Handwritten Digit Classification

1.	ARJUN SETH	(BT18GCS122)	CSE:Data Science
2.	AKSHAT BHANDARI	(BT18GCS129)	CSE:Data Science
3.	TRISHA SHARMA	(BT18GCS019)	CSE:Data Science

Under supervision of

MR. KARTIKAY GUPTA

NIIT University, Neemrana, Rajasthan-301705

1. INTRODUCTION:

In the current age of digitization, handwriting recognition plays an important role in information processing. There is a lot of information available on paper and less processing of digital files than the processing of traditional paper files. The purpose of the handwriting recognition system is to convert handwritten letters into machine-readable formats. Major applications include vehicle license-plate identification, postal paper-sorting services, historical document preservation in the check truncation system (CTS) scanning and archaeology departments, old document automation in libraries and banks, and more. All of these areas deal with large databases and therefore require high identification accuracy, low computational complexity, and consistent performance of the identification system.

Over time, the number of fields that can implement deep learning is increasing. In deep learning, convolutional neural networks (CNN) are being used for visual image analysis. CNN can be used in object detection, facial recognition, robotics, video analysis, segmentation, pattern recognition, natural language processing, spam detection, topical gradation, regression analysis, speech recognition, image classification.

Detection of handwritten numbers, including accuracy in these areas, has reached human perfection using deep convolutional neural networks (CNNs). Recently CNN has become one of the most attractive approaches and has been the ultimate factor in recent success and in several challenging machine learning applications. Considering all the factors stated above we have chosen CNN for our challenging tasks of image classification. We can use it to identify handwritten numbers, which is one of the higher education and business transactions. There are many applications of

handwritten digit recognition for our real-life purposes. Hence we are using the Convolutional Neural Network (CNN) and MNIST dataset.

2. PROBLEM STATEMENT & OBJECTIVE

Handwriting recognition has been the main subject of research for almost the last forty years. This research work analyzes the behaviour of classification techniques (CNN) in a large handwriting dataset (MNIST) to predict a digit. Machine-learning techniques, particularly when applied to Neural Networks like CNN or ANN, have played an increasingly important role in the design of these recognition systems. Several methods have been developed in handwritten digit recognition and these methods have been classified into categories: knowledge-based methods, feature-based methods, template-based methods and appearance-based methods. Errors in Digit recognition cause severe problems like digits written on a bank cheque if recognized erroneously could result in unfortunate consequences.

The goal of our work is to create a model that will be able to recognize and classify the handwritten digits from images by using concepts of Convolution Neural Network. Though the goal of our research is to create a model for digit recognition and classification, it can also be extended to letters and an individual's handwriting. The major goal of the proposed system is understanding Convolutional Neural Network, and applying it to the handwritten digit recognition system by working on the MNIST dataset.

There have already been significant advancements in this area of research previously. We have tried to form a model around the Conventional Neural Network with MNIST as our dataset so that the model has high accuracy and has been trained and tested on a large dataset. We shall also consider developing a robust test harness for estimating the performance of the model and then exploring improvements to the model. With high accuracy rates, the model can solve a lot of real-life problems.

3. LITERATURE REVIEW

Handwritten digit recognition has gained the interest of many researchers in recent years for its wide use in the field of text recognition systems in all languages and scripts. Research [11] by Anuj Dutt and Aashi Dutta, showed a comparison using the MNIST dataset between multilayer CNN using TensorFlow and Keras withTheano and machine learning algorithms like SVM, KNN, and RFC. The results show that the machine learning algorithms are good with high dimensional, imbalance, and missing dataset but they have slow processing speed, can't handle the curse of dimensionality, and are sensitive to outliers. They also have a comparably low accuracy rate than Convolutional Neural networks (CNN). because of this CNN is being used on a larger scale in image classification. In some other research papers [12,14], the CNN model is used for classification with backpropagation neural networks and is trained with a set of handwritten digits. For this, two datasets were created in different languages i.e Arabic and English. The dataset was composed of 46,000 digits and was partitioned into two sets, the first set of 36,00 samples for training and the second set of 10,000 samples for testing. In the proposed model CNN was used for classification along with convolution and morphological operations for noise reduction on the image. The accuracy of this CNN model was high i.e 95.7%, the only difficulty was that it can't classify the images in different orientations and was a bit slow due to the max pool layer.

