Volume 9, Issue 7, July – 2024
ISSN No:-2456-2165
International Journal of Innovative Science and Research Technology
https://doi.org/10.38124/ijisrt/IJISRT24JUL929

SVM, KNN, and Neural Networks Investigated for Machine Learning in Written Word Decoding

Gottipati Ajay[1]
Department of Computer Science and Enginnering,
Koneru Lakshmaiah Education Foundation,
Vaddeswaram, AP, India

Srungavarapu Bhuvanesh Babu[2]
Department of Computer Science and Engineering,
Koneru Lakshmaiah Education Foundation,
Vaddeswaram,AP, India

Madala Narasimha Rao[3]
Department of Computer Science and Engineering,
Koneru Lakshmaiah Education Foundation,
Vaddeswaram, AP, India

Magam Satya Siva Krishna[4]
Department of Computer Science and Engineering,
Koneru Lakshmaiah Education Foundation,
Vaddeswaram, AP, India

Dr. M. D Gouse[5]
(Professor)
Department of CSE,
Koneru Lakshmaiah Education Foundation,
Vaddeswaram, AP, India

Abstract:- The capacity of a device to recognise and understand legible handwriting input from a variety of origins, including written material, snap shots, displays, and other electronics, is known as handwritten reputation. In this study, we investigate three classification algorithms: Support Vector Machines (SVM), K-Nearest_Neighbours (KNN), and Neural Networks for handwritten character recognition, and we will identify the best one among these three.

Keywords:- *Handwritten popularity, SVM, Neural Network, K-Nearest Neighbor;*

I. INTRODUCTION

Handwriting perception empowers computers and devices to understand handwritten input, whether it is from bodily files, pictures, or direct touch display screen input. This era has end up fundamental to diverse gadgets together with smartphones, drugs, and PDAs, allowing users to rapidly enter numbers and textual content using a stylus or finger. The versatility of handwriting reputation has caused its widespread adoption in numerous packages these days. Among the strategies used for handwriting recognition is Optical Character Recognition (OCR), which extracts textual content from scanned documents and translates it into a format that computers can process. In this paper, we explore three category algorithms—Support Vector Machine (SVM), Neural Network, and K-Nearest_Neighbor (KNN) to beautify handwriting recognition competencies. We delve into the details of those algorithms later inside the paper, discussing their strengths and packages in recognizing handwritten text.

II. RELATED WORK

Several research studies have contributed significantly to the advancement of handwriting recognition in various fields. In a paper titled "Feature Set Evaluation for Offline Handwriting Recognition Systems: Application of Recurrent Neural Networks" Yusuf Chherawala, Partha Pratim Roy, and Mohammad Cheriet thus emphasize the important role of feature extraction in handwriting recognition systems if they emphasize that the systems are effective rely heavily on extracting features from handwritten word pictures[3]. While there are a variety of feature extraction methods, identifying the most promising ones often requires more than a simple comparison based on detection rates To overcome this challenge a system of propose a list of items based on a common design. Their methodology predicts the significance of various feature types, demonstrating the robustness and universality of the technique through an integrated idea of embedded RNN (recurrent neural network) classifiers.

In another research entitled "Handwriting Digit Detection and K-Nearest Neighbor Classification Using Local Binary Pattern Variance", Nurul Ilmi, Tjokorda Agung Budi W, and Kurniawan Nur R focus on digit detection in handwriting using local binary pattern variance method in addition to K-Nearest over Neighbors (KNN) classification[2]. Their study has focused exclusively on the C1 form that Indonesia's General Electoral Commission uses. Through the experiment, it was observed that although the LBP variance method achieved an impressive accuracy of 89.81 percent in identifying the handwriting characters in the MNIST dataset, when applied to the data from the C1 form in which the accuracy remained promising at 70.91 percent [2] .

