call the PyNgrok module as shown in Figure 12.

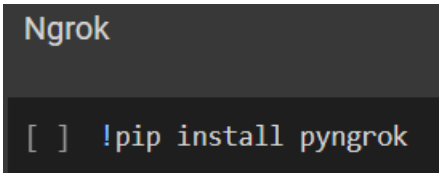

Fig 12. *Install PyNgrok*

3. Then, type and run the code "!streamlit run app.py &>/dev/null&" as shown in Figure 13.

```
!streamlit run app.py &>/dev/null&
```

Fig 13. *Streamlit Run App Developer*

4. Then, type and run the codes “from pyngrok import ngrok” and “!ngrok authtoken 2K5VMsLbfPowtf2cr81cJBK9NSq_5woqSrwvdkEpaCqsU9LdJ” to import the Ngrok library along with token authentication as shown in Figure 14.

```
from pyngrok import ngrok
!ngrok authtoken 2K5VMsLbfPowtf2cr81cJBK9NSq_5woqSrwvdkEpaCqsU9LdJ
```

Fig 14. *Import Ngrok Library & Authtoken*

5. Then, type and run the code “public_url = ngrok.connect(port='8501')” to determine which port will be used by the local website and “public_url” to run the local website on the browser as shown in Figure 15.

```
public_url = ngrok.connect(port='8501')
public_url
```

Fig 15. *Ngrok Public URL*

6. When you have finished using the Streamlit application on the local website, type and run the code "ngrok.kill()" to close the local website and turn off the PyNgrok module and the Ngrok library as shown in Figure 16.

```
# Kill Ngrok After Use
ngrok.kill()
```

Fig 16. *Ngrok Closed*

When you have successfully opened the Streamlit application using these two methods, you can draw any object on the canvas in the black box. For example, because the writer wants to make a dataset in the form of numbers, the writer draws numbers from the range 0 – 9, as shown in Figure 17.

Fig 17. *Draw The Digit*

The author can use the Undo menu to cancel previously created images, as shown in Figure 18, Redo to repeat images that have been canceled previously, as shown in Figure 18. Figure 19 and Reset Canvas & History to delete the entire image on the canvas as shown in Figure 20.

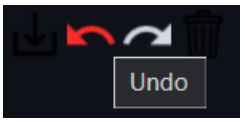

Fig 18. *Undo Menu*

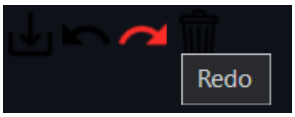

Fig 19. *Redo Menu*

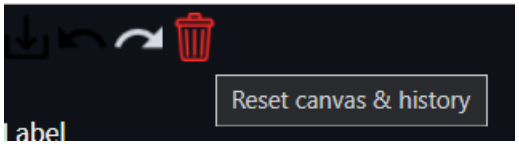

Fig 20. Reset Canvas & History Menu

Then when the author wants to label the numbered image that has been drawn, the author must press the Send to Streamlit menu, as shown in Figure 21, so that later the number image along with the labels can be processed into an image which will appear in the next column so that the author can download the number images that have been made.

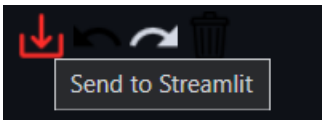

Fig 21. *Send to Streamlit Menu*

Then in the label column the author gives a number label that matches what is drawn on the canvas. Next, the author presses the input button when he feels that the number images labeled are appropriate as shown in Figure 22.

Fig 22. *Input Button for Labelling Digit Image*

Finally, the author saves the number images that have been labeled in the storage location according to the download settings in the browser used in the .JPG format as shown in Figure 23.

Fig 23. *Download Digit Image for Dataset in .JPG format*

B. *Pre-Processing*

The code as shown in Figure 24 starts by creates a new image with cv2.imread() and converts it to grayscale using cv2.cvtColor(). Next, it blurs the image with a Gaussian blur of 5 pixels on each side (5,5). Then finds contours in the blurred image using cv2.findContours(), which returns an array of rectangles that represent where there are objects in the original image. These rectangles are stored in a list called "boundingBoxes". The next step is to create an empty list called "display" that will hold all the images found in boundingBoxes as they're displayed one at a time on screen.