In another research[13] two deep learning algorithms were used i.e Artificial neural network (ANN) and Convolutional Neural networks (CNN). The ANN method has an advantage, that external feature extraction techniques are not required in pattern recognition. However, it does not perform well in the case of large image sizes. On the other hand, CNN gives high accuracy with any size of images. Although the CNN model can be a bit more complex to train than other neural networks, it gives higher accuracy.

In research[1], the main focus is to achieve equivalent accuracy by using a pure CNN(Convolutional Neural Network) architecture but without ensemble architecture, as ensemble architecture brings along high testing complexity and an increase in the cost of computation. Therefore, they propose a CNN architecture without an ensemble architecture so that operational complexity and cost could be reduced. Also, they develop a relevant combination of learning parameters in plotting a CNN that helps them to achieve a good result in classifying MNIST. They were successful in getting a recognition accuracy of 99.87%.[1] Some researchers[2] also used the KNN (K Nearest Neighbours) technique.KNN is also known as a lazy learner (instance-based learning). It learns nothing during training. It does not receive any indiscriminate performance from training data. In other words, it does not have a training period. It stores training datasets and only learns from them when making

real-time estimates. Therefore the KNN algorithm is much faster in comparison to other algorithms that undergo training like Linear Regression, SVM, etc. One of the biggest drawbacks of using KNN is that it does not work very accurately when the data is of high dimension because then it becomes hard for the algorithm to compute the distance in each and every dimension. Also, KNN is sensitive to noise.[2]

Here they compare [3] that using CNN along with TensorFlow returns a much higher accuracy in comparison with other algorithms like KNN, SVM, RFC etc. They achieved better performance for CNN. In this experiment, they found that at epoch 15 the maximum training accuracy was both 100% and the maximum verification accuracy was 99.92%. The overall performance of the network was 99.21%. In addition, the total loss ranged from 0.026303 to 0.049449. Therefore, the method of CNN works more effectively than any other method for digit recognition. However, CNN does not include the position and orientation of the object in their estimates. They completely lose their internal data about the orientation of the object and they move all the information to a single neuron, which may not be able to deal with such information.[3][5]

It is said that[4] the traditional convolutional neural network treats the mapping of image pixels with surrounding space rather than as a fully connected layer of neurons. It is a powerful tool in signal and image processing. Even in the fields of computer vision, such as handwriting recognition, natural object classification, and segmentation, CNN is a much better tool than all the tools already in place. Developing a machine learning model that recognizes people's handwriting may be a broader goal, but can be achieved[4]. While developing the CNN, sometimes a pooling layer is inserted after each convolutional layer to minimize the spatial size of feature maps. Also, pooling layers are known for overcoming the overfitting problem. They[4] choose the pooling size to minimize the number of parameters by selecting the maximum, average or total value within these pixels. Since hardening and pooling layers reduce the complexity of time-space, a fully integrated network can finally be built to classify images.

According to this research[4], a flaw in CNN is that CNN does not include the position and orientation of the object in their estimates. They completely lose their internal data about the location and orientation of the object and they move all the information to the same neurons, which may not be able to deal with such information[5]. A CNN looks at an image and checks if there are any parts in the image. If they are, it will categorize the image accordingly.[4]

While CNN does provide higher accuracy rates than some of the other alternatives. There have been other approaches like combining classifiers, using Multilayer Perceptrons, using probabilistic neural networks or using Support vector machines. In a combination of classifiers, the classifier that

is closest to the true data generating process will always be the most accurate one out of the combination. Combined classifiers suffer from a lack of interpretability and computational infeasibility[7]. Using Multilayer perceptrons (MLP) does increase the number of parameters significantly which gives rise to redundancy. An MLP also disregards spatial information[9]. Support vector machines could prove to be highly memory efficient but since it is not suitable for large data sets using SVM could be disadvantageous[10]. PNN networks do generate accurate target probability scores but since they require more memory space to store the model they are inferior to CNN[10]. CNN does try to overcome all these limitations as it is easily interpreted and is computationally feasible. The redundancy problem which is present while using MLP is reduced in CNN making the latter more feasible. The efficiency of CNN with large data sets makes it a great alternative to SVM. We could also use the Hidden Markov Model (HMM) to solve this problem but a large number of unstructured parameters make it difficult to use HMM and hence we aren't using it to solve the problem.

The use of the backpropagation algorithm has its advantages as there are no parameters to tune while using it other than the inputs. Since it doesn't require any prior knowledge about the network it is quite flexible and easily usable. Other than this Backpropagation algorithm has its disadvantages as well as it might get us locked in a local optimum[8].

