

**Project Report**

**Handwritten Digit Recognition with Neural Network**

**Course Code**: CSE366

**Course Title:** Artificial Intelligence

**Section No**: 01

**Semester**: Summer 22

**Submitted By**

**Israfil Arman (ID: 2020-1-60-086)**

**Md. Habibuzzaman (2020-1-60-212)**

**Tanjilul Haq (ID: 2020-1-60-214)**

**Submitted To**

**Jesan Ahammed Ovi**

Senior Lecturer

Department of CSE

East West University

**Date of Submission**: 13-09-2022

**Problem Statement**

Handwritten digit recognition is the ability of a computer to recognize the human handwritten digits from different sources like images, papers, touch screens, etc., and classify them into 10 predefined classes (0-9). It is a hard task for the machine because handwritten digits are not perfect and can be made with many different shapes and sizes. In the project we built a Neural Network model and trained and tested the model with the MNIST database of handwritten digits. Moreover, we have also tested the model with images outside the database and found satisfactory results.

**System Requirements**

1. Processor: Intel(R) Core (TM) i3-1005G1 CPU @ 1.20GHz 1.20 GHz
2. RAM: 4GB
3. Operating System: Windows
4. IDE: Google Colaboratory

**System Design**

In the project we used **Adam** (*adaptive moment estimation)* optimization algorithm.

The Adam optimization algorithm is an extension to stochastic gradient descent that has recently seen broader adoption for deep learning applications in computer vision and natural language processing. Adam was presented by **DiederikKingma** from **OpenAI** and **Jimmy Ba** from the University of Toronto in their 2015 [ICLR](http://www.iclr.cc/doku.php?id=iclr2015:main) paper (poster) titled “[Adam: A Method for Stochastic Optimization](https://arxiv.org/abs/1412.6980)“. It can be used instead of the classical stochastic gradient descent procedure to update network weights iteratively based on training data.

Adam is different to classical stochastic gradient descent. Stochastic gradient descent maintains a single learning rate (termed alpha) for all weight updates and the learning rate does not change during training. But Adam’s learning rate is maintained for each network weight (parameter) and separately adapted as learning unfolds.

The authors describe Adam as combining the advantages of two other extensions of stochastic gradient descent. Specifically:

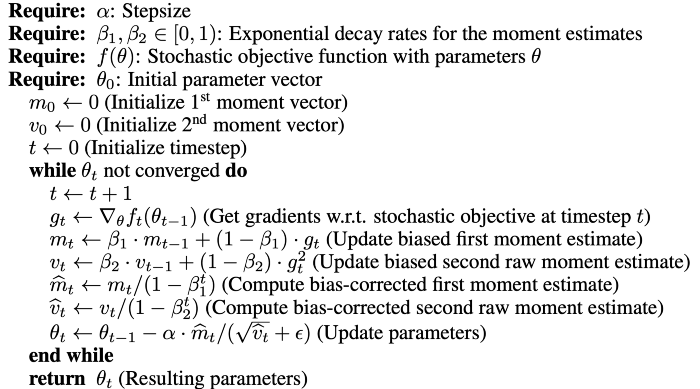
* **Adaptive Gradient Algorithm** (AdaGrad) that maintains a per-parameter learning rate that improves performance on problems with sparse gradients (e.g. natural language and computer vision problems).
* **Root Mean Square Propagation** (RMSProp) that also maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight (e.g. how quickly it is changing). This means the algorithm does well on online and non-stationary problems (e.g. noisy).

Adam realizes the benefits of both AdaGrad and RMSProp. Instead of adapting the parameter learning rates based on the average first moment (the mean) as in RMSProp, Adam also makes use of the average of the second moments of the gradients (the uncentered variance). Specifically, the algorithm calculates an exponential moving average of the gradient and the squared gradient, and the parameters beta1 and beta2 control the decay rates of these moving averages. The initial value of the moving averages and beta1 and beta2 values close to 1.0 (recommended) result in a bias of moment estimates towards zero. This bias is overcome by first calculating the biased estimates before then calculating bias-corrected estimates.

Adam configuration parameters:

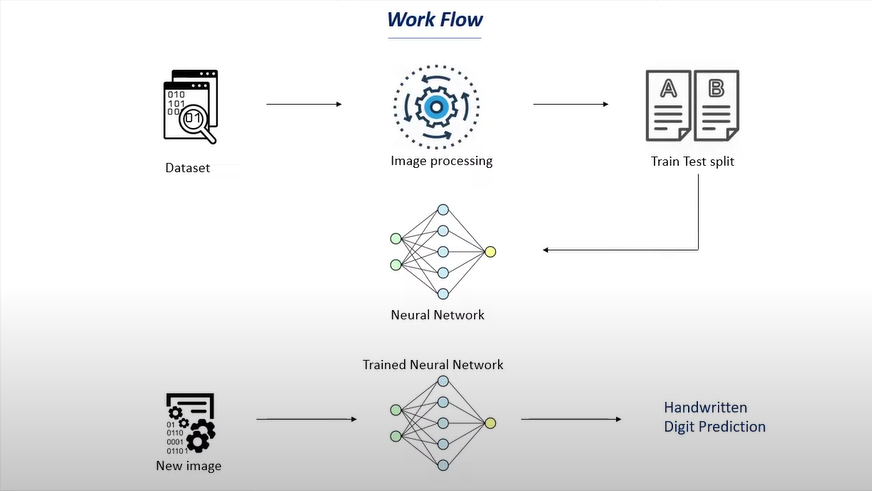
* **alpha**. Also referred to as the learning rate or step size. The proportion that weights are updated (e.g. 0.001). Larger values (e.g. 0.3) results in faster initial learning before the rate is updated. Smaller values (e.g. 1.0E-5) slow learning right down during training
* **beta1**. The exponential decay rate for the first moment estimates (e.g. 0.9).
* **beta2**. The exponential decay rate for the second-moment estimates (e.g. 0.999). This value should be set close to 1.0 on problems with a sparse gradient (e.g. NLP and computer vision problems).
* **epsilon**. Is a very small number to prevent any division by zero in the implementation (e.g. 10E-7).