This is done by looping through each contour and creating an empty list for every number found inside its rectangle (x, y, w, h) before appending them to display so that they can be drawn on screen one at a time as well as saved into separate files when finished drawing them all out onto screen or saving them into individual files if desired later on down the line. Next up is finding out how many numbers were found inside any given rectangle's area by comparing its.

```
def recognize_digit(image):
  gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
  blur = cv2.GaussianBlur(gray, (5,5), 0)

  canny = cv2.Canny(blur, 30, 150)

  contours, _ = cv2.findContours(canny.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
  boundingBoxes = [cv2.boundingRect(c) for c in contours]

  display = []
  image_numbers = []

  for contour in contours:
    (x, y, w, h) = cv2.boundingRect(contour)

    if w>= 5 and h>=20:
      area = blur[y:y+h, x:x+w]
      ret, area = cv2.threshold(area, 127, 255, cv2.THRESH_BINARY_INV)

      new_square = drawSquare(area)
      number = resize(new_square, 100)
      result = number.reshape((1, 10000));
      # number = resize(new_square, 20)
      # result = number.reshape((1, 400));
      result = result.astype(np.float32);
      image_numbers.append(number)
```

Fig 24. *Pre-Processing Image*

Then as shown in Figure 25, creates a new model using the cv2.SVC() function, which is an SVM classifier. This model will be used to classify images into one of two categories like "dog" or "cat". The next line uses the predict() method on this newly created model to get predictions for each image that was classified as a dog or cat. These predictions are stored in a list called result and can be accessed with n. The next line calculates how many times each number appears in the list of results and prints them out on screen with display. The code will create a rectangle of the text "123" in the middle of an image.

```
    #knn opencv
    # ret, res, neighbours, distance = model_cv_knn.findNearest(result, k=1)
    # n = str(int(float(res[0]))); print(res.shape)
    # display.append(n)

    #svm
    # res = model_svm_mnist_new.predict(result)
    res = model_svm_manual_new.predict(result)
    n = str(int(float(res)))
    display.append(n)

    #tensor keras
    # result = np.array(result).astype(np.float32)
    # res = model_tensor.predict(result)
    # n = str(int(float(res[0][0])))
    # display.append(n)

    # draw rectangle around individual digit
    cv2.rectangle(image, (x,y), (x+w, y+h), (255,0,0), 1)
    cv2.putText(image, n, (x,y-10), cv2.FONT_ITALIC, 2, (0,255,0), 2)

return image, display, image_numbers
```

Fig 25. *Create Model*

C. *Feature Extraction*

1. MNIST Dataset

The code as shown in Figure 26 is meant to train a neural network with the MNIST dataset. The code starts by reading the digits.png image file and converting it to grayscale using cv2.cvtColor(). Then resizes the gray image to 50x50 pixels using cv2.pyrDown(). Then splits each pixel into a red, green, or blue component with np.hsplit() and reshapes them into an array with np.reshape(-1, 400). It converts these arrays of color values from floating point numbers to integers with astype(np.float32) before feeding them into X_train_mnist and X_test_mnist which are both matrices of size 400x400 that have been initialized as zeros except for one column in each matrix where they contain 350 elements representing the digits 0-9 respectively (the first 70 columns in X_train_mnist represent digits 1-9 while the last 100 columns in X_test_mnist represent digits 0-9). The next step is creating a list of lists called y which has 350 rows representing all possible combinations of 10 different digits (0 through 9) and 150 columns representing all possible combinations of 10 different digits (0 through 9).

```
# Loading the digits data
data = cv2.imread('./drive/MyDrive/DigitRecognition_Dataset/digits.png')
gray = cv2.cvtColor(data, cv2.COLOR_BGR2GRAY)

# Resizing each digit from 20x20 to 10x10
resized = cv2.pyrDown(gray)

# Splitting image original image into 5000 different arrays of size 20x20
# Resulting array: 50 * 100 * 20 * 20
arr = [np.hsplit(i, 100) for i in np.vsplit(gray, 50)]
arr = np.array(arr)

# Spliting into training and test set
# Total: 5000, Train: 3500 images, Test: 1500
X_train_mnist = arr[:, :70].reshape(-1, 400).astype(np.float32)
X_test_mnist = arr[:, 70:100].reshape(-1, 400).astype(np.float32)

## Targets for each image
y = [0,1,2,3,4,5,6,7,8,9]

y_train_mnist = np.repeat(y, 350)[:, np.newaxis]
y_test_mnist = np.repeat(y, 150)[:, np.newaxis]
```