The MNIST dataset with its variety of handwritten digits has an easily understandable domain which allows fast and efficient comparisons between different methods[6,9]. With 60,000 images in the training set and 10,000 in the training set, MNIST is a substantially large dataset as compared to some other datasets. Some of the previous research work had considered custom datasets which bring variety into the picture as different styles and sizes help us create a better-trained model. Although there are advantages of using these custom datasets, these datasets also introduce classification errors into the model. Thus we are going to use MNIST as our training and testing dataset to solve all these issues[7,8].

In some of the research work, the error rates were high due to their model not being able to distinguish between similar classes[10]. These models weren't able to differentiate between classes '6' and '9' because some of their features were rotation invariants[7]. This led us to not using their techniques to avoid these types of discrepancies.

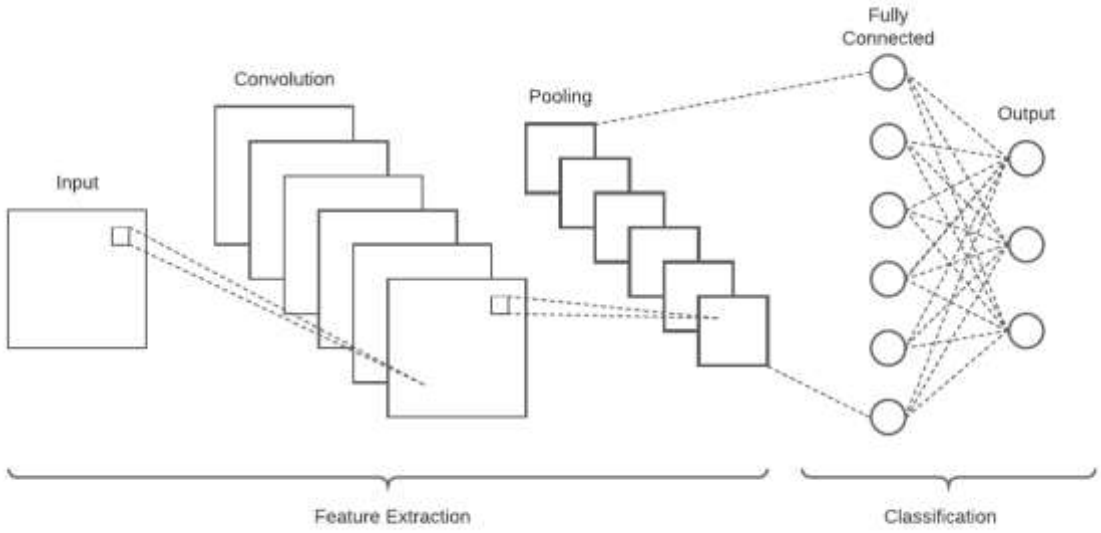
4. PROPOSED METHODOLOGY TO SPECIFIC GAP AND CHALLENGE

We can easily now figure out that CNN along with the MNIST dataset will give us the best results in recognizing handwritten digits. Therefore, we are going to implement this work using the same.

TECHNOLOGIES

Convolutional Neural Networks (CNN)

Convolutional Neural Networks are categorized as deep artificial neural networks. It is used for image recognition and also in object recognition. It has been used under various other applications in detection algorithms. CNN's core building block is the convolutional layer. This layer parameter is composed of kernels (also known as learnable filters) which have a not-so-large receptive field but extend through the full depth of the input volume. When the forward pass is applied, each filter is convolved across the width and height of the input and then computing the dot product and then developing a 2-D activation map of the corresponding filter. As a result, the network learns when they see certain types of features at a spatial location in the input. Activation maps are then given in the lower sample layer and, as a decision, this method is applied to a patch time. CNN has a fully connected layer, which classifies the output with a label per node**.**



CNN ARCHITECTURE

MNIST (Modified National Institute of Standards and Technology database)

It is a handwritten numeric dataset used to train different image processing models. It is mostly used for training and testing in the field of machine learning. With 60,000 training and 10,000 testing examples, it is quite a large dataset. All the images are of the fixed size of 28 * 28 pixels. For those who want to try learning methods and model recognition methods on real-world data with minimal effort on formatting and preprocessing MNIST is an ideal dataset. We will use this dataset for our research work.

We differ for various reasons in using this MNIST handwritten database. First, as mentioned above it is a relatively simple database for rapid testing theories and algorithms. We want to test neural networks that are applicable to practical problems in the real world, as the MNIST database already contains handwritten digits, including partitioning and generalization so that making a minimal effort for preprocessing and formatting is a good start for us. In addition, many researchers are evaluating their theories and algorithms using MNIST, which means that our results can be compared to a broader literature set.