Researchers Kutscher, T., Dietze, M., Bönninger,I. Travieso, C. M., Dutta, M. K., & Singh, A. conducted a study titled "Online handwriting verification with safe password and increasing number of features "[4]. Their primary objective is to reduce processing time or reduce volume while maintaining or improving distribution performance. Their analysis shows that the Bayes net classifier outperformed the others, achieving 100 percent correct accuracy in terms of lead time, speed andcorrelation, with a false acceptance rate (FAR) of only 3.13 [4].

In a provocative study titled "Identifying Human Personality Parameters Based on Handwriting Using Neural Networks", Behnam Fallah and Hasan Khotanlo investigate the possible relationship between handwriting characteristics and human personality traits is a hidden Markov model of neural networks (MLP) are concerned for handwriting-personality based classification in their approach[9]. f combination is employed integrating dependent and independent text features into the feature extraction process Their identity recognition system proposed makes it more accurate and reliable Furthermore, their automated method eliminates the need for segmentation time a feature extraction is eliminated, while techniques such as Gaussian discriminant analysis (GDA) are used to optimize class separation

Collectively, these research efforts also highlight the importance of handwriting recognition technology for a variety of applications, ranging from increased classification accuracy to strong correlations with handwriting characteristics searching between human minds.

III. RECOMMENDED SYSTEMS

We utilised sklearn, openCV, and python for this gadget's categorization and dataset examination. For training and type evaluation, we used the MNIST dataset. An assessment tool for machine learning styles on handwritten digit problems is the MNIST problem dataset. The dataset was created by combining several scanned document datasets that were made accessible by the National Institute of Standards and Technology (NIST). There are 748 total pixels in each image on this dataset, measuring 28 by 28 pixels [5]. In the dataset, there are seventy thousand images that can be utilised for both training and inspection of the gadget. Three (3) categorization sets of rules, namely Support Vector Machine (SVM), Multi Layer Perceptron and K-Nearest_Neighbour, were employed in this proposed gadget to determine its popularity.

A. Support Vector Machine (SVM)

SVM is a collection of supervised learning techniques used in classification, regression, and outlier identification. In SVM, the price of each feature is the fee of a certain coordinate. Each record item can be plotted as a factor in n-dimensional space (n = number of feature). The method used to complete the type is to identify the hyper-plane that separates the two (2) instructions[6].

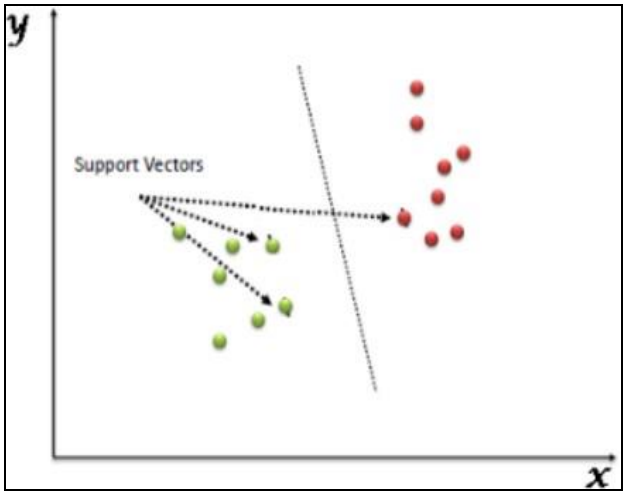

Fig 1 SVM Categorization

The SVM implementation in scikit-learn, namely LinearSVC, works well with both sparse and dense data sources. With the use of the liblinear library and a linear kernel, LinearSVC provides more options for selecting loss functions and penalties. Because of this, it is especially effective at managing huge datasets like MNIST, which has a huge collection of handwritten digit pictures. There will be two reports in this category: generate_Classifier.Py for the category and perform_Recognition.Py for the type experiment. There are three steps that we can complete in the generate_Classifier.Py, including: a. Determine the HOG characteristics for each sample in the database. B. Use each sample's HOG capabilities at the side of the matching label to train a multi-class linear SVM. C. Store the classifier in a data file.

```
12 # Load the dataset
13 dataset = datasets.fetch_mldata("MNIST Original")
```
Fig 2 Acquire MINST Dataset.

