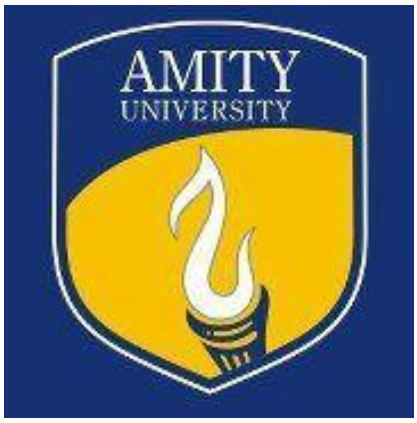
‘ARTIFICIAL INTELLIGENCE’

OPEN ENDED EXPERIMENT ON

“DEVELOPING A CNN FOR MNIST HANDWRITTEN DIGIT CLASSIFICATION”

Submitted to

AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY



Submitted by:-

ANURAG TRIPATHI (A12405117004)

Submitted to:-

Dr. PARATHA SARATHI MANGIPUDI

DEPARTMENT OF ELECTRONICS and COMMUNICATION ENGG.

AMITY SCHOOL OF ENGINEERING and TECHNOLOGY

AMITY UNIVERSITY, UTTAR PRADESH

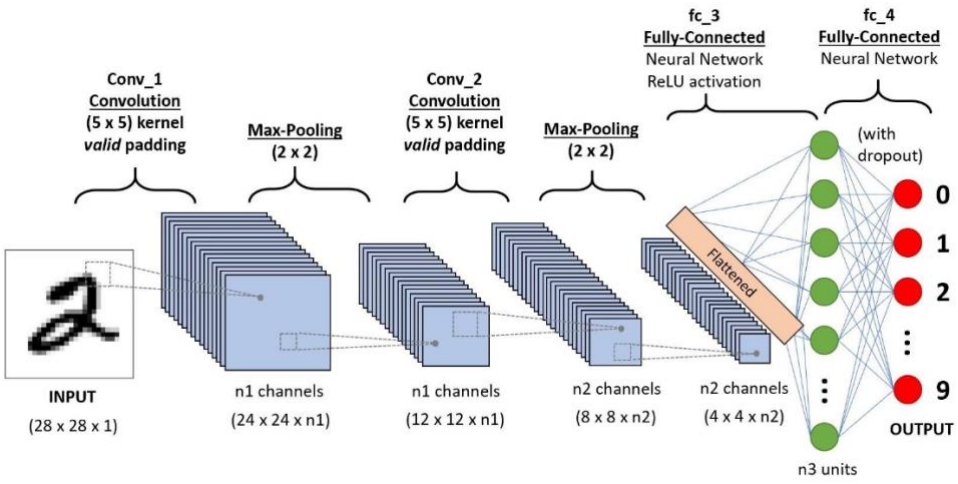
NOIDA (UP)

TABLE OF CONTENT-

1. CONVOLUTIONAL NEURAL NETWORK(CNN)

Artificial Intelligence has been witnessing a monumental growth in bridging the gap between the capabilities of humans and machines. Researchers and enthusiasts alike, work on numerous aspects of the field to make amazing things happen. One of many such areas is the domain of Computer Vision.

The agenda for this field is to enable machines to view the world as humans do, perceive it in a similar manner and even use the knowledge for a multitude of tasks such as Image & Video recognition, Image Analysis & Classification etc. The advancements in Computer Vision with Deep Learning has been constructed and perfected with time, primarily over one particular algorithm — a **Convolutional Neural Network**.