MNIST DATASET

After we complete our task and build our model, we will compare our accuracy rate and how our model is performing with other existing models as well which are in the same domain.

Python

Python is an interpreted, object-oriented, high-level programming language. It avails rapid application development with its high-level built-in data structures that combine with dynamic binding and dynamic typing.

In our model, we have used python as it has a simple, easy-to-learn syntax that emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages encouraging program modularity and code reuse. It has an extensive library that is available in source or binary form for all major platforms. No special software and hardware requirements are needed for python to run smoothly.

Tensorflow

TensorFlow is an open-source platform for fast numerical computing. It's a comprehensive and variety of tools, libraries, and other resources that provide workflows with high-level APIs. Tensorflow's longing execution allows for immediate iteration along with instinctive debugging. It has multiple levels of abstraction to build and train models which provides an Easy Model Building. It is no language or platform-specific which allows the programmer to train and deploy their model easily. TensorFlow supports flexibility and control for different features like the Model Subclassing API and Keras Functional API, which creates complex topologies.

There are many different libraries available for Deep Learning, however, TensorFlow was designed to be used in both research and development and the production systems. It can run on a single CPU System along with mobile devices and in hundreds of machines with larger-scale distributed systems

OpenCV

OpenCV or Open-Source Computer Vision Library is a machine learning library for Computer Vision applications. Used to give common infrastructure to the application. With the increase of Computer Vision applications, it has gained much prominence among organizations and academia. It supports a wide variety of programming languages, for example, C++, Python, Java, etc., and is available on different platforms including Windows, Linux, OS X, Android, and iOS.

With operations like CUDA and OpenCL, it can easily access and manipulate them for high-speed GPUs. It combines the best qualities of the OpenCV C++ API and the Python language.

Numpy

NumPy stands for Numerical Python, it is a Python library used in the industry for array computing. It has a large number of mathematical, algebraic, and transformation functions for working in the domain of linear algebra, Fourier transforms, trigonometric, statistical, and matrices. It has a multi-dimensional array and matrix data structures. NumPy aims to provide an array object that is faster than Python lists. NumPy arrays are stored at one continuous place in memory, which allows the processes to easily access and manipulate them. This behaviour is called locality of reference, this the main reason NumPy is faster than python list and Why we have used it in our model

Matplotlib

Matplotlib is an open-source visualization utility and plotting library for Python and NumPy. It is used as an alternative to MATLAB. Mostly it is written in python, however, for some platform compatibility, a few segments are written in C, Objective-C, and Javascript. A Python matplotlib script is structured

to instantly generate a visual data plot that only requires a few lines of code. Its scripting layer overlays two APIs. The pyplot API is a hierarchy of Python code objects topped by matplotlib.py plot a collection of functions used to makes many change to a figure such as creating a figure, creating a plotting area in a figure, decorating a plot with labels, and plots some lines in a plotting area. The other pyplot API Object-Oriented API collection of objects, assembled with greater flexibility than pyplot. This API allows direct access to Matplotlib's backend layers. In this model, we have only used matplotlib.py

OS

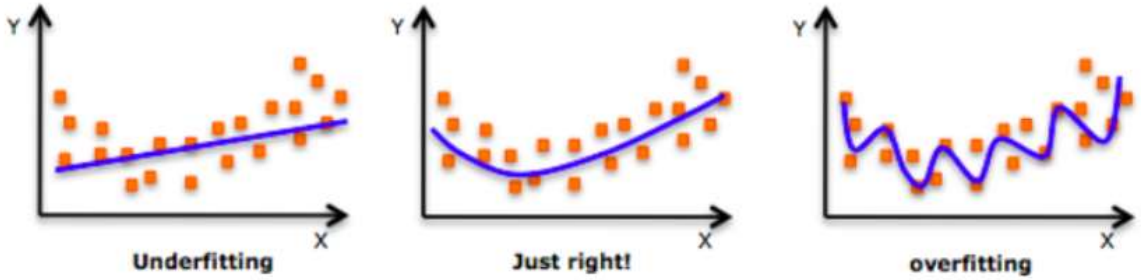
The OS module in Python provides the facility to establish the interaction with the operating system. This comes under Python's standard utility modules. It offers a suitable way of using operating system-dependent functionalities to create folders, remove folders, move folders, and sometimes change the working directory. OS allows us to work with the files and directories. For using the functions of this module it is necessary to import this module by using the "import os" statement.