The MNIST dataset will be downloaded, as demonstrated in Distinct 2. Then, we may extract the MNIST dataset's characteristic using Histogram of Oriented Gradients (HOG) feature extraction. The digit photos from the dataset may be stored in a numpy array with labels that match. The HOG functions for every snapshot can then be computed and saved in a different numpy array. The coding is displayed in figure three below.

International Journal of Innovative Science and Research Technology
https://doi.org/10.38124/ijisrt/IJISRT24JUL929

```
# Extract the features and labels
features = np.array(dataset.data, 'int16')
labels = np.array(dataset.target, 'int')

# Extract the hog features
list_hog_fd = []
for feature in features:
    fd = hog(feature.reshape((28, 28)), orientations=9, pixels_per_cell=(14, 14), cells_per
    list_hog_fd.append(fd)
hog_features = np.array(list_hog_fd, 'float64')
```

Fig 3 Features Detection

In order to compute HOG functions, we first set the size of cells to be 14 × 14. Since the MNIST dataset is 28 by 28 pixels in size, we can have four (4) blocks or cells that are each 14 by 14. The vector of orientation is approximately nine. This suggests that the HOG function vector may have a dimension of 4 x 9 = 36. The classifier can then be kept in a report, as seen in parent four below, once we have created a Linear SVM object and completed the dataset's education.

```
# Create an Linear SVM object
clf = LinearSVC()

# Perform the training
clf.fit(hog_features, labels)

# Save the classifier
joblib.dump((clf, pp), "digits_cls.pkl", compress=3)
```

Fig 4 Linear SVM Categorization

B. K-Nearest_Neighbor (KNN)

The K-nearest_neighbours classifier is a popular technique in the realm of image classification due to its simplicity and effectiveness. Unlike more complex algorithms that require extensive training processes, KNN operates on a different principle. It doesn't undergo traditional learning iterations with labeled data. Instead, KNN relies on the inherent structure of the data itself.

The idea of closeness or proximity between feature vectors is fundamental to KNN. When a new, unlabeled data point is introduced, KNN looks around it by calculating the separations between the known data points that are the closest to it. The categorization of the new data point is mostly dependent on these neighbours, whose characteristics are most similar to it.

Suppose that every data point in the dataset corresponds to an image that has different properties, including texture patterns, colour distributions, or pixel values. Based on how comparable these features are, KNN finds the nearest neighbours of a new image that is included for classification.

This is where the idea of "voting" is relevant. In a sense, every neighbour "votes" for the class to which it belongs. For example, the new data point is likely to be classed as belonging to Class A if the majority of its closest neighbours are also in Class A. By using a democratic procedure, the classification decision is made based on the opinions of all nearby neighbours, as opposed to just one particular criterion.

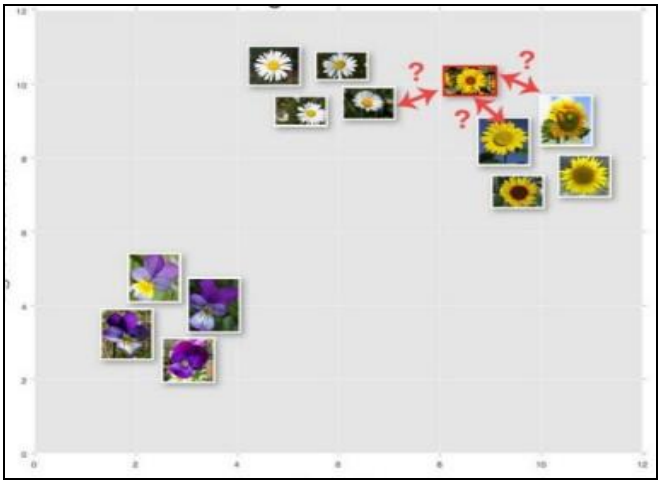

Fig 5 KNN Classification

➢ *To train an image classifier, we go through five steps:*

- Prepare the dataset: We organize the data for training and evaluation.
- Divide the data: The dataset is split into sections for testing and training.
- Extract features: We identify important characteristics from the images.
- Train the model: We teach the classifier to recognize patterns in the features.
- Evaluate the model: We check how well the classifier performs using the testing data.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24JUL929