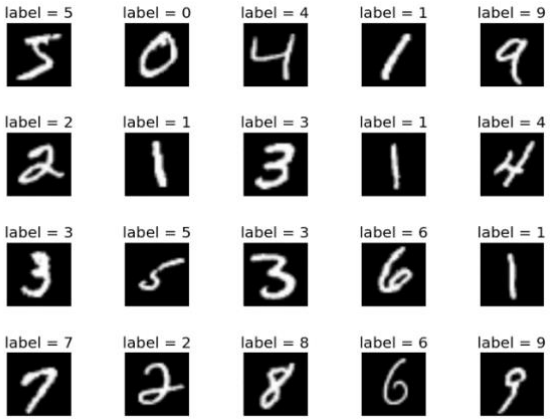
A CNN sequence to classify handwritten digits

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the

visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

2.MIXED NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (MNIST)



Sample image for MNIST test datasheet

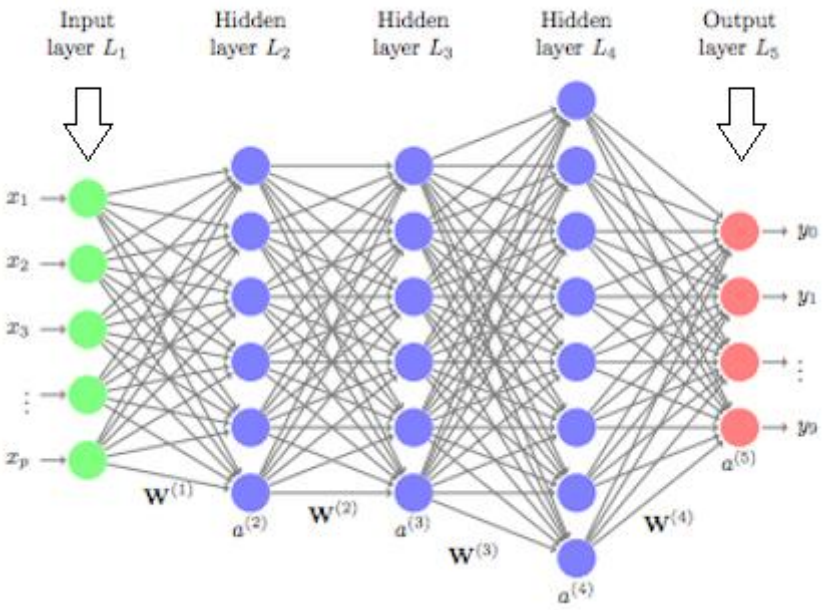
The MNIST database is a large database of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning. It was created by "re-mixing" the samples from NSITs original dataset. The creators felt that since NIST's training dataset was taken from American Census Bureau employees, while the testing dataset was taken from American high school students, it was not well-suited for machine learning experiments. Furthermore, the black and white images from NIST were normalized to fit into a 28x28 pixel bounding box and anti-aliased, which introduced grayscale levels.

The MNIST database contains 60,000 training images and 10,000 testing images. Half of the training set and half of the test set were taken from NIST's training dataset, while the other half of the training set and the other half of the test set were taken from NIST's testing dataset. There have been a number of scientific papers on attempts to achieve the lowest error rate; one paper, using a hierarchical system of convolutional neual networks, manages to get an error-rate on the MNIST database of 0.23%. The original creators of the database keep a list of some of the methods tested on it. In their original paper, they use a support-vector machine to get an error rate of 0.8%. An extended dataset similar to MNIST called EMNIST has been published in 2017, which contains 240,000 training images, and 40,000 testing images of handwritten digits and characters

3. DEEP LEARNING

Deep learning is an artificial intelligence function that imitates the workings of the human brain in processing data and creating patterns for use in decision making. Deep learning is a subset of machine learning in artificial intelligence (AI) that has networks capable of learning unsupervised from data that is unstructured or unlabeled. Also known as deep neural learning or deep neural network.

Deep learning has evolved hand-in-hand with the digital era, which has brought about an explosion of data in all forms and from every region of the world. This data, known simply as big data, is drawn from sources like social media, internet search engines, e-commerce platforms, and online cinemas, among others. This enormous amount of data is readily accessible and can be shared through fintech applications like cloud computing.