Fig 26. *MNIST Dataset Splitting Data*

2. DigitRecognition_Dataset

The code as shown in Figure 27 is meant to split the data into two sets, one for training and one for testing. Starts by getting the path to the train dataset. Then it gets a list of all the names in that folder and creates a new variable called "train_image_list" which is just an array with images from each class. Next, it splits the data into two sets: one for training and one for testing. It then creates variables for X_train, X_test, y_train, and y_test. These are lists of 25% of the data from each set respectively. The code then trains a model on these two sets using cross-validation (randomly splitting them into training and test).

```
train_root_path = './drive/MyDrive/DigitRecognition_Dataset/Train'
train_names = get_path_list(train_root_path)
train_image_list,
image_classes_list,
image_blur_list = get_class_id(train_root_path, train_names)

(X_train, X_test, y_train, y_test) = train_test_split(train_image_list,
                                                      image_classes_list,
                                                      test_size=0.25,
                                                      random_state=42)
```

Fig 27. *DigitRecognition_Dataset Splitting Data*

D. Classification

1. KNN with OpenCV Algorithm

Create an image classifier using the KNN algorithm by calling the cv2 library from OpenCV. It then trains the classifier on the training data using two different datasets using ROW_SAMPLE and prints the results of the training data and tests in the variables: response, result, neighbor, distance to calculate the accuracy after each iteration to see how it improves over time. When using the MNIST Dataset the accuracy results are 93.46% as shown in Figure 28, while when using the DigitRecognition_Dataset the accuracy results are 77.35% as shown in Figure 29.

```
# Using K-NN(k- nearest neighbors) as the ML algorithm
classifier_knn = cv2.ml.KNearest_create()
classifier_knn.train(X_train_mnist, cv2.ml.ROW_SAMPLE, y_train_mnist)
response, result, neighbours, distance = classifier_knn.findNearest(X_test_mnist, k=3)

# Testing and calculating the accuracy of knn classifier
correct = result == y_test_mnist
correct = np.count_nonzero(correct)
accuracy = correct * (100.0/result.size)
print ("MNIST Dataset : ", accuracy)
```

```
MNIST Dataset :  93.46666666666667
```

Fig 28. *KNN with OpenCV Algorithm using MNIST Dataset*

```
# Using K-NN(k- nearest neighbors) as the ML algorithm
model_cv_knn = cv2.ml.KNearest_create()
model_cv_knn.train(X_train, cv2.ml.ROW_SAMPLE, y_train)
response, result, neighbours, distance = model_cv_knn.findNearest(X_test, k=3)

# Testing and calculating the accuracy of knn classifier
res_flatten = result.flatten()
correct = res_flatten == y_test
correct = np.count_nonzero(correct)
accuracy = correct * (100/res_flatten.size)
print ("Manual Dataset : ", accuracy)
```

```
Manual Dataset :  77.35849056603773
```

Fig 29. *KNN with OpenCV Algorithm using DigitRecognition_Dataset*

2. KNN with Scikit-Learn Algorithm

Create an image classifier using the KNN with Scikit-Learn Algorithm by importing the required libraries. Then, creating a KNeighborsClassifier object with a neighbor limit of 5, algorithm set to 'auto' and job count set to 10. This class' fit method is used to train the model on the X_train_mnist and y_train_mnist training datasets. The model_knn_sklearn object will then correspond to the X_train_mnist and y_train_mnist training data. Once matched, the score function returns a confidence value that can be used to evaluate how confident we are in our predictions for the X_test_mnist test data. Finally, predictions for the X_test_mnist test data are made by calling predict on the model. When using the MNIST Dataset the accuracy results are 93.13% as shown in Figure 30, while when using the DigitRecognition_Dataset the accuracy results are 78.30% as shown in Figure 31.

```
model_knn_sklearn = KNeighborsClassifier(n_neighbors=5,algorithm='auto',n_jobs=10)
model_knn_sklearn.fit(X_train_mnist,y_train_mnist)
confidence = model_knn_sklearn.score(X_test_mnist,y_test_mnist)
y_pred_mnist = model_knn_sklearn.predict(X_test_mnist)
accuracy = accuracy_score(y_test_mnist, y_pred_mnist)
print ("MNIST Dataset : ", accuracy*100)
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/neighbors/_classification.py:198: Da
  return self._fit(X, y)
MNIST Dataset :  93.13333333333334
```

