```
# Train the model
model.fit(X_train, y_train)

# Make predictions on the test set y_pred =
model.predict(X_test)

# Evaluate the model performance
mse = mean_squared_error(y_test, y_pred) r2 =
r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
```

Output -

```
Avg. Area Income Avg. Area House Age Avg. Area Number of Rooms \
  0 79545.458574          5.682861                7.009188
  1 79248.642455          6.002900                6.730821
  2 61287.067179          5.865890                8.512727
  3 63345.240046          7.188236                5.586729
  4 59982.197226          5.040555                7.839388

   Avg. Area Number of Bedrooms  Area Population         Price  \
  0 4.09                         23086.800503  1.059034e+06
  1 3.09                         40173.072174  1.505891e+06
  2 5.13                         36882.159400  1.058988e+06
  3 3.26                         34310.242831  1.260617e+06
  4 4.23                         26354.109472  6.309435e+05

Address
  0  208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
  1  188 Johnson Views Suite 079\nLake Kathleen, CA...
  2  9127 Elizabeth Stravenue\nDanieltown, WI 06482...
  3                    USS Barnett\nFPO AP 44820
  4                   USNS Raymond\nFPO AE 09386
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999 Data
columns (total 7 columns):
#   Column                       Non-Null Count  Dtype
---  ---------                    --------------------  -------
  0  Avg. Area Income             5000 non-null   float64
  1  Avg. Area House Age          5000 non-null   float64
  2  Avg. Area Number of Rooms    5000 non-null   float64
  3  Avg. Area Number of Bedrooms 5000 non-null   float64
  4  Area Population              5000 non-null   float64
  5  Price                        5000 non-null   float64
  6  Address                      5000 non-null   object
 dtypes: float64(6), object(1)
memory usage: 273.6+ KB None
Mean Squared Error: 10089009300.894518 R-
```

squared: 0.9179971706834289

References – https://en.wikipedia.org/wiki/Linear_regression

https://www.khanacademy.org/math/statistics-probability/describing-relationships-quantitative-data/introduction-to-trend-lines/a/linear-regression-review

https://github.com/huzaifsayed/Linear-Regression-Model-for-House-Price-Prediction/blob/master/linear-regression-model.jpg

Conclusion: Thus Implemented a Linear Regression Model to predict house prices for regions in the USA using the provided dataset.

ASSIGNMENT No: 02

Title:- 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.

Outcome -

Implement the technique of Convolution neural network (CNN)

Software Requirements -

- Python (3.x recommended)
- TensorFlow (Deep learning framework for building CNNs)
- Jupyter Notebook, any Python IDE, or Google Colab (for running Python code)

Hardware Requirements -

- A machine with at least 8GB of RAM is recommended for model training.
- A multi-core CPU is suitable, and for faster training, a GPU (Graphics Processing Unit) is highly recommended.

Prerequisites -

- Basic understanding of Python programming
- Familiarity with the concepts of Neural Networks, especially Convolutional Neural Networks (CNNs)

Dataset –

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

Libraries or Modules Used –

- Numpy - for linear algebra.
- Pandas - for data analysis.
- Matplotlib - for data visualization.
- 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

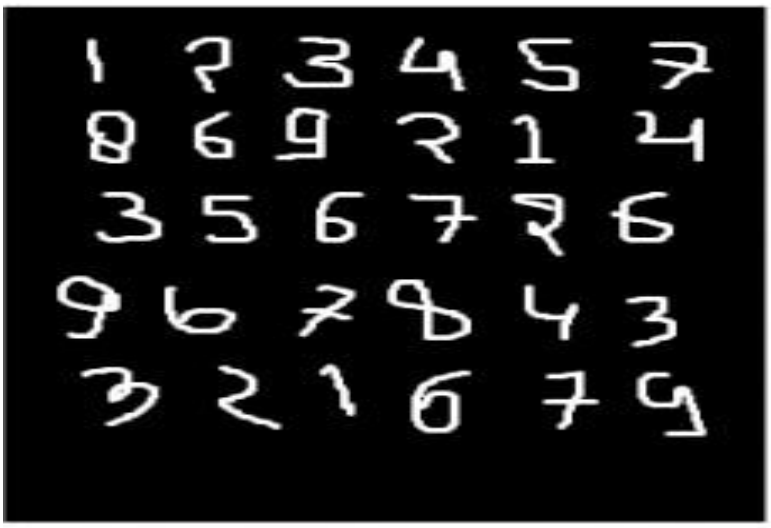
MNIST dataset consists of 28×28 grayscale images of hand-written digits (0-9), with a training set of 60,000 examples and a test set of 10,000 examples.

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.

MNIST dataset:

mnist dataset is a dataset of handwritten images as shown below in the image



We can get 99.06% accuracy by using CNN(Convolutional Neural Network) with a functional model. The reason for using a functional model is to maintain easiness while connecting the layers.

Firstly, include all necessary libraries

Python3:

```
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
```

Create the train data and test data

- **Test data:** Used for testing the model that how our model has been trained. Used to
  Train data: train our model.

Python3:

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

While proceeding further, **img_rows** and **img_cols** are used as the image dimensions. In mnis dataset, it is 28 and 28. We also need to check the data format i.e. 'channels_first' or 'channels_last'. In CNN, we can normalize data before hands such that large terms of the calculations can be reduced to smaller terms. Like, we can normalize the x_train and x_test data by dividing it by 255.