Image Data Augmentation

Deep learning models have been largely attributed to the quantity and diversity of data which allows it to perform better with a huge dataset. Data augmentation is a technique that can be used by practitioners to significantly increase the diversity of data available for training models. This process artificially expands the available dataset by applying domain-specific techniques to the training data that create new and different training examples for training a deep learning model. In other words, it involves creating transformed versions of images that belong to the same class as the original image in the training dataset. Data augmentation makes the model more skillful with its techniques such as cropping, padding, and horizontal flipping which are used to train large neural networks. The augmentation techniques can create variations of the images that can improve the ability of the models to classify the data with help of what they have learned to new images.

Regularization

During the training of the convolutional neural networks, it is necessary to avoid overfitting. This is a situation where the model performed exceptionally well on training data but will give huge errors in predicting the target values for the test dataset. For overcoming this situation regularization is used.



REGULARIZATION TECHNIQUE(S)

The regularization technique makes slight modifications to the function by adding an additional penalty term in the error function. The additional term controls the unstable function which results in the coefficient's not considering extreme values. This improves the model's performance on the unseen data. In our model, we have used L1 and L2 regularization technique

5. RESULTS & ANALYSIS

Without image data augmentation

Case	Number of Dense Layers	Minimum Training Accuracy	Minimum Validation Accuracy	Maximum Training Accuracy	Maximum Validation Accuracy	Test Accuracy
		Accuracy(%)	Accuracy(%)	Accuracy(%)	Accuracy(%)	Accuracy(%)
1	3	99.46	97.83	99.85	98.61	98.61
2	4	88.00	95.02	99.70	98.60	98.69
3	3	89.13	95.14	99.62	98.49	98.17
4	4	85.24	99.51	94.82	98.26	98.40

With image data augmentation

Case	Number of Dense Layers	Minimum Training Accuracy(%)	Maximum Training Accuracy(%)	Test Accuracy
5	3	91.07	99.47	98.82
6	4	89.43	99.45	98.71
7	3	90.52	99.33	98.82
8	4	87.77	99.22	98.37

Case 1: In this experiment, we created a CNN model with 3 Convolutional layers and 3 dense layers where the 3rd dense layer is the output layer. The Convolution layers are the ones responsible for feature extraction and all these layers have rectified linear units (ReLU) as the activation function. We've used the flatten to convert the 2D filter matrix into a 1D feature vector before moving onto the fully connected layers. The output layer contains 10 neurons for 10 classes and determines the digits numbered from 0 to 9. A 'softmax' activation function is used with the output layer to output digits from 0 to 9. The Network is then fit over 20 epochs. The minimum training accuracy of 99.46% is found while the minimum validation accuracy of 97.83% is found. The maximum training accuracy of 99.85% is found while the maximum validation accuracy of 98.61% is found. The test accuracy for this case is 98.61%. This model does not have features like regularization and/or image data augmentation.

Case 2: In this case, we created a modification of the model in the first case. In this, we've increased the number of dense layers from 3 to 4. So this model contains 3 convolutional layers and 4 dense layers. The Network is then fit over 20 epochs. The minimum training accuracy of 88.00% is found while the minimum validation accuracy of 95.02% is found. The maximum training accuracy of 99.70% is found while the maximum validation accuracy of 98.60% is found. The test accuracy for this case is 98.69%.

Case 3: In this case, we modified the model in the first case by adding an L2 regularizer in the first layer. The L2 regularizer has a value of 0.01 and we've also used a bias regularizer with it which is also having a value of 0.01. So this model contains 3 convolutional layers and 3 dense layers with the first convolutional layer having L2 regularization. The Network is then fit over 20 epochs. The

minimum training accuracy of 89.13% is found while the minimum validation accuracy of 95.14% is found. The maximum training accuracy of 99.62% is found while the maximum validation accuracy of 98.49% is found. The test accuracy for this case is 98.17%. This model does not have image data augmentation.

Case 4: In this case, we modified the model in the second case by adding an L2 regularizer in the first layer. The L2 regularizer has a value of 0.01 and we've also used a bias regularizer with it which is also having a value of 0.01. So this model contains 3 convolutional layers and 4 dense layers with the first convolutional layer having L2 regularization. The Network is then fit over 20 epochs. The minimum training accuracy of 85.24% is found while the minimum validation accuracy of 99.51% is found. The maximum training accuracy of 94.82% is found while the maximum validation accuracy of 98.26% is found. The test accuracy for this case is 98.40%. This model does not have image data augmentation.