Fig 30. *KNN with Scikit-Learn Algorithm using MNIST Dataset*

```
model_knn_sklearn = KNeighborsClassifier(n_neighbors=5,algorithm='auto',n_jobs=10)
model_knn_sklearn.fit(X_train,y_train)
confidence = model_knn_sklearn.score(X_test,y_test)
y_pred = model_knn_sklearn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print ("Manual Dataset : ", accuracy*100)
```

```
Manual Dataset :  78.30188679245283
```

Fig 31. *KNN with Scikit-Learn Algorithm using DigitRecognition_Dataset*

3. Random Forest Algorithm

Create an image classifier using the Random Forest Algorithm by importing the required libraries. Then, create a RandomForestClassifier object with 100 thresholds and 10 jobs. The fit method is used for the training model on the X_train_mnist training data set. Next, we use the scores to calculate how confident we are that the predictions will be accurate for the X_test_mnist test data set. Then calculating accuracy as the proportion of correct predictions (y-pred-mnist) divided by all possible predictions (y-test-mnist). Finally, predict is called on this trained model to make predictions about the new instances in the X_test_ test data set. When using the MNIST Dataset the accuracy is 93.06% as shown in Figure 32, while when using the DigitRecognition_Dataset the results are 81.13% accurate as shown in Figure 33.

```
model_rf = RandomForestClassifier(n_estimators=100, n_jobs=10)
model_rf.fit(X_train_mnist,y_train_mnist)
confidence = model_rf.score(X_test_mnist,y_test_mnist)
y_pred_mnist = model_rf.predict(X_test_mnist)
accuracy = accuracy_score(y_test_mnist, y_pred_mnist)
print ("MNIST Dataset : ", accuracy*100)
```

```
<ipython-input-16-d210f56e470b>:2: DataConversionWarning: A col
  model_rf.fit(X_train_mnist,y_train_mnist)
MNIST Dataset :  93.06666666666666
```

Fig 32. *Random Forest Algorithm using MNIST Dataset*

```
model_rf = RandomForestClassifier(n_estimators=100, n_jobs=10)
model_rf.fit(X_train,y_train)
confidence = model_rf.score(X_test,y_test)
y_pred = model_rf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print ("Manual Dataset : ", accuracy*100)
```

```
Manual Dataset :  81.13207547169812
```

Fig 33. *Random Forest Algorithm using DigitRecognition_Dataset*

4. CNN with TensorFlow Algorithm

Create an image classifier using the CNN with TensorFlow Algorithm by defining the model_tensor variable. This is a Sequential Keras object that will be used to build the neural network. Next, delimit the dense layer with 10 neurons and a sigmoid activation function. The input shape for this layer is 400 x 10 (the number of inputs in each dimension). Then compile the model using adam as optimizer and loss='sparse_categorical_crossentropy' as metric. Finally, the activation function of this dense layer is set to the sigmoid function to be tested. When using the MNIST Dataset the accuracy is 86.59% as shown in Figure 34, while when using the DigitRecognition_Dataset the results are 73.58% accurate as shown in Figure 35.

```
# Sequential create a stack of layers
model_tensor = keras.Sequential([
    keras.layers.Dense(10, input_shape=(400,), activation='sigmoid')
])

# Optimizer will help in backproagation to reach better global optima
model_tensor.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

# Does the training
model_tensor.fit(X_train_mnist, y_train_mnist, epochs=50)
model_tensor.evaluate(X_test_mnist, y_test_mnist)
```

Fig 34. *CNN with TensorFlow using MNIST Dataset*

```
# Sequential create a stack of layers
model_tensor = keras.Sequential([
    keras.layers.Dense(10, input_shape=(10000,), activation='sigmoid')
])

# Optimizer will help in backproagation to reach better global optima
model_tensor.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

# Does the training
model_tensor.fit(X_train, y_train, epochs=50)
model_tensor.evaluate(X_test, y_test)
```