We import the MNIST dataset and divide it into two parts: 25% for testing and 75% for training. Ten percent of the training data were reserved for validation.

```
# load the MNIST digits dataset
mnist = datasets.load_digits()

# take the MNIST data and construct the training and testing split, using 75% of the
# data for training and 25% for testing
(trainData, testData, trainLabels, testLabels) = train_test_split(np.array(mnist.data),
      mnist.target, test_size=0.25, random_state=42)

# now, let's take 10% of the training data and use that for validation
(trainData, valData, trainLabels, valLabels) = train_test_split(trainData, trainLabels,
      test_size=0.1, random_state=84)
```

Fig 6 Divided Validation, Testing, and Training.

The classifier is then trained, and the optimal value for k—a parameter used in the K-nearest_neighbours (KNN) algorithm—is ascertained. Furthermore, we compute the classifier's accuracy. We cycle over a range of k numbers, from 1 to 15, in this process. The classifier's performance is next evaluated in order to confirm its efficiency, as shown in Figure seven below.

```
# loop over various values of `k` for the k-Nearest Neighbor classifier
for k in range(1, 30, 2):
      # train the k-Nearest Neighbor classifier with the current value of `k`
      model = KNeighborsClassifier(n_neighbors=k)
      model.fit(trainData, trainLabels)

      # evaluate the model and update the accuracies list
      score = model.score(valData, valLabels)
      print("k=%d, accuracy=%.2f%%" % (k, score * 100))
      accuracies.append(score)

# find the value of k that has the largest accuracy
i = np.argmax(accuracies)
print("k=%d achieved highest accuracy of %.2f%% on validation data" % (kVals[i],
      accuracies[i] * 100))
```

Fig 7 Classifier for Training and Validation

Afterward, we apply the trained classifier to the testing dataset, utilizing a k value set to 1. Following this, we conduct the final evaluation of the classifier's performance, as illustrated in Figure seven.

```
# re-train our classifier using the best k value and predict the labels of the
# test data
model = KNeighborsClassifier(n_neighbors=kVals[i])
model.fit(trainData, trainLabels)
predictions = model.predict(testData)

# show a final classification report demonstrating the accuracy of the classifier
# for each of the digits
print("EVALUATION ON TESTING DATA")
print(classification_report(testLabels, predictions))
```

Fig 8 Testing Classifier

Next, we will choose five (5) photos at random from the testing dataset in order to investigate each unique prediction. We'll analyze the classifier's output for each of these images.

```
# loop over a few random digits
for i in np.random.randint(0, high=len(testLabels), size=(5,)):
    # grab the image and classify it
    image = testData[i]
    prediction = model.predict(image)[0]

    # convert the image for a 64-dim array to an 8 x 8 image compatible with OpenCV,
    # then resize it to 32 x 32 pixels so we can see it better
    image = image.reshape((8, 8)).astype("uint8")
    image = exposure.rescale_intensity(image, out_range=(0, 255))
    image = imutils.resize(image, width=32, inter=cv2.INTER_CUBIC)

    # show the prediction
    print("I think that digit is: {}".format(prediction))
    cv2.imshow("Image", image)
    cv2.waitKey(0)
```

Fig 9 Analysing the Classifier

C. Multi-Layer Perceptron (MLP)

The Multi-layer Perceptron (MLP) is a supervised learning algorithm that aims to learn a function: $f(\cdot) : Rm \rightarrow Ro$