Case 5: In this case, we modified the model in the first case by using Image Data Augmentation. We've augmented all the images in the training set by giving a rotation range of 10 degrees. So this model contains 3 convolutional layers and 3 dense layers. The Network is then fit over 20 epochs. The minimum training accuracy of 91.07% is found. The maximum training accuracy of 99.47% is found. The test accuracy for this case is 98.82%. This model does not have regularization.

Case 6: In this case, we modified the model in the first case by using Image Data Augmentation. We've augmented all the images in the training set by giving a rotation range of 10 degrees. So this model contains 3 convolutional layers and 4 dense layers. The Network is then fit over 20 epochs. The minimum training accuracy of 89.43% is found. The maximum training accuracy of 99.45% is found. The test accuracy for this case is 98.71%. This model does not have regularization.

Case 7: In this case, we modified the model in the first case by using Image Data Augmentation and by adding an L2 regularizer in the first layer. The L2 regularizer has a value of 0.01 and we've also used a bias regularizer with it which is also having a value of 0.01. We've augmented all the images in the training set by giving a rotation range of 10 degrees. So this model contains 3 convolutional layers and 3 dense layers with the first convolutional layer having L2 regularization. The Network is then fit over 20 epochs. The minimum training accuracy of 90.52% is found. The maximum training accuracy of 99.33% is found. The test accuracy for this case is 98.82%.

Case 8: In this case, we modified the model in the first case by using Image Data Augmentation and by adding an L2 regularizer in the first layer. The L2 regularizer has a value of 0.01 and we've also used a bias regularizer with it which is also having a value of 0.01. We've augmented all the images

in the training set by giving a rotation range of 10 degrees. So this model contains 3 convolutional layers and 3 dense layers with the first convolutional layer having L2 regularization. The Network is then fit over 20 epochs. The minimum training accuracy of 87.77% is found. The maximum training accuracy of 99.22% is found. The test accuracy for this case is 98.37%.

6. Conclusion And Future Scope

In this project, we observed variations of accuracies for 20 epochs by varying the hidden layers and also toggling the regularization and image data augmentation. We carried out 8 experiments (Case1 to Case8) which are mentioned in the Results & Analysis section. These 8 cases are performing differently because of the various combinations of layers in the model and also toggling of regularization technique and image data augmentation. Our main aim was to recognize digits from images that can be coloured or augmented(rotated by a certain degree). We have achieved this in our last two case(s) [Case 7 & Case8] where we are recognizing digits from coloured images, black and white images and also we are handling image augmentation. In Case-7 we have built the model using 3 dense layers and so getting the maximum training accuracy as 99.33% and test accuracy as 98.82%. In Case-8 we have built the model using 4 dense layers and so getting the maximum training accuracy as 99.22% and test accuracy as 98.37%. This is the reason why our final model is going to be the one which is in Case-7 with 3 dense layers and applying both image data augmentation and regularization.

In the future, we are planning to observe the variation in overall classification accuracy by varying the number of hidden layers and also toggling with other parameters of the model. At present we have applied an image data augmentation technique that can handle rotated images that are rotated by 10 degrees. We plan to make our model more optimized in this direction so we can also make our model recognize the rotated 6 and 9 image(s). We also plan to achieve better accuracy results on the coloured images also as there can be various boundary conditions that may exist in coloured images. We will also try to implement our models on letters and double-digit(s) database so that we can extend our implementation and can diversify our model's application.

7. REFERENCES

[1] Improved Handwritten Digit Recognition Using Convolutional Neural Networks (CNN) Savita Ahlawat 1, Amit Choudhary 2, Anand Nayyar 3, Saurabh Singh 4 and Byungun Yoon 4.

[2] Classification Algorithms for Determining Handwritten Digit Hayder Naser Khraibet AL-Behadili.

[3] F. Siddique, S. Sakib and M. A. B. Siddique, "Recognition of Handwritten Digit using Convolutional Neural Network in Python with Tensorflow and Comparison of Performance for Various Hidden Layers," 2019 5th International Conference on Advances in Electrical Engineering (ICAEE), Dhaka, Bangladesh, 2019, pp. 541-546, doi: 10.1109/ICAEE48663.2019.8975496.

