410251: Deep Learning

Group A

Assignment No.: 1

Title of the Assignment: Linear regression by using Deep Neural network: Implement Boston housing price prediction problem by Linear regression using Deep Neural network. Use Boston House price prediction dataset..

Objective of the Assignment: Students should be able to implement linear regression by using deep neural networks. Students should know about neural networks and its importance over machine learning models.

Prerequisite:

1. Basic of Python Programming

2. Good understanding of machine learning algorithms.

3. Knowledge of basic statistics

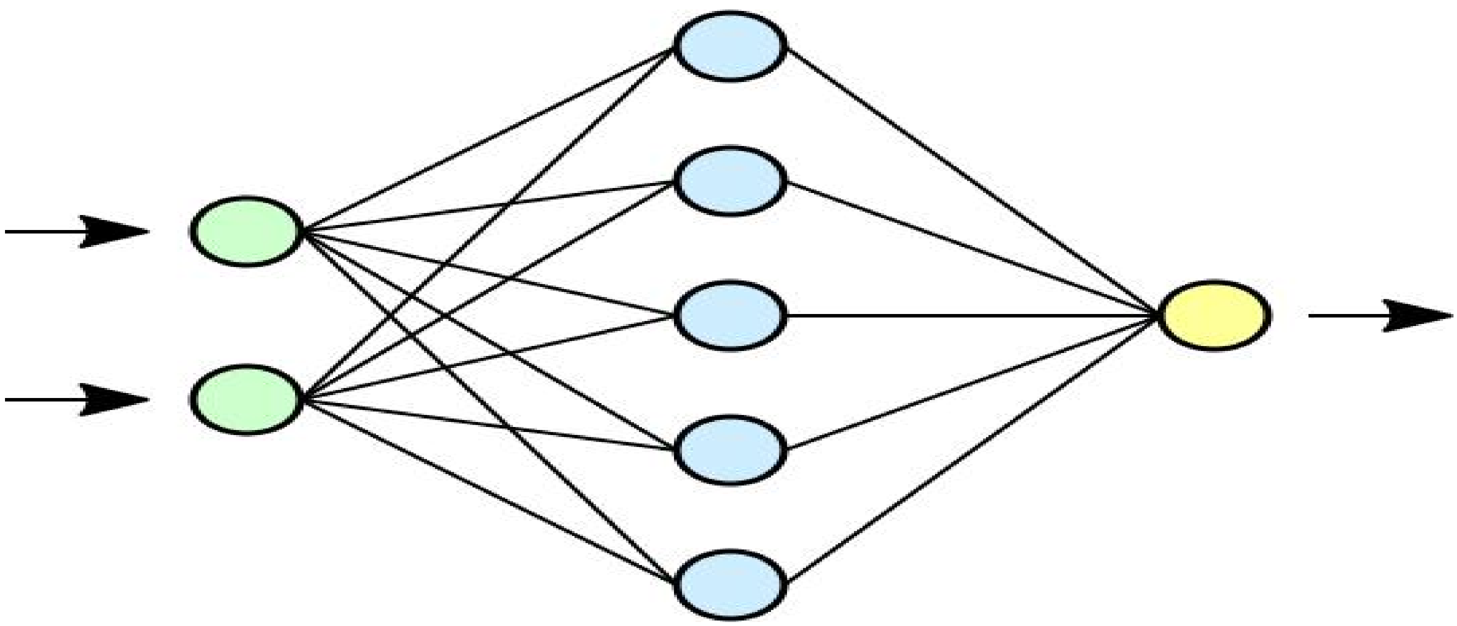
Contents for Theory:

Linear Regression : Linear regression is a basic and commonly used type of predictive analysis. The overall idea of regression is to examine two things: (1) does a set of predictor variables do a good job in predicting an outcome (dependent) variable? (2) Which variables in particular are significant predictors of the outcome variable, and in what way do they–indicated by the magnitude and sign of the beta estimates–impact the outcome variable? These regression estimates are used to explain the relationship between one dependent variable and one or more independent variables. The simplest form of the regression equation with one dependent and one independent variable is defined by the formula y = c + b*x, where y = estimated dependent variable score, c = constant, b = regression coefficient, and x = score on the independent variable.

What is a Neural Network?