The number of input dimensions, denoted by m, and the number of output dimensions, denoted by o, are used to train the function on a dataset. For either regression or classification problems, the MLP may learn a non-linear function approximator given a target y and a set of features X = x1, x2,... xm. By adding one or more non-linear layers—also referred to as hidden layers—between the input and output layers, it differs from logistic regression. Neural network topologies called Multi-layer Perceptrons (MLPs) may identify complex patterns in data. Multiple hidden layers with non-linear activation functions are a feature of MLPs, in contrast to more straightforward models like logistic regression. Consequently, they perform remarkably well in fields such as natural language processing, picture identification, and financial forecasting. Throughout training, MLPs iteratively modify weights and biases to improve their representations, enabling them to generalise and produce precise predictions on fresh data. Because of their versatility, MLPs are often used for a wide range of supervised learning tasks involving intricate, non-linear interactions.

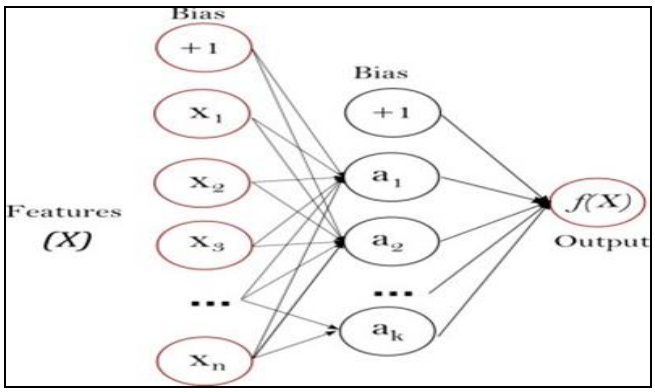

Fig 10 The Hidden Layer of MLP

Two files will be used to perform the classification process: perform_Recognition.py for classification evaluation and generate_Classifier-nn.py for handling the actual classification.

- *Within generate_Classifier-nn.py, We'll take three crucial steps:*

- Calculating HOG Features: For every sample in the database, this entails calculating the Histogram of Oriented Gradients (HOG) features.
- Training a Multi-class MLP Neural Network: Utilising the HOG features of every sample together with their respective labels, we will train a Multi-Layer Perceptron neural network which is multi-class.
- Saving the classifier: Once the classifier is trained, we'll save it into a file for later use.

```
# Load the dataset
dataset = datasets.fetch_mldata("MNIST Original")
```

Fig 11 Download MNIST Dataset

Initially, we will download the MNIST dataset, which is made up of handwritten numbers. Next, we'll use a method known as HOG (Histogram of Oriented Gradients) to take these digit snapshots and extract features. The digit images and the labels that go with them will be arranged into numpy arrays. Next, for every image, we will compute the HOG features and store them in a different numpy array. This procedure's detailed coding instructions can be found in the figure 12 beneath.

International Journal of Innovative Science and Research Technology
https://doi.org/10.38124/ijisrt/IJISRT24JUL929

```
# Extract the features and labels
features = np.array(dataset.data, 'int16')
labels = np.array(dataset.target, 'int')

# Extract the hog features
list_hog_fd = []
for feature in features:
    fd = hog(feature.reshape((28, 28)), orientations=9, pixels_per_cell=(14, 14), cells_per
    list_hog_fd.append(fd)
hog_features = np.array(list_hog_fd, 'float64')
```

Fig 12 Extraction of Features

In order to calculate the HOG features, we will use a 14 × 14 cell size. Since the MNIST dataset images are 28 x 28 pixels, we'll divide them into four blocks or cells, each with a size of 14X14 pixels. We'll define the orientation vector to 9, resulting in a feature vector of HOG size of 4 x 9 = 36.