Deep learning learns from vast amounts of unstructured data that could normally take humans decades to understand and process.

4. IMAGE CLASSIFICATION

Image classification refers to a process in computer vision that can classify an image according to its visual content. For example, an image classification algorithm may be designed to tell if an image contains a human figure or not. While detecting an object is trivial for humans, robust image classification is still a challenge in computer vision application.

5.MNIST handwritten digit classification dataset

It is a dataset of 60,000 small square 28×28 pixel grayscale images of handwritten single digits between 0 and 9.

The task is to classify a given image of a handwritten digit into one of 10 classes representing integer values from 0 to 9, inclusively.

It is a widely used and deeply understood dataset and, for the most part, is "*solved*." Top-performing models are deep learning convolutional neural networks that achieve a classification accuracy of above 99%, with an error rate between 0.4 %and 0.2% on the hold out test dataset.

6. Steps to Develop a Baseline Model

6.1. Load Data Set

We can load the images and reshape the data arrays to have a single color channel and use a one hot encoding for the class element of each sample, transforming the integer into a 10 element binary vector with a 1 for the index of the class value, and 0 values for all other classes.

```
# load train and test dataset
def load_dataset():
    # load dataset
    (trainX, trainY), (testX, testY) = mnist.load_data()
    # reshape dataset to have a single channel
    trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
    testX = testX.reshape((testX.shape[0], 28, 28, 1))
    # one hot encode target values
    trainY = to_categorical(trainY)
    testY = to_categorical(testY)
    return trainX, trainY, testX, testY
```

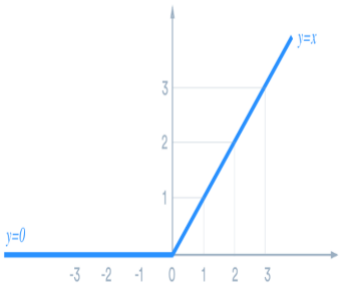
2. Prepare Pixel Data

The pixel values for each image in the dataset are unsigned integers in the range between black and white, or 0 and 255 and we some scaling will be required. A good starting point is to normalize the pixel values of grayscale images, e.g. rescale them to the range [0,1]. This involves first converting the data type from unsigned integers to floats, then dividing the pixel values by the maximum value. The *prep_pixels()* function below implements these behaviors and is provided with the pixel values for both the train and test datasets that will need to be scaled.

```python
# scale pixels
def prep_pixels(train, test):
    # convert from integers to floats
    train_norm = train.astype('float32')
    test_norm = test.astype('float32')
    # normalize to range 0-1
    train_norm = train_norm / 255.0
    test_norm = test_norm / 255.0
    # return normalized images
    return train_norm, test_norm
```

3. Define Model

Then comes the next task to define the model. All layers will use the ReLU activation function and the He weight initialization scheme, both best practices. ReLU stands for Rectified Linear Unit, and is a type of activation function. Mathematically, it is defined as $y = max(0, x)$.




```python
# define cnn model
def define_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(10, activation='softmax'))
    # compile model
    opt = SGD(lr=0.01, momentum=0.9)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

4. Evaluate Model

After the model is defined, we need to evaluate it.

```python
# evaluate a model using k-fold cross-validation
def evaluate_model(model, dataX, dataY, n_folds=5):
    scores, histories = list(), list()
    # prepare cross validation
    kfold = KFold(n_folds, shuffle=True, random_state=1)
    # enumerate splits
    for train_ix, test_ix in kfold.split(dataX):
        # select rows for train and test
        trainX, trainY, testX, testY = dataX[train_ix], dataY[train_ix], dataX[test_ix], dataY[test_ix]
        # fit model
        history = model.fit(trainX, trainY, epochs=10, batch_size=32, validation_data=(testX, testY), verbose=0)
        # evaluate model
        _, acc = model.evaluate(testX, testY, verbose=0)
        print('> %.3f' % (acc * 100.0))
        # stores scores
        scores.append(acc)
        histories.append(history)
    return scores, histories
```

7. COMPLETE RESULT