Fig 35. *CNN with TensorFlow using DigitRecognition_Dataset*

5. SVM Algorithm

Create an image classifier using the SVM Algorithm by dengan mengimpor perpustakaan yang diperlukan. Selanjutnya, Kemudian mencocokan dengan model pada data pelatihan X_train_mnist dan y_train_mnist. Langkah selanjutnya adalah memprediksi pada data uji X_test_mnist dan y_pred_. Terakhir, akurasi dihitung sebagai skor akurasi y-y' untuk setiap baris dalam x-x'.membuat model SVC dengan gamma=0.001 dan kernel='poly'. When using the MNIST Dataset the accuracy is 93,53% as shown in Figure 36, while when using the DigitRecognition_Dataset the results are 83,96% accurate as shown in Figure 37.

```
model_svm_mnist = svm.SVC(gamma=0.001, kernel='poly')
model_svm_mnist.fit(X_train_mnist,y_train_mnist)
y_pred_mnist = model_svm_mnist.predict(X_test_mnist)
accuracy = accuracy_score(y_test_mnist, y_pred_mnist)
print("MNIST Dataset : ", accuracy*100)
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/utils/va
  y = column_or_1d(y, warn=True)
MNIST Dataset :  93.53333333333333
```

Fig 36. *SVM Algorithm using MNIST Dataset*

```
model_svm_manual = svm.SVC(gamma=0.001, kernel='poly')
model_svm_manual.fit(X_train,y_train)
y_pred = model_svm_manual.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Manual Dataset : ", accuracy*100)
```

```
Manual Dataset :  83.9622641509434
```

Fig 37. *SVM Algorithm using DigitRecognition_Dataset*

E. Result & Final Discussion

In the results of this paper, the first author wants to show the results of an overview of one of the images that has been successfully recognized by the model and architecture after going through the pre-processing stage. As can be seen in Figure 38, where the model itself has been able to recognize the number 4 clearly, although of course the input as a variable is not fixed which is written by each person, of course it is different due to different writing styles, differences in structure and orientation angle.

Fig 38. *Model & Architecture Result*

Second, the author wants to show the results of the accuracy of the model and architecture after going through the pre-processing stage. As can be seen in Figure 39, the model has been able to recognize not only one input number but more than one. Even though there is still a lack of accuracy in the recognition output, the accuracy obtained from the model and architecture that has been made by the author can be said to be good and quite close when compared from the point of view of using the dataset which will be explained in the next results.

Fig 39. *Model & Architecture Accuracy*

Third, as is known from the discussion in the classification section, this paper obtains results where the SVM (Support Vector Machine Algorithm) algorithm is the best algorithm that can perform digit recognition accurately when compared to other algorithms. The author also conducted two trials of all methods where in the first trial the results can be seen in Table 1, when using the MNIST Dataset the accuracy for the SVM method was obtained as the best method of 86.59% and when using the handmade digit dataset, the accuracy was 73 .58%. The accuracy obtained from the dataset that the author made himself in the first trial has standard accuracy results because it is quite far from the accuracy of using the MNIST dataset.

Algorithm	**Dataset**	
	MNIST	**DigitRecognition**
KNN with OpenCV (k = 3)	93.46%	77.35%
KNN with Scikit-Learn (k=3)	93.13%	78.30%
Random Forest	93.06%	81.13%
CNN with TensorFlow (50 epochs)	86,59%	73,58%
SVM	**93.53%**	**83.96%**

Table 1. *Comparison Table Result of First Trial*

Then when the authors conducted a second trial of all methods, the results of which can be seen in Table 2, the accuracy of all methods has increased. This is because the author added to the number of datasets made on handmade digit datasets. Where is the SVM method as the best method to get an accuracy of 88.09%. The accuracy obtained from the dataset made by the author himself in the second trial has good accuracy results compared to the first trial. Because if quoted from (Gonzalez and Herrador, 2007), accuracy is determined by the total % recovery formula based on AOAC (2016) in a good accuracy range of 80-110% and precision with an RSD value below 7.3%. This indicates that the author has succeeded in increasing the accuracy of the model and architecture and the dataset from the author himself can be included in the benchmark dataset.

Algorithm	**Dataset**	
	MNIST	**DigitRecognition**
KNN with OpenCV (k = 3)	93.46%	83.33%
KNN with Scikit-Learn (k=3)	93.13%	84.12%
Random Forest	93.26%	86.11%
CNN with TensorFlow (50 epochs)	86,40%	82,53%
SVM	**93.53%**	**88.09%**

Table 2. *Results Table Comparison of the Second Trial*

F. *Implementation*

To make it easier for users to operate digit recognition, the author creates a website-based application platform using Streamlit which produces a canvas that can be draw as shown in Figure 40.

