Aim –

Build a Multiclass classifier using the CNN model. Use MNIST or any other suitable dataset.

a. Perform Data Pre-processing

b. Define Model and perform training

c. Evaluate Results using confusion matrix. Mapping with Syllabus - Unit 3

Objective - Building a multiclass classifier using a Convolutional Neural Network (CNN) using MNIST or any other suitable dataset. It involves several steps, including data pre-processing, defining the model architecture, training the model, and evaluating its performance using a confusion matrix

Software Requirements -

1)Python (3.x recommended)

2) TensorFlow (Deep learning framework for building CNNs)

3) Jupyter Notebook, any Python IDE, or Google Colab (for running Python code)

Hardware Requirements -

1) A machine with at least 8GB of RAM is recommended for model training.

2) A multi-core CPU is suitable, and for faster training, a GPU (Graphics Processing Unit) ishighly recommended.

Dataset – https://github.com/AmritK10/MNIST-CNN

Libraries or Modules Used – 1) Numpy - for linear algebra.

2) Pandas - for data analysis.

3) Matplotlib - for data visualization.

4) Tensorflow - for neural networks.

Theory –

ANN or Artificial Neural Network is a multi-layer fully-connected neural net that consists of many layers, including an input layer, multiple hidden layers, and an output layer. This is a very popular deep learning algorithm used in various classification tasks like audio and words. Similarly, we have Convolutional Neural Networks(CNNs) for image classification.

CNN is basically a model known to be Convolutional Neural Network and in recent times it has gained a lot of popularity because of its usefulness. CNN uses multilayer perceptrons to do computational works. CNN uses relatively little pre-processing compared to other image classification algorithms. This means the network learns through filters that in traditional algorithms were hand-engineered. So, for the image processing tasks CNNs are the best- suited option. Applying a Convolutional Neural Network (CNN) on the MNIST dataset is a popular way to learn about and demonstrate the capabilities of CNNs for image classification tasks. The

MNIST dataset consists of 28×28 grayscale images of hand-written digits (0-9), with a training set of 60,000 examples and a test set of 10,000 examples. Here is a basic approach to applying a CNN on the MNIST dataset using the Python

programming language and the Keras library:

1. Load and preprocess the data: The MNIST dataset can be loaded using the Keras library, and the images can be normalized to have pixel values between 0 and 1.

2. Define the model architecture: The CNN can be constructed using the Keras Sequential API, which allows for easy building of sequential models layer-by-layer. The architecture should typically include convolutional layers, pooling layers, and fully-connected layers.

3. Compile the model: The model needs to be compiled with a loss function, an optimizer, and a metric for evaluation.

4. Train the model: The model can be trained on the training set using the Keras fit() function. It is important to monitor the training accuracy and loss to ensure the model is converging properly.

5. Evaluate the model: The trained model can be evaluated on the test set using the Keras evaluate() function. The evaluation metric typically used for classification tasks is accuracy.

Algorithm –

1. Import Libraries: - Import necessary libraries, including TensorFlow and Keras.

2. Load and Pre-process the MNIST Dataset: - Load the MNIST dataset, which consists of 28x28 grayscale images of handwritten digits (0through 9). - Pre-process the data by normalizing pixel values (between 0 and 1), reshaping images, andone-hot encoding labels.

3. Define CNN Model Architecture: - Design the CNN architecture with convolutional layers, pooling layers, and fully connectedlayers. - Use activation functions like ReLU to introduce non-linearity. - The final layer has 10 units with softmax activation for multiclass classification.

4. Compile the Model: - Specify the optimizer (e.g., 'adam'), loss function (e.g., 'categorical_crossentropy' formulticlass classification), and evaluation metric (e.g., 'accuracy').

5. Train the Model: - Train the CNN using the training dataset. - Specify the number of epochs (passes through the entire dataset) and batch size.

6. Evaluate the Model: - Evaluate the trained CNN on the test dataset to assess its performance. - Measure metrics such as accuracy to understand how well the model generalizes to unseen data.

These steps provide a comprehensive overview of the process involved in building and training a CNN for image classification using the MNIST dataset. Adjustments to these steps can be made based on specific model requirements and task objectives.

Application –

1. Handwritten Digit Recognition: Recognizes handwritten digits (0-9) with applications in automated

2. Automatic Check Processing: Processes checks by recognizing handwritten amounts and account numbers.

3. Postal Code Recognition: Recognizes postal codes on envelopes for automated mail sorting.

4. Document Classification:

Code –

```
import numpy as np

import keras

from keras.datasets import mnist

from keras.models import Model

from keras.layers import Dense, Input

from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten

from keras import backend as k

(x_train, y_train), (x_test, y_test) = mnist.load_data()

img_rows, img_cols=28, 28

if k.image_data_format() == 'channels_first':
```

```
x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)

x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)

inpx = (1, img_rows, img_cols)

else:

x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)

x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)

inpx = (img_rows, img_cols, 1)

x_train = x_train.astype('float32')

x_test = x_test.astype('float32')

x_train /= 255

x_test /= 255

y_train = keras.utils.to_categorical(y_train)

y_test = keras.utils.to_categorical(y_test)

inpx = Input(shape=inpx)

layer1 = Conv2D(32, kernel_size=(3, 3), activation='relu')(inpx)

layer2 = Conv2D(64, (3, 3), activation='relu')(layer1)

layer3 = MaxPooling2D(pool_size=(3, 3))(layer2)

layer4 = Dropout(0.5)(layer3)

layer5 = Flatten()(layer4)

layer6 = Dense(250, activation='sigmoid')(layer5)

layer7 = Dense(10, activation='softmax')(layer6)

model = Model([inpx], layer7)

model.compile(optimizer=keras.optimizers.Adadelta(),
loss=keras.losses.categorical_crossentropy, metrics=['accuracy'])
```

```
model.fit(x_train, y_train, epochs=12, batch_size=500)

score = model.evaluate(x_test, y_test, verbose=0)

int('loss=', score[0])

print('accuracy=', score[1])
```