```
# baseline cnn model for mnist
From numpy import mean
From numpy import std
From matplotlib import pyplot
From sklearn.model_selection import KFold
From keras.datasets import mnist
From keras.utils import to_categorical
From keras.models import Sequential
From keras.layers import Conv2D
From keras.layers import MaxPooling2D
From keras.layers import Dense
From keras.layers import Flatten
From keras.optimizers import SGD


# load train and test dataset
Def load_dataset():
        # load dataset
        (trainX, trainY), (testX, testY) = mnist.load_data()
        # reshape dataset to have a single channel
```

```
        trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
        testX = testX.reshape((testX.shape[0], 28, 28, 1))
        # one hot encode target values
        trainY = to_categorical(trainY)
        testY = to_categorical(testY)
        return trainX, trainY, testX, testY


# scale pixels
Def prep_pixels(train, test):
        # convert from integers to floats
        Train_norm = train.astype(‘float32’)
        Test_norm = test.astype(‘float32’)
        # normalize to range 0-1
        Train_norm = train_norm / 255.0
        Test_norm = test_norm / 255.0
        # return normalized images
        Return train_norm, test_norm


# define cnn model
Def define_model():
        Model = Sequential()
```

```
        Model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))

        Model.add(MaxPooling2D((2, 2)))

        Model.add(Flatten())

        Model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))

        Model.add(Dense(10, activation='softmax'))

        # compile model

        Opt = SGD(lr=0.01, momentum=0.9)

        Model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])

        Return model


# evaluate a model using k-fold cross-validation

Def evaluate_model(model, dataX, dataY, n_folds=5):

        Scores, histories = list(), list()

        # prepare cross validation

        Kfold = KFold(n_folds, shuffle=True, random_state=1)

        # enumerate splits

        For train_ix, test_ix in kfold.split(dataX):

                # select rows for train and test

                trainX, trainY, testX, testY = dataX[train_ix], dataY[train_ix], dataX[test_ix], dataY[test_ix]

                # fit model
```

```
            History = model.fit(trainX, trainY, epochs=10, batch_size=32, validation_data=(testX, testY), verbose=0)

            # evaluate model

            _, acc = model.evaluate(testX, testY, verbose=0)

            Print('> %.3f' % (acc * 100.0))

            # stores scores

            Scores.append(acc)

            Histories.append(history)

        Return scores, histories


# plot diagnostic learning curves

Def summarize_diagnostics(histories):

        For i in range(len(histories)):

            # plot loss

            Pyplot.subplot(211)

            Pyplot.title('Cross Entropy Loss')

            Pyplot.plot(histories[i].history['loss'], color='blue', label='train')

            Pyplot.plot(histories[i].history['val_loss'], color='orange', label='test')

            # plot accuracy

            Pyplot.subplot(212)

            Pyplot.title('Classification Accuracy')
```

```
		Pyplot.plot(histories[i].history['acc'], color='blue', label='train')
		Pyplot.plot(histories[i].history['val_acc'], color='orange', label='test')
	Pyplot.show()

# summarize model performance
Def summarize_performance(scores):
	# print summary
	Print('Accuracy: mean=%.3f std=%.3f, n=%d' % (mean(scores)*100, std(scores)*100, len(scores)))
	# box and whisker plots of results
	Pyplot.boxplot(scores)
	Pyplot.show()

# run the test harness for evaluating a model
Def run_test_harness():
	# load dataset
	trainX, trainY, testX, testY = load_dataset()
	# prepare pixel data
	trainX, testX = prep_pixels(trainX, testX)
	# define model
	Model = define_model()
	# evaluate model
```

```
        Scores, histories = evaluate_model(model, trainX, trainY)
        # learning curves
        Summarize_diagnostics(histories)
        # summarize estimated performance
        Summarize_performance(scores)