Following that, we'll instantiate a MLP (Multi-Layer Perceptron) Neural Network object & train it using the dataset. As seen in Figure 13, after training, we'll save the classification model into a file.

```
# Create an MLP Neural Network object
clf = MLPClassifier(solver='lbfgs', alpha=1e-5,
                    hidden_layer_sizes=(5, 2), random_state=1)

# Perform the training
clf.fit(hog_features, labels)
```

Fig 13 Neural Network Classification using MLP

D. Experiment

In order to evaluate the classifiers developed using the MLP (Multi-Layer Perceptron) neural networks and Support Vector Machine (SVM), we'll utilize the perform_ Recognition.py script. Within this script, we'll load the classifier files that were previously saved, as demonstrated in Figure 14.

```
# Load the classifier
clf, pp = joblib.load(args["classiferPath"])
```

Fig 14 Call Classifier File

After loading the image intended for testing the classifier, we'll preprocess it. This preprocessing involves converting the image to grayscale and applying Gaussian filtering to reduce noise. Subsequently, we'll apply thresholding to segment the image and identify contours within it. Following this, we'll extract the rectangles that encompass each contour, as illustrated in Figure 15.

```
# Read the input image
im = cv2.imread(args["image"])

# Convert to grayscale and apply Gaussian filtering
im_gray = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
im_gray = cv2.GaussianBlur(im_gray, (5, 5), 0)

# Threshold the image
ret, im_th = cv2.threshold(im_gray, 90, 255, cv2.THRESH_BINARY_INV)

# Find contours in the image
ctrs, hier = cv2.findContours(im_th.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Get rectangles contains each contour
rects = [cv2.boundingRect(ctr) for ctr in ctrs]
```

Fig 15 Preprocessing the Image

International Journal of Innovative Science and Research Technology
https://doi.org/10.38124/ijisrt/IJISRT24JUL929

The classifier will then be applied to the test image. This entails computing the image's HOG features and applying them to forecast the digit that the image represents. The final result and the anticipated number will then be shown, as seen in Figure 17.

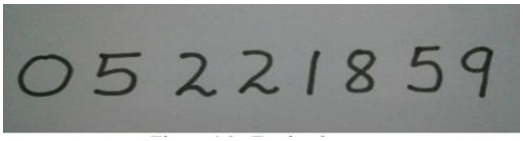

Fig 16 Testing image

```
# For each rectangular region, calculate HOG features and predict
# the digit using classifier
for rect in rects:
    # Draw the rectangles
    cv2.rectangle(im, (rect[0], rect[1]), (rect[0] + rect[2], rect[1] + rect[3]), (0, 255,
    # Make the rectangular region around the digit
    leng = int(rect[3] * 1.6)
    pt1 = int(rect[1] + rect[3] // 2 - leng // 2)
    pt2 = int(rect[0] + rect[2] // 2 - leng // 2)
    roi = im_th[pt1:pt1+leng, pt2:pt2+leng]
    # Resize the image
    roi = cv2.resize(roi, (28, 28), interpolation=cv2.INTER_AREA)
    roi = cv2.dilate(roi, (3, 3))
    # Calculate the HOG features
    roi_hog_fd = hog(roi, orientations=9, pixels_per_cell=(14, 14), cells_per_block=(1, 1),
    roi_hog_fd = pp.transform(np.array([roi_hog_fd], 'float64'))
    nbr = clf.predict(roi_hog_fd)
    cv2.putText(im, str(int(nbr[0])), (rect[0], rect[1]),cv2.FONT_HERSHEY_DUPLEX, 2, (0, 25

cv2.namedWindow("Resulting Image with Rectangular ROIs", cv2.WINDOW_NORMAL)
cv2.imshow("Resulting Image with Rectangular ROIs", im)
cv2.waitKey()
```

Fig 17 Examining the Classifier

For KNN (K-Nearest_Neighbors), we'll conduct experiments similar to those described in the previous sections for MLP Neural Network and SVM. We'll evaluate the classification model using random images from the testing dataset. This process involves testing the classifier on a variety of images to assess its performance across different digit representations.