The basic unit of the brain is known as a neuron; there are approximately 86 billion neurons in our nervous system which are connected to 10^14-10^15 synapses. Each neuron receives a signal from the synapses and gives output after processing the signal. This idea is drawn from the brain to build a neural network.

Each neuron performs a dot product between the inputs and weights, adds biases, applies an activation function, and gives out the outputs. When a large number of neurons are present together to give out a large number of outputs, it forms a neural layer. Finally, multiple layers combine to form a neural network.



Neural Network Architecture :

Neural networks are formed when multiple neural layers combine with each other to give out a network, or we can say that there are some layers whose outputs are inputs for other layers. The most common type of layer to construct a basic neural network is the fully connected layer, in which the adjacent layers are fully connected pairwise and neurons in a single layer are not connected to each other.

Naming conventions. When the N-layer neural network, we do not count the input layer. Therefore, a single-layer neural network describes a network with no hidden layers (input directly mapped to output). In the case of our code, we’re going to use a single-layer neural network, i.e. We do not have a hidden layer.

Output layer. Unlike all layers in a Neural Network, the output layer neurons most commonly do not have an activation function (or you can think of them as having a linear identity activation function). This is because the last output layer is usually taken to represent the class scores (e.g. in classification), which are arbitrary real-valued numbers, or some kind of real-valued target (e.g. In regression). Since we’re performing regression using a single layer, we do not have any activation function.

Sizing neural networks. The two metrics that people commonly use to measure the size of neural networks are the number of neurons, or more commonly the number of parameters.

The Boston Housing Dataset is a popular dataset in machine learning and contains information about various attributes of houses in Boston. The goal of using deep neural networks on this dataset is to predict the median value of owner-occupied homes.

The Boston Housing Dataset contains 13 input variables or features, such as crime rate, average number of rooms per dwelling, and distance to employment centers. The target variable is the median value of owner-occupied homes. The dataset has 506 rows, which is not very large, but still sufficient to train a deep neural network.

To implement a deep neural network on the Boston Housing Dataset, we can follow these steps:

Load the dataset: We can load the dataset using libraries like pandas or numpy.

Preprocess the data: We need to preprocess the data by scaling the input features so that they have zero mean and unit variance. This step is important because it helps the neural network to converge faster.

Split the dataset: We split the dataset into training and testing sets. We can use a 70/30 or 80/20 split for training and testing, respectively.

Define the model architecture: We need to define the architecture of our deep neural network. We can use libraries like Keras or PyTorch to define our model. The architecture can include multiple hidden layers with various activation functions and regularization techniques like dropout.

Compile the model: We need to compile the model by specifying the loss function, optimizer, and evaluation metrics. For regression problems like this, we can use mean squared error as the loss function and adam optimizer.

Train the model: We can train the model using the training data. We can use techniques like early stopping to prevent overfitting.

Evaluate the model: We can evaluate the model using the testing data. We can calculate the mean squared error or the mean absolute error to evaluate the performance of the model.

Overall, using a deep neural network on the Boston Housing Dataset can result in accurate predictions of the median value of owner-occupied homes. By following the above steps, we can implement a deep neural network and fine-tune its hyperparameters to achieve better performance.

Practical Implementation of Boston Dataset and prediction using deep neural network.

Step 1: Load the dataset

```
import pandas as pd

# Load the dataset from a CSV file

df = pd.read_csv('boston_housing.csv')

# Display the first few rows of the dataset
print(df.head())
```

Step 2: Preprocess the data

```
from sklearn.preprocessing import StandardScaler

# Split the data into input and output variables

X = df.drop('medv', axis=1)

y = df['medv']

# Scale the input features scaler =
StandardScaler() X =
scaler.fit_transform(X)

# Display the first few rows of the scaled input features
print(X[:5])
```

Step 3: Split the dataset

```
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
#	Print the shapes of the training and testing sets
print('Training set shape:', X_train.shape, y_train.shape)
print('Testing set shape:', X_test.shape, y_test.shape)
```

Step 4: Define the model architecture

```
from keras.models import Sequential

from keras.layers import Dense, Dropout

#	Define the model architecture model =
Sequential()

model.add(Dense(64, input_dim=13, activation='relu'))
model.add(Dropout(0.2))

model.add(Dense(32, activation='relu'))
model.add(Dense(1))

#	Display the model summary
print(model.summary())
```

Step 5: Compile the model

```
# Compile the model

model.compile(loss='mean_squared_error', optimizer='adam', metrics=['mean_absolute_error'])
```

Step 6: Train the model

```
from keras.callbacks import EarlyStopping

# Train the model

early_stopping = EarlyStopping(monitor='val_loss', patience=5)

history = model.fit(X_train, y_train, validation_split=0.2, epochs=100, batch_size=32,
callbacks=[early_stopping])
```

```
#        Plot the training and validation loss over epochs
import matplotlib.pyplot as plt plt.plot(history.history['loss'])
plt.plot(history.history['val_loss']) plt.title('Model Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss') plt.legend(['Training',
'Validation']) plt.show()
```

Step 7: Evaluate the model

```
# Evaluate the model on the testing set

loss, mae = model.evaluate(X_test, y_test)

#        Print the mean absolute error
print('Mean Absolute Error:', mae)
```

Conclusion : In this way we are able to learn about the Deep Neural Network and its implementation on the boston dataset.

Assignment No.: 2

Title of the Assignment: Classification using Deep neural network.Binary classification using Deep Neural Networks Example: Classify movie reviews into positive" reviews and "negative" reviews, just based on the text content of the reviews. Use the IMDB dataset.

Objective of the Assignment: Students should be able to implement deep neural networks on textual data and they should know the basics of natural language processing and its applications in the real world.

Prerequisite:

1. Basic of Python Programming
2. Good understanding of machine learning algorithms.
3. Knowledge of basic statistics
4. Knowledge about natural language processing.

Contents for Theory:

Classification using deep neural networks is a popular approach to solve various supervised learning problems such as image classification, text classification, speech recognition, and many more. In this approach, the neural network is trained on labeled data to learn a mapping between the input features and the corresponding output labels.

Binary classification is a type of classification problem in which the task is to classify the input data into one of the two classes. In the example of classifying movie reviews as positive or negative, the input data is the text content of the reviews, and the output labels are either positive or negative.

The deep neural network used for binary classification consists of multiple layers of interconnected neurons, which are capable of learning complex representations of the input data. The first layer of the neural network is the input layer, which takes the input data and passes it to the hidden layers.

The hidden layers perform non-linear transformations on the input data to learn more complex features. Each hidden layer consists of multiple neurons, which are connected to the neurons of the previous and next layers. The activation function of the neurons in the hidden layers introduces non-linearity into the network and allows it to learn complex representations of the input data.

The last layer of the neural network is the output layer, which produces the classification result. In binary classification, the output layer consists of one neuron, which produces the probability of the input data belonging to the positive class. The probability of the input data belonging to the negative class can be calculated as (1 - probability of positive class).

The training of the neural network involves optimizing the model parameters to minimize the loss function. The loss function measures the difference between the predicted output and the actual output. In binary classification, the commonly used loss function is binary cross-entropy loss.

The IMDB dataset is a popular dataset used for binary classification of movie reviews. It contains 50,000 movie reviews, which are split into 25,000 reviews for training and 25,000 reviews for testing. The reviews are preprocessed and encoded as sequences of integers, where each integer represents a word in the review. The deep neural network can be trained on this dataset to classify the movie reviews into positive or negative categories.

In summary, binary classification using deep neural networks involves designing a neural network architecture with multiple layers of interconnected neurons, training the network on labeled data using a suitable loss function, and using the trained network to classify new data. The IMDB dataset provides a suitable example to implement and test this approach on movie review classification.

Dataset information :

The IMDB (Internet Movie Database) dataset is a popular dataset used for sentiment analysis, particularly binary classification of movie reviews into positive or negative categories. It consists of 50,000 movie reviews, which are evenly split into a training set and a testing set, each containing 25,000 reviews.

The reviews are encoded as sequences of integers, where each integer represents a word in the review. The words are indexed based on their frequency in the dataset, with the most frequent word assigned the index 1, the second most frequent word assigned the index 2, and so on. The indexing is capped at a certain number of words, typically the top 10,000 most frequent words, to limit the size of the vocabulary.

The reviews are preprocessed to remove punctuations and convert all the letters to lowercase. The reviews are also padded or truncated to a fixed length, typically 250 words, to ensure all the input sequences have the same length. Padding involves adding zeros to the end of the review sequence to make it of the fixed length, while truncating involves cutting off the sequence at the maximum length.

The reviews are labeled as positive or negative based on the overall sentiment expressed in the review. The labels are assigned as follows: reviews with a score of 7 or higher on a scale of

1-10 are labeled as positive, while reviews with a score of less than 4 are labeled as negative. Reviews with a score between 4 and 7 are excluded from the dataset to ensure clear distinction between positive and negative categories.

The IMDB dataset is a popular benchmark dataset for sentiment analysis and has been used extensively to evaluate various machine learning and deep learning models. Its popularity is attributed to the large size of the dataset, the balanced distribution of positive and negative reviews, and the preprocessed format of the reviews.

Steps to implement the IMDB dataset sentiment analysis.

1. Load the IMDB dataset using Keras' built-in imdb.load_data() function. This function loads the dataset and preprocesses it as sequences of integers, with the labels already converted to binary (0 for negative, 1 for positive).

2. Pad or truncate the sequences to a fixed length of 250 words using Keras' pad_sequences() function.

3. Define a deep neural network architecture, consisting of an embedding layer to learn the word embeddings, followed by multiple layers of bidirectional LSTM (Long Short-Term Memory) cells, and a final output layer with a sigmoid activation function to output the binary classification.

4. Compile the model using binary cross-entropy loss and the Adam optimizer.

5. Train the model on the training set and validate on the validation set.

6. Evaluate the trained model on the test set and compute the accuracy and loss.

Code to implement sentiment analysis :

```
import numpy as np

from keras.datasets import imdb

from keras.preprocessing.sequence import pad_sequences from
keras.models import Sequential

from keras.layers import Embedding, Bidirectional, LSTM, Dense

# Load the IMDB dataset

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

#       Pad or truncate the sequences to a fixed length of 250 words
max_len = 250

x_train = pad_sequences(x_train, maxlen=max_len) x_test =
pad_sequences(x_test, maxlen=max_len)

# Define the deep neural network architecture

model = Sequential()

model.add(Embedding(input_dim=10000, output_dim=128, input_length=max_len))

model.add(Bidirectional(LSTM(64, return_sequences=True)))

model.add(Bidirectional(LSTM(32)))

model.add(Dense(1, activation='sigmoid'))

# Compile the model

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model

history = model.fit(x_train, y_train, epochs=10, batch_size=128, validation_split=0.2)

# Evaluate the model on the test set
```

```
loss, acc = model.evaluate(x_test, y_test, batch_size=128)

print(f'Test accuracy: {acc:.4f}, Test loss: {loss:.4f}')
```

This example implements a deep neural network with two layers of bidirectional LSTM cells, which are capable of learning complex patterns in sequence data. The Embedding layer learns the word embeddings from the input sequences, which are then fed into the LSTM layers. The output of the LSTM layers is then fed into a dense output layer with a sigmoid activation function, which outputs the binary classification.

The compile() method is used to compile the model with binary cross-entropy loss and the Adam optimizer. The fit() method is used to train the model on the training set for 10 epochs with a

batch size of 128. The evaluate() method is used to evaluate the trained model on the test set and compute the accuracy and loss.

This example demonstrates how deep neural networks can be used for binary classification on text data, specifically for classifying movie reviews as positive or negative based on the text content.

Conclusion :

In this way we are able to learn about the Deep Neural Network and its implementation on the IMDB dataset.Learn about sentiment analysis.

Assignment No.: 3

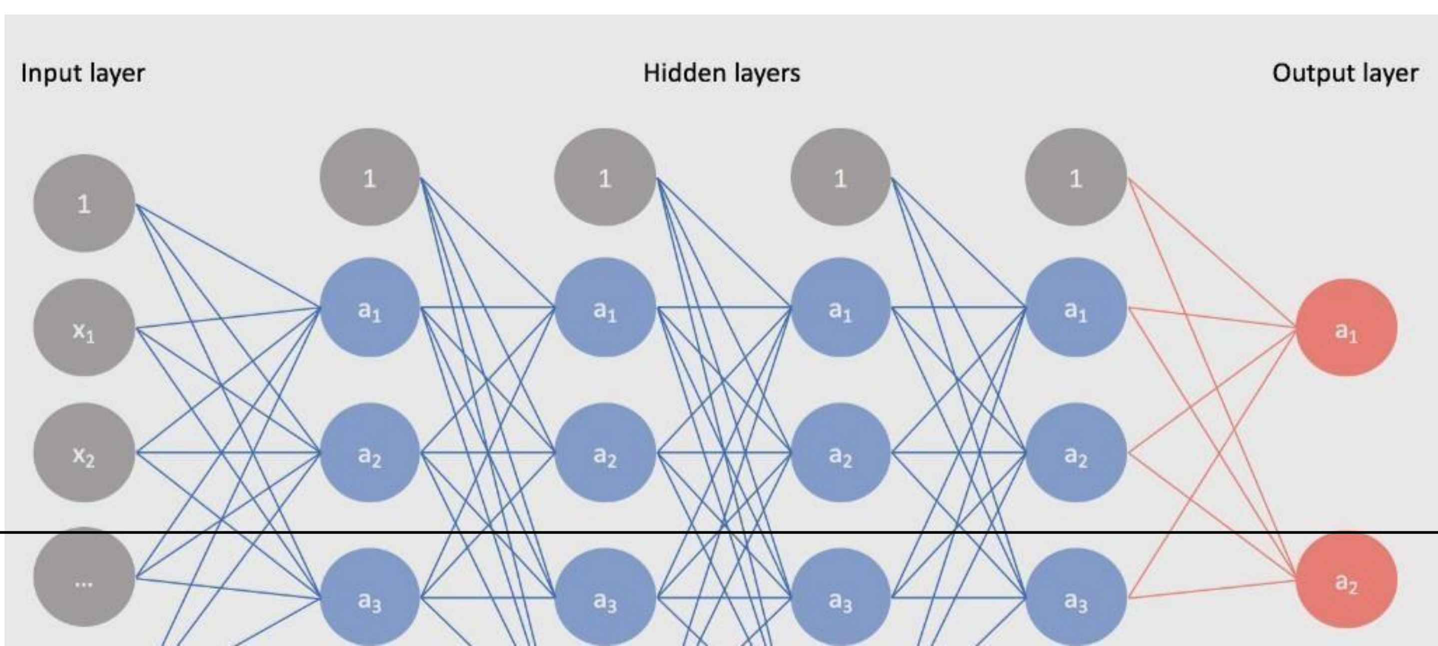
Title of the Assignment: Convolutional neural network (CNN).Use MNIST Fashion Dataset and create a classifier to classify fashion clothing into categories.

Objective of the Assignment: Students should be able to implement Convolution Neural Network. Implement classification of clothing categories on the basis of MNIST dataset.

Prerequisite:

1. Basic of Python Programming
2. Good understanding of machine learning algorithms.
3. Knowledge of basic statistics
4. Knowledge about convolution neural network and tensorflow built-in dataset.

Contents for Theory:



Convolutional Neural Networks (CNNs) are a class of artificial neural networks that are specially designed to analyze and classify images, videos, and other types of multidimensional data. They are widely used in computer vision tasks such as image classification, object detection, and image segmentation.

The main idea behind CNNs is to perform convolutions, which are mathematical operations that apply a filter to an image or other input data. The filter slides over the input data and performs a dot product between the filter weights and the input values at each position, producing a new output value. By applying different filters at each layer, the network learns to detect different features in the input data, such as edges, shapes, and textures.

CNNs typically consist of several layers that perform different operations on the input data. The most common types of layers are:

Convolutional Layers: These layers perform convolutions on the input data using a set of filters. Each filter produces a feature map, which represents the presence of a specific feature in the input data.

Pooling Layers: These layers reduce the spatial dimensions of the feature maps by taking the maximum or average value within a small region of the feature map. This reduces the amount of computation needed in the subsequent layers and makes the network more robust to small translations in the input data.

Activation Layers: These layers apply a nonlinear activation function, such as ReLU (Rectified Linear Unit), to the output of the previous layer. This introduces nonlinearity into the network and allows it to learn more complex features.