[4] Recognition of Handwritten Digit using Convolutional Neural Network (CNN) By Md. Anwar Hossain & Md. Mohon Ali

[5] Classification of MNIST Handwritten Digit Database using Neural Network Wan Zhu

[6] Shekhar and Kaushik; Handwritten Digit Recognition Using Computer Vision, 2020

[7] Breukelen, Duin, Tax and Hartog; Handwritten Digit Recognition by Combined Classifier, 1998

[8] Le Cun, Boser, Denker, Henderson, Howard, Hubbard and Jackel; Handwritten Digit Recognition with a Back-Propagation Network

[9] Cire¸san, Meier, Gambardella and Schmidhuber; Handwritten Digit Recognition with a Committee of Deep Neural Nets on GPUs, 2011

[10] Boukharouba and Bennia; Novel feature extraction technique for the recognition of handwritten digits, 2011

[11]LeCun, Jackel, Bottou*, Cortes, Denker, Drucker, Guyon, Muller, Sackinger, Simard, and Vapnik; Learning Algorithms For Classification: A Comparison On Handwritten Digit Recognition

[12]LeCun, Jackel, Bottou, Brunot, Cortes, Denker, Drucker, Guyon, Muller, Sackinger, Simard, and Vapnik; Comparison Of Learning Algorithms For Handwritten Digit Recognition

[13]Alwzwazy, Albehadili, Alwan, Islam; Handwritten Digit Recognition Using Convolutional Neural Networks, 2016

[14]Hallale, Salunke; Offline Handwritten Digit Recognition Using Neural Network, 2013

[15]Flora and Kakkad; Comparison Study of Handwritten Digit Recognition using Artificial Neural Network and Convolutional Neural Network: A Review, 2019