IV. RESULT

The experiment shows consistent findings for K-Nearest Neighbours (KNN) for a variety of values of the parameter k, which vary from 1 to 15. The accuracy obtained from all these values is a remarkable 99.26 percent, meaning that the accuracy stays high no matter which value of k is selected within this range. In the following figure, this observation is illustrated.

```
k=1, accuracy=99.26%
k=3, accuracy=99.26%
k=5, accuracy=99.26%
k=7, accuracy=99.26%
k=9, accuracy=99.26%
k=11, accuracy=99.26%
k=13, accuracy=99.26%
k=15, accuracy=99.26%
k=17, accuracy=98.52%
k=19, accuracy=98.52%
k=21, accuracy=97.78%
k=23, accuracy=97.04%
k=25, accuracy=97.78%
k=27, accuracy=97.04%
k=29, accuracy=97.04%
k=1 achieved highest accuracy of 99.26% on validation data
```

Fig 18 The classifier's accuracy

The testing data evaluation is depicted in the figure 19.

EVALUATION ON TESTING DATA

	precision	recall	f1-score	support
0	1.00	1.00	1.00	43
1	0.95	1.00	0.97	37
2	1.00	1.00	1.00	38
3	0.98	0.98	0.98	46
4	0.98	0.98	0.98	55
5	0.98	1.00	0.99	59
6	1.00	1.00	1.00	45
7	1.00	0.98	0.99	41
8	0.97	0.95	0.96	38
9	0.96	0.94	0.95	48
avg / total	0.98	0.98	0.98	450

Fig 19 Assessment of the test data

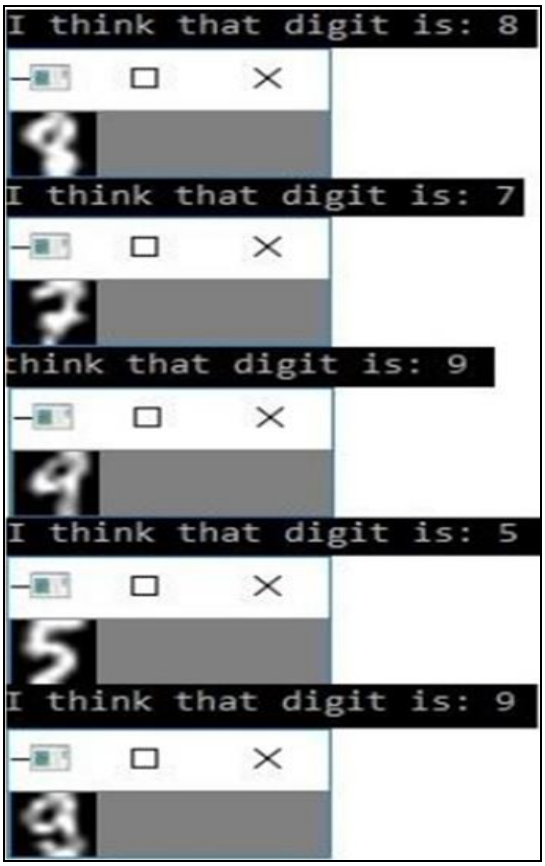

Fig 20 Analysing the Outcome

As seen in the figure 20, the analysing classifier uses five (5) random images from the training dataset.

The SVM classification result is displayed in the figure 21.

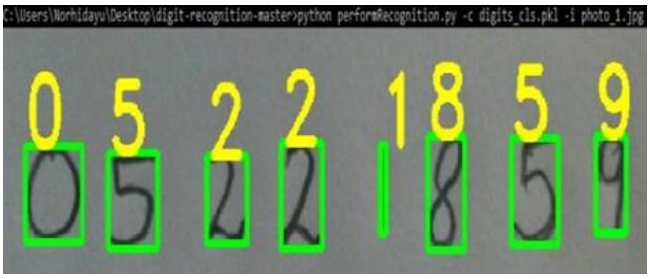

Fig 21 SVM Classification Model Output

The next figure 22 displays the MLP Neural Network classification result.