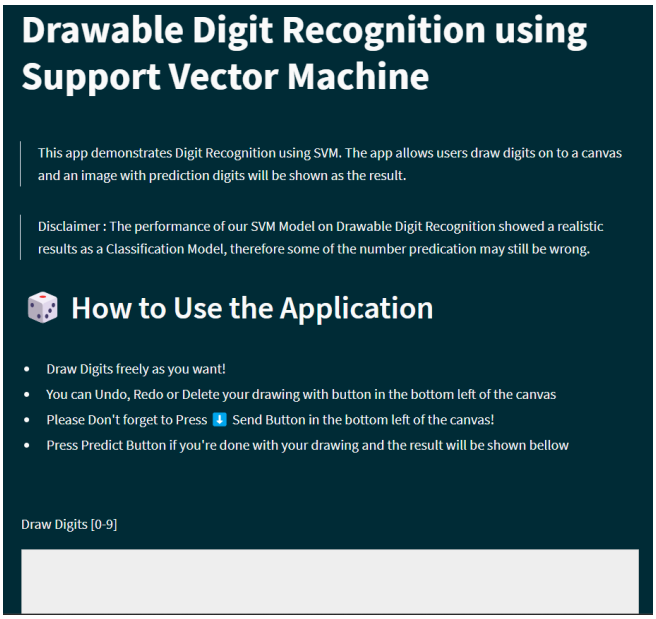

Fig 40. *Drawable Canvas for Digit Recognition based on Web Application using Streamlit*

Development is carried out using models and architectures and then taking samples of the input in the form of figures that have been drawn on the canvas as shown in Figure 41.

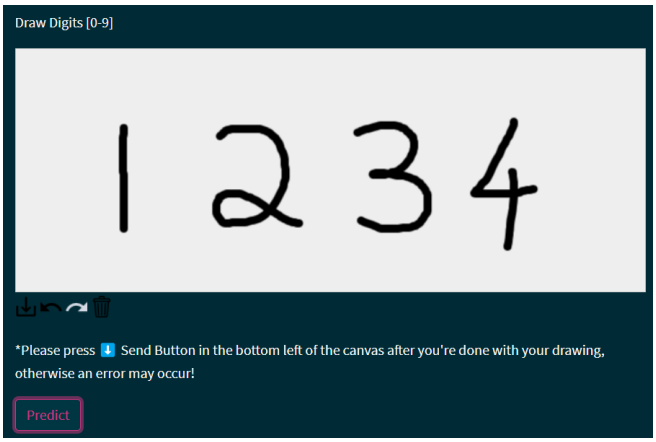

Fig 41. *Drawable Canvas for Digit Recognition*

The drawing number is unlimited, meaning you can draw more than 1 number on the canvas with a range of numbers from 0-9 which can later be predicted by the model and architecture as shown in Figure 42.

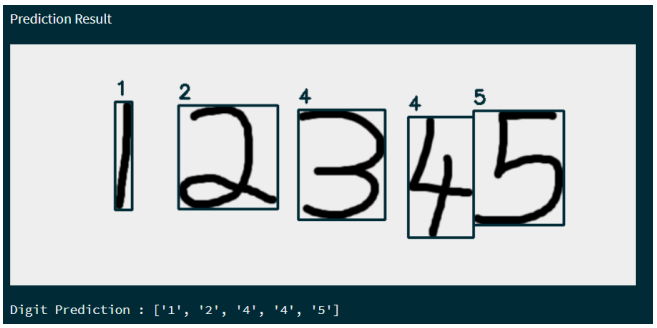

Fig 42. *Drawable Canvas for Digit Recognition Prediction Result*

This canvas drawable for digit recognition based on web application is available at the link https://fadlyhaikal-digitrecognition-app-c7mmqs.streamlit.app/ [35] which can be accessed freely and online.

V. Conclusion

So, in this paper we have analysed efficiency of all method which already experimented and compared to find out the best techniques between using Supervised Learning Algorithm are KNN Algorithm (K-Nearest Neighbours Algorithm) such as KNN with OpenCV Library and KNN with Scikit-Learn library, Random Forest Classification Algorithm and SVM Algorithm (Support Vector Machine Algorithm) and with Deep Learning Algorithm, CNN (Convolutional Neural Network) with TensorFlow on handwritten digit recognition.