8. ANNEXURE

SOURCE CODE:

```
import tensorflow as tf
tf.keras.datasets.mnist.load_data(path="mnist.npz")
mnist = tf.keras.datasets.mnist
x_train.shape
import matplotlib.pyplot as plt
plt.imshow(x_train[2])
plt.show()
plt.imshow(x_train[2],cmap=plt.cm.binary)
print(x_train[2])
x_train = tf.keras.utils.normalize(x_train, axis=1)
x_test = tf.keras.utils.normalize(x_test, axis=1)
import matplotlib.pyplot as plt
plt.imshow(x_train[2])
plt.show()
plt.imshow(x_train[2],cmap=plt.cm.binary)
print(x_train[2])
print(y_train[2])
import numpy as np
IMG_SIZE = 28
x_trainr = np.array(x_train).reshape(-1, IMG_SIZE, IMG_SIZE, 1)
```

```
x_testr = np.array(x_test).reshape(-1, IMG_SIZE, IMG_SIZE, 1)
print("Training samples dimension: ",x_trainr.shape)
print("Testing samples dimension: ",x_testr.shape)
from tensorflow.keras.models import Sequential
from tensorflow.keras.layersimport
Dense,Dropout,Activation,Flatten,Conv2D,MaxPooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.optimizers import Adam, SGD
from tensorflow.keras.regularizers import l2

datagen = ImageDataGenerator(
    featurewise_center=False,  # set input mean to 0 over the dataset
    samplewise_center=False,  # set each sample mean to 0
    featurewise_std_normalization=False,  # divide inputs by std of
the dataset
    samplewise_std_normalization=False,  # divide each input by its
std
    zca_whitening=False,  # apply ZCA whitening
    rotation_range=10,  # randomly rotate images in the range
(degrees, 0 to 180)
#     width_shift_range=0.2,  # randomly shift images horizontally
(fraction of total width)
#     height_shift_range=0.2,  # randomly shift images vertically
(fraction of total height)
    horizontal_flip=False,  # randomly flip images
    vertical_flip=False)  # randomly flip images

datagen.fit(x_trainr)
model = Sequential()

model.add(Conv2D(64,(3,3),input_shape=x_trainr.shape[1:],kernel_regul
arizer=l2(0.01),bias_regularizer=l2(0.01)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64,(3,3)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64,(3,3)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())
```

```
model.add(Dense(64))
model.add(Activation("relu"))

model.add(Dense(32))
model.add(Activation("relu"))

model.add(Dense(10))
model.add(Activation("softmax"))
model.summary()
model.compile(loss="sparse_categorical_crossentropy",optimizer="adam"
,metrics=['accuracy'])
history=model.fit(datagen.flow(x_trainr,y_train),epochs=20)
print(min(history.history['accuracy']))

print(max(history.history['accuracy']))
test_loss, test_acc = model.evaluate(x_testr,y_test)
print("Test loss on 10,000 test samples: ",test_loss)
print("Validation accuracy on 10,000 test samples: ",test_acc)
prediction = model.predict([x_testr])
print(prediction)
#COLOUR
import matplotlib.pyplot as plt
from skimage import io
import cv2
from skimage import util
import os


for filename in os.listdir('colour'):
    if filename.endswith(".png") or filename.endswith(".jpg") or
filename.endswith(".jpeg"):

        pass
    else:
        continue
    name="colour/"
    name+=filename
    i=name



    img = cv2.imread(i)
    # plt.imshow(img) #ORIGINAL IMAGE
    img1 = cv2.imread(i,cv2.IMREAD_GRAYSCALE)
    average = img1.mean(axis=0).mean(axis=0)
```

```
#     print(average)
    # plt.imshow(img1) GRAYSCALED READ IMAGE
    thresh = average
    img_binary = cv2.threshold(img1, thresh, 255,
cv2.THRESH_BINARY)[1]

    cv2.imwrite('test_colour/test_c.png',img_binary)

    img2=cv2.imread('test_colour/test_c.png')
    # plt.imshow(img2) #BLACK AND WHITE
    # print(img2)
    unique_elements, counts_elements = np.unique(img2,
return_counts=True)
#     print("Frequency of unique values of the said array:")
#     print(np.asarray((unique_elements, counts_elements)))
    d=dict(zip(unique_elements, counts_elements))
#     print(d)

    if(d[0]<d[255]):
        img2 = util.invert(img2)
    #     print(img2)
    cv2.imwrite('test_colour/test_c.png',img2)



    gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

    resized2 = cv2.resize(gray2, (28,28),interpolation =
cv2.INTER_AREA)
    newimg2 = tf.keras.utils.normalize(resized2, axis=1)

    newimg2 = np.array(newimg2).reshape(-1, IMG_SIZE, IMG_SIZE, 1)
    predictions2 = model.predict(newimg2)
#     plt.subplot(1,15,images.index(i)+1)
    t="test_colour/"
    t+=filename
    cv2.imwrite(t,img2)
    plt.imshow(img2)

    print("PREDICTION: ",np.argmax(predictions2),"IMAGE NAME: ",i)

    # plt.imshow(resized2) #FINAL IMAGE
#BW
import matplotlib.pyplot as plt
from skimage import io
```

```
import cv2
from skimage import util
import os


for filename in os.listdir('bw'):
    if filename.endswith(".png") or filename.endswith(".jpg") or
filename.endswith(".jpeg"):

        pass
    else:
        continue
    name="bw/"
    name+=filename
    i=name


    img = cv2.imread(i)
    # plt.imshow(img) #ORIGINAL IMAGE
    img1 = cv2.imread(i,cv2.IMREAD_GRAYSCALE)
    average = img1.mean(axis=0).mean(axis=0)
#     print(average)
    # plt.imshow(img1) GRAYSCALED READ IMAGE
    thresh = average
    img_binary = cv2.threshold(img1, thresh, 255,
cv2.THRESH_BINARY)[1]

    cv2.imwrite('test_bw/test_b.png',img_binary)

    img2=cv2.imread('test_bw/test_b.png')
    # plt.imshow(img2) #BLACK AND WHITE
    # print(img2)
    unique_elements, counts_elements = np.unique(img2,
return_counts=True)
#     print("Frequency of unique values of the said array:")
#     print(np.asarray((unique_elements, counts_elements)))
    d=dict(zip(unique_elements, counts_elements))
#     print(d)

    if(d[0]<d[255]):
        img2 = util.invert(img2)
    #     print(img2)
    cv2.imwrite('test_bw/test_b.png',img2)
```

```
    gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

    resized2 = cv2.resize(gray2, (28,28),interpolation =
cv2.INTER_AREA)
    newimg2 = tf.keras.utils.normalize(resized2, axis=1)

    newimg2 = np.array(newimg2).reshape(-1, IMG_SIZE, IMG_SIZE, 1)
    predictions2 = model.predict(newimg2)
#     plt.subplot(1,15,images.index(i)+1)
    t="test_bw/"
    t+=filename
    cv2.imwrite(t,img2)
    plt.imshow(img2)

    print("PREDICTION: ",np.argmax(predictions2),"IMAGE NAME: ",i)

    # plt.imshow(resized2) #FINAL IMAGE
```

GitHub Repository: https://github.com/arjuns007/HandwrittenDigitClassification