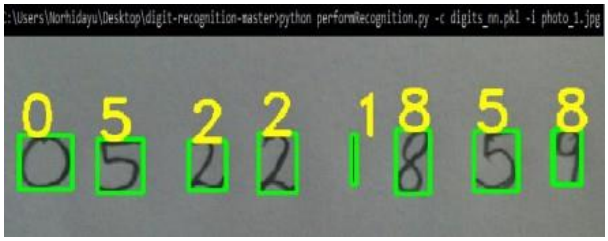

Fig 22 MLP Neural Network Result

V. CONCLUSION

The outcomes show that the dataset is accurately classified by both KNN (K-Nearest_Neighbours) and SVM (Support Vector Machine) classifiers. But there are certain inaccuracies with the MLP (Multi-Layer Perceptron) Neural Network, especially when it comes to forecasting the number nine (9). This disparity results from the fact that MLP uses a non-linear function, whereas KNN and SVM predict based only on the features that have been retrieved. Multiple local minima result from the non-convex loss functions of MLPs, particularly those with hidden layers. As a result, different random weight initializations may produce varied validation accuracy. This issue can be mitigated by employing Convolutional Neural Networks (CNNs) with frameworks like Keras, which are better suited for capturing complex, non-linear relationships in data.

REFERENCES

[1]. Shakoor, U., Mim, S. S., & Logofatu, D. (2023). Use of machine learning algorithms to analyze the digit recognizer problem in an effective manner. Artificial Neural Networks and Machine Learning – ICANN 2023, 496-507. https://doi.org/10.1007/978-3-031-44201-8_40

[2]. Ilmi, N., Budi, W. T., & Nur, R. K. (2016). Handwriting digit recognition using local binary pattern variance and k-nearest neighbor classification. 2016 4th International Conference on Information and Communication Technology (ICoICT). https://doi.org/10.1109/icoict.2016.7571937

[3]. Chherawala, Y., Roy, P. P., & Cheriet, M. (2016). Feature set evaluation for offline handwriting recognition systems: Application to the recurrent neural network. IEEE Transactions on Cybernetics, 46(12), December 2016.

[4]. Kutzner, T., Dietze, M., Bönninger, I., Travieso, C. M., Dutta, M. K., & Singh, A. (2016). Online handwriting verification with safe password and increasing number of features. 2016 3rd International Conference on Signal Processing and Integrated Networks (SPIN).

[5]. Yao, Y., & Cao, J. (2017). An adaptive scheduling mechanism for analytical workflow model. Communications in Computer and Information Science, 31-45. https://doi.org/10.1007/978-981-10-3996-6_3

[6]. Belavagi, M. C., & Muniyal, B. (2016). Performance evaluation of supervised machine learning algorithms for intrusion detection. Procedia Computer Science, 89, 117-123. https://doi.org/10.1016/j.procs.2016.06.016

[7]. El-Bendary, N., Zawbaa, H. M., Daoud, M. S., Hassanien, A. E., & Nakamatsu, K. (2010). ArSLAT: Arabic sign language alphabets translator.2010 International Conference on Computer Information Systems and Industrial Management Applications (CISIM). https://doi.org/10.1109/cisim.2010.5643519

[8]. Seiderer, A., Flutura, S., & André, E. (2017). Development of a mobile multi-device nutrition logger. Proceedings of the 2nd ACM SIGCHI International Workshop on Multisensory Approaches to Human-Food Interaction. https://doi.org/10.1145/3141788.3141790

[9]. Fallah, B., & Khotanlou, H. (2016). Identify human personality parameters based on handwriting using neural network. 2016 Artificial Intelligence and Robotics (IRANOPEN). https://doi.org/10.1109/rios.2016.7529501