This paper has results where the SVM (Support Vector Machine Algorithm) algorithm is the best algorithm that can perform digit recognition accurately when compared to other algorithms. Where in the first trial when using the MNIST Dataset an accuracy of 86.59% was obtained and when using a handmade digit dataset an accuracy of 73.58%. Then in the second trial an accuracy of 88.09% was obtained from the handmade digit dataset. A combination of different feature extraction methods can be used to reduce features. The combination of classifier and addition to the dataset can be used to increase recognition accuracy but will increase computation time. In the future this method will be applied to handwritten number recognition and character recognition in other languages as well.

VI. ACKNOWLEDGMENT

This study involved just two students and a lecturer at Bina Nusantara University, Jakarta, Indonesia, 2023. This research was conducted without receiving funds assistance from any agency. All research and projects are available at https://github.com/FadlyHaikal/DigitRecognition [36].

VII. References

[1] P. Ghadekar, S. Ingole, and D. Sonone, "Handwritten Digit and Letter Recognition Using Hybrid DWT-DCT with KNN and SVM Classifier," *Proc. - 2018 4th Int. Conf. Comput. Commun. Control Autom. ICCUBEA 2018*, Jul. 2018, doi: 10.1109/ICCUBEA.2018.8697684.

[2] Y. Bengio, "Learning Deep Architectures for AI," *Found. Trends® Mach. Learn.*, vol. 2, no. 1, pp. 1–127, 2009, doi: 10.1561/2200000006.

[3] D. Anguita, S. Ridella, F. Rivieccio, and R. Zunino, "Quantum optimization for training support vector machines," *Neural Networks*, vol. 16, no. 5–6, pp. 763–770, 2003, doi: 10.1016/S0893-6080(03)00087-X.

[4] N. Wiebe, A. Kapoor, and K. M. Svore, "Quantum algorithms for nearest-neighbor methods for supervised and unsupervised learning," *Quantum Inf. Comput.*, vol. 15, no. 3–4, pp. 318–358, 2015, doi: 10.26421/qic15.3-4-7.

[5] "Streamlit documentation." [Online]. Available: https://docs.streamlit.io/.

[6] "pyngrok - a Python wrapper for ngrok — pyngrok 5.2.1 documentation." https://pyngrok.readthedocs.io/en/latest/index.html (accessed Jan. 16, 2023).

[7] "Generating Random id's using UUID in Python - GeeksforGeeks." https://www.geeksforgeeks.org/generating-random-ids-using-uuid-python/ (accessed Jan. 16, 2023).

[8] "Localtunnel ~ Expose yourself to the world." https://theboroer.github.io/localtunnel-www/ (accessed Jan. 16, 2023).

[9] "localtunnel - npm." https://www.npmjs.com/package/localtunnel (accessed Jan. 16, 2023).

[10] "Complex and semantic figure composition — Matplotlib 3.5.2 documentation." Accessed: Jun. 18, 2022. [Online]. Available: https://matplotlib.org/stable/tutorials/provisional/mosaic.html.

[11] "Matplotlib — Visualization with Python." https://matplotlib.org/ (accessed Jan. 16, 2023).

[12] "io — Core tools for working with streams — Python 3.11.1 documentation." https://docs.python.org/3/library/io.html (accessed Jan. 16, 2023).

[13] "scikit-learn: machine learning in Python — scikit-learn 1.2.0 documentation." https://scikit-learn.org/stable/ (accessed Jan. 16, 2023).

[14] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, no. January, pp. 2825–2830, 2011.

[15] "NumPy documentation — NumPy v1.22 Manual." https://numpy.org/doc/stable/ (accessed Jun. 18, 2022).

[16] S. Van der Walt and M. Aivazis, "The NumPy Array: A Structure for Efficient Numerical Computation, Computing in Science & Engineering," *Comput. Sci. Eng.*, vol. 13, no. 2, pp. 22–30, 2011, [Online]. Available: http://aip.scitation.org/doi/abs/10.1109/MCSE.2011.37.

[17] "OpenCV: Introduction." https://docs.opencv.org/4.x/d1/dfb/intro.html (accessed Jan. 16, 2023).

[18] "Pillow (PIL Fork) 9.4.0 documentation." https://pillow.readthedocs.io/en/stable/ (accessed Jan. 16, 2023).