**Algorithm:**



**Architecture of the project**:

First, we imported a dataset from Keras MNIST dataset for training our algorithm. Using some library, we processed the images from the dataset. Then we trained and tested the neural network. Finally, we inserted new handwritten digit image in a trained neural network and got the predicted handwritten digit as output.



**Implementation**

Importing the necessary libraries

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import cv2

from google.colab.patches import cv2\_imshow

from PIL import Image

import tensorflow as tf

tf.random.set\_seed(3)

from tensorflow import keras

from keras.datasets import mnist

from tensorflow.math import confusion\_matrix

**Loading the MNIST data from keras.datasets**

(X\_train, Y\_train), (X\_test, Y\_test) =  mnist.load\_data()

**Building the Neural Network**

#setting up the layers of the Neural Network

model = keras.Sequential([

               keras.layers.Flatten(input\_shape=(28,28)),#input layer

               keras.layers.Dense(80, activation='relu'),#hidden layer1

               keras.layers.Dense(80, activation='relu'), #hidden layer2

               keras.layers.Dense(10, activation='sigmoid')#output layer

                        ])

Here we have used ReLU (Rectified Linear Unit) activation function for our inner/hidden layers. The main advantage of using the ReLU function over other activation functions is that it does not activate all the neurons at the same time.

This means that the neurons will only be deactivated if the output of the linear transformation is less than 0.

Function: f(x) = max (0, x)

Moreover, the Sigmoid activation function for the output layer. It transforms the values between the range 0 and 1 as we have reshaped the trained and tested data between 0 and 1.

Function: f(x) = 1/(1+e^-x)

**Compiling the Neural Network**

model.compile(optimizer='adam',

              loss = 'sparse\_categorical\_crossentropy',

              metrics=['accuracy'])

**Training the Neural Network**

model.fit(X\_train, Y\_train, epochs=250)

**Checking Accuracy on Test data**

loss, accuracy = model.evaluate(X\_test, Y\_test)

print(accuracy)

**Creating a confusion matrix to evaluate the test data**

conf\_mat = confusion\_matrix(Y\_test, Y\_pred\_labels)

plt.figure(figsize=(15,7))

sns.heatmap(conf\_mat, annot=True, fmt='d', cmap='Reds')

plt.ylabel('True Labels')

plt.xlabel('Predicted Labels')

**Predictive System**

input\_image\_path = input ('Path of the image to be predicted: ')

input\_image = cv2.imread(input\_image\_path)

cv2\_imshow(input\_image)

input\_image = cv2.cvtColor(input\_image, cv2.COLOR\_RGB2GRAY)

input\_image\_resize = cv2.resize(input\_image, (28, 28))

input\_image\_resize = input\_image\_resize/255

image\_reshaped = np.reshape(input\_image\_resize, [1,28,28])

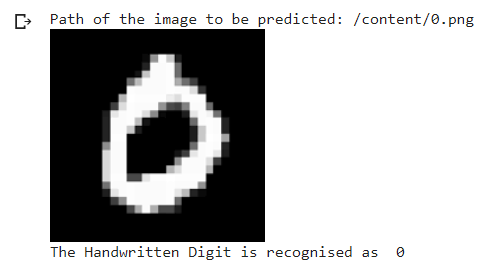
input\_prediction = model.predict(image\_reshaped)

input\_pred\_label = np.argmax(input\_prediction)

print ("The Handwritten Digit is recognised as ", input\_pred\_label)

Here, after taking the image path to be predicted, cv2 library is used to process for visualizing the image. Using cv2.cvtColor we converted the image color from RGB to grayscale. After resizing and reshaping, the image is sent as an input to the neural network for predicting. After prediction it takes the maximum prediction of that digit is taken and printed as the result.

**Testing Results:**

Graphical user interface, text, application

Description automatically generated

Graphical user interface, application

Description automatically generatedGraphical user interface, application, qr code

Description automatically generated

We have uploaded the image file in the project directory and copied and given the path into the program. After execution the trained neural network successfully recognized the image.

**Future Scope:**

Limitations:

* We must use the same image background for prediction as the training dataset.
* This model cannot recognize RGB images.

Future Scope:

* We will try to detect handwritten words in the future.