Checking data-format:

Python3:

```
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
```

Since the output of the model can comprise any of the digits between 0 to 9. so, we need 10 classes in output. To make output for 10 classes, use keras.utils.to_categorical function, which will provide the 10 columns. Out of these 10 columns, only one value will be one and the rest 9 will be zero and this one value of the output will denote the class of the digit.

Description of the output classes:

Python3:

```
y_train = keras.utils.to_categorical(y_train) y_test =
keras.utils.to_categorical(y_test)
```

- Now, the dataset is ready so let's move towards the CNN model :

Python3:

```
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)
```

- Explanation of the working of each layer in the CNN model:

layer1 is the Conv2d layer which convolves the image using 32 filters each of size (3*3). layer2 is again a Conv2D layer which is also used to convolve the image and is using 64 filters each of size (3*3).
layer3 is the MaxPooling2D layer which picks the max value out of a matrix of size (3*3).
layer4 is showing Dropout at a rate of 0.5.
layer5 is flattening the output obtained from layer4 and this flattens output is passed to layer6.
layer6 is a hidden layer of a neural network containing 250 neurons.
layer7 is the output layer having 10 neurons for 10 classes of output that is using the softmax function.

- Calling compile and fit function:

Python3:

```
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)
```

```
Epoch 1/12
60000/60000 [==============================] - 968s 16ms/step - loss: 0.7357 - acc: 0.7749
Epoch 2/12
60000/60000 [==============================] - 955s 16ms/step - loss: 0.2087 - acc: 0.9413
Epoch 3/12
60000/60000 [==============================] - 968s 16ms/step - loss: 0.1287 - acc: 0.9631
Epoch 4/12
60000/60000 [==============================] - 968s 16ms/step - loss: 0.0948 - acc: 0.9728
Epoch 5/12
60000/60000 [==============================] - 956s 16ms/step - loss: 0.0780 - acc: 0.9774
Epoch 6/12
60000/60000 [==============================] - 915s 15ms/step - loss: 0.0655 - acc: 0.9807
Epoch 7/12
60000/60000 [==============================] - 907s 15ms/step - loss: 0.0575 - acc: 0.9829
Epoch 8/12
60000/60000 [==============================] - 914s 15ms/step - loss: 0.0498 - acc: 0.9852
Epoch 9/12
60000/60000 [==============================] - 917s 15ms/step - loss: 0.0468 - acc: 0.9861
Epoch 10/12
60000/60000 [==============================] - 912s 15ms/step - loss: 0.0420 - acc: 0.9873
Epoch 11/12
60000/60000 [==============================] - 967s 16ms/step - loss: 0.0405 - acc: 0.9880
Epoch 12/12
60000/60000 [==============================] - 993s 17ms/step - loss: 0.0371 - acc: 0.9888

<keras.callbacks.History at 0x21ce04bb6a0>
```

- Firstly, we made an object of the model as shown in the above-given lines, where [inpx] is the input in the model and layer7 is the output of the model. We compiled the model using the required optimizer, loss function and printed the accuracy and at the last model.fit was called along with parameters like x_train(means image vectors), y_train(means the label), number of epochs, and the batch size. Using fit function x_train, y_train dataset is fed to model in particular batch size.

Evaluate function:

model.evaluate provides the score for the test data i.e. provided the test data to the model. Now, the model will predict the class of the data, and the predicted class will be matched with the y_test label to give us the accuracy.

Python3:

```
score = model.evaluate(x_test, y_test, verbose=0) int('loss=',
score[0])
print('accuracy=', score[1])
```

Output:

loss= 0.0295960184669

accuracy = 0.991

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 (0 through 9).
   - Pre-process the data by normalizing pixel values (between 0 and 1), reshaping images, and one-hot encoding labels.

3. Define CNN Model Architecture:
   - Design the CNN architecture with convolutional layers, pooling layers, and fully connected layers.
   - 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' for multiclass 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 systems.
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:

- Classifies documents based on handwritten patterns or characters.

5. **Medical Imaging:**
   - Analyzes medical images for detecting anomalies or identifying patterns.
6. **Character Recognition in Forms:**
   - Recognizes handwritten characters in forms for efficient data entry :
7. **Gesture Recognition:**
   - Recognizes hand gestures for sign language translation or device control :
8. **Product Label Recognition:**
   - Reads and interprets product labels for inventory or quality control.
9. **Traffic Sign Recognition:**
   - Identifies handwritten or printed characters on traffic signs for intelligent transportation.
10. **Historical Document Analysis:**
    - Analyzes historical handwritten documents for digitization and preservation.

Inference –

CNN excels in diverse applications: finance (check processing), healthcare (image analysis), retail (label recognition), and transportation (sign recognition). It aids digitization efforts and ensures cultural heritage preservation. Overall, CNN offers valuable automation and efficiency across industries.

References:

https://www.geeksforgeeks.org/applying-convolutional-neural-network-on-mnist-dataset/

https://www.youtube.com/watch?v=9cPMFTwBdM4

Conclusion: Thus I Build a Multiclass classifier using the CNN model using MNIST or any other suitable dataset by Performing Data Pre-processing, Defining Model and perform training and Evaluating Results using confusion matrix.