[19] "Tensorflow - Posit Documentation." https://docs.posit.co/resources/tensorflow/ (accessed Jan. 16, 2023).

[20] "Home - Keras Documentation." https://faroit.com/keras-docs/1.2.0/ (accessed Jan. 16, 2023).

[21] H. Dani, "Review on Frameworks Used for Deployment of Machine Learning Model," *Int. J. Res. Appl. Sci. Eng. Technol.*, vol. 10, no. 2, pp. 211–215, 2022, doi: 10.22214/ijraset.2022.40222.

[22] J. K. Rask, F. P. Madsen, N. Battle, H. D. Macedo, H. Daniel Macedo, and P. G. Larsen, "Visual Studio Code VDM Support," no. December, pp. 1–20, 2020, [Online]. Available: https://pypl.github.io/IDE.html.

[23] "(PDF) PENGGUNAAN PIRANTI LUNAK JUPYTER NOTEBOOK DALAM UPAYA MENSOSIALISASIKAN OPEN SCIENCE." [Online]. Available: https://www.researchgate.net/publication/326132474_PENGGUNAAN_PIRANTI_LUNAK_JUPYTER_NOTEBOOK_DALAM_UPAYA_MENSOSIALISASIKAN_OPEN_SCIENCE.

[24] "Google Colab vs Jupyter Notebook: Compare data science software |." https://www.techrepublic.com/article/google-colab-vs-jupyter-notebook/ (accessed Jan. 16, 2023).

[25] "'Hello World - GitHub Docs.' [Online]. Available: https://docs.github.com/en/get-started/quickstart/hello-world. - Google Search." https://www.google.com/search?q="Hello+World+-+GitHub+Docs."+%5BOnline%5D.+Available%3A+https%3A%2F%2Fdocs.github.com%2Fen%2Fget-started%2Fquickstart%2Fhello-world.&rlz=1C1CHBD_enID1007ID1007&oq="Hello+World+-+GitHub+Docs."+%5BOnline%5D.+Available%3A+https%3A%2F%2Fdocs.github.com%2Fen%2Fget-started%2Fquickstart%2Fhello-world.&aqs=chrome.0.69i59.233j0j7&sourceid=chrome&ie=UTF-8 (accessed Jan. 16, 2023).

[26] "Welcome To Colaboratory - Colaboratory." https://colab.research.google.com/ (accessed Jan. 16, 2023).

[27] J. Ali, R. Khan, N. Ahmad, and I. Maqsood, "Random forests and decision trees," *IJCSI Int. J. Comput. Sci. Issues*, vol. 9, no. 5, pp. 272–278, 2012.

[28] Z. Jin, J. Shang, Q. Zhu, C. Ling, W. Xie, and B. Qiang, "RFRSF: Employee Turnover Prediction Based on Random Forests and Survival Analysis,"

Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 12343 LNCS, pp. 503–515, 2020, doi: 10.1007/978-3-030-62008-0_35.

[29] K. Taunk, "2019 International Conference on Intelligent Computing and Control Systems, ICCS 2019," *2019 Int. Conf. Intell. Comput. Control Syst. ICCS 2019*, no. Iciccs, pp. 1255–1260, 2019.

[30] P. Cunningham and S. J. Delany, "K-Nearest Neighbour Classifiers-A Tutorial," *ACM Comput. Surv.*, vol. 54, no. 6, 2021, doi: 10.1145/3459665.

[31] "Support Vector Machine (SVM) — H2O 3.38.0.4 documentation." https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/svm.html (accessed Jan. 16, 2023).

[32] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Stat. Comput.*, vol. 14, no. 3, pp. 199–222, Aug. 2004, doi: 10.1023/B:STCO.0000035301.49549.88.

[33] "Convolutional Neural Network. Learn Convolutional Neural Network from… | by dshahid380 | Towards Data Science." https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529 (accessed Jan. 16, 2023).

[34] "CNN Tutorial — MinPy 0.3.4 documentation." https://minpy.readthedocs.io/en/latest/tutorial/cnn_tutorial/cnn_tutorial.html (accessed Jan. 16, 2023).

[35] "Streamlit." https://fadlyhaikal-digitrecognition-app-c7mmqs.streamlit.app/ (accessed Jan. 22, 2023).

[36] "FadlyHaikal/DigitRecognition." [Online]. Available: https://github.com/FadlyHaikal/DigitRecognition.