# entry point, run the test harness
Run_test_harness()# baseline cnn model for mnist
From numpy import mean
From numpy import std
From matplotlib import pyplot
From sklearn.model_selection import KFold
From keras.datasets import mnist
From keras.utils import to_categorical
From keras.models import Sequential
From keras.layers import Conv2D
From keras.layers import MaxPooling2D
From keras.layers import Dense
From keras.layers import Flatten
From keras.optimizers import SGD
```

```
# load train and test dataset
Def load_dataset():
    # load dataset
    (trainX, trainY), (testX, testY) = mnist.load_data()
    # reshape dataset to have a single channel
    trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
    testX = testX.reshape((testX.shape[0], 28, 28, 1))
    # one hot encode target values
    trainY = to_categorical(trainY)
    testY = to_categorical(testY)
    return trainX, trainY, testX, testY


# scale pixels
Def prep_pixels(train, test):
    # convert from integers to floats
    Train_norm = train.astype('float32')
    Test_norm = test.astype('float32')
    # normalize to range 0-1
    Train_norm = train_norm / 255.0
    Test_norm = test_norm / 255.0
    # return normalized images
    Return train_norm, test_norm
```

```
# define cnn model

Def define_model():

        Model = Sequential()

        Model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))

        Model.add(MaxPooling2D((2, 2)))

        Model.add(Flatten())

        Model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))

        Model.add(Dense(10, activation='softmax'))

        # compile model

        Opt = SGD(lr=0.01, momentum=0.9)

        Model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])

        Return model


# evaluate a model using k-fold cross-validation

Def evaluate_model(model, dataX, dataY, n_folds=5):

        Scores, histories = list(), list()

        # prepare cross validation

        Kfold = KFold(n_folds, shuffle=True, random_state=1)

        # enumerate splits
```

```
        For train_ix, test_ix in kfold.split(dataX):
            # select rows for train and test
            trainX, trainY, testX, testY = dataX[train_ix], dataY[train_ix], dataX[test_ix], dataY[test_ix]
            # fit model
            History = model.fit(trainX, trainY, epochs=10, batch_size=32, validation_data=(testX, testY), verbose=0)
            # evaluate model
            _, acc = model.evaluate(testX, testY, verbose=0)
            Print('> %.3f' % (acc * 100.0))
            # stores scores
            Scores.append(acc)
            Histories.append(history)
        Return scores, histories


# plot diagnostic learning curves
Def summarize_diagnostics(histories):
        For i in range(len(histories)):
            # plot loss
            Pyplot.subplot(211)
            Pyplot.title('Cross Entropy Loss')
            Pyplot.plot(histories[i].history['loss'], color='blue', label='train')
```

```
            Pyplot.plot(histories[i].history['val_loss'], color='orange', label='test')

            # plot accuracy

            Pyplot.subplot(212)

            Pyplot.title('Classification Accuracy')

            Pyplot.plot(histories[i].history['acc'], color='blue', label='train')

            Pyplot.plot(histories[i].history['val_acc'], color='orange', label='test')

        Pyplot.show()


# summarize model performance

Def summarize_performance(scores):

        # print summary

        Print('Accuracy: mean=%.3f std=%.3f, n=%d' % (mean(scores)*100, std(scores)*100, len(scores)))

        # box and whisker plots of results

        Pyplot.boxplot(scores)

        Pyplot.show()


# run the test harness for evaluating a model

Def run_test_harness():

        # load dataset

        trainX, trainY, testX, testY = load_dataset()
```

```
        # prepare pixel data
        trainX, testX = prep_pixels(trainX, testX)
        # define model
        Model = define_model()
        # evaluate model
        Scores, histories = evaluate_model(model, trainX, trainY)
        # learning curves
        Summarize_diagnostics(histories)
        # summarize estimated performance
        Summarize_performance(scores)

# entry point, run the test harness
Run_test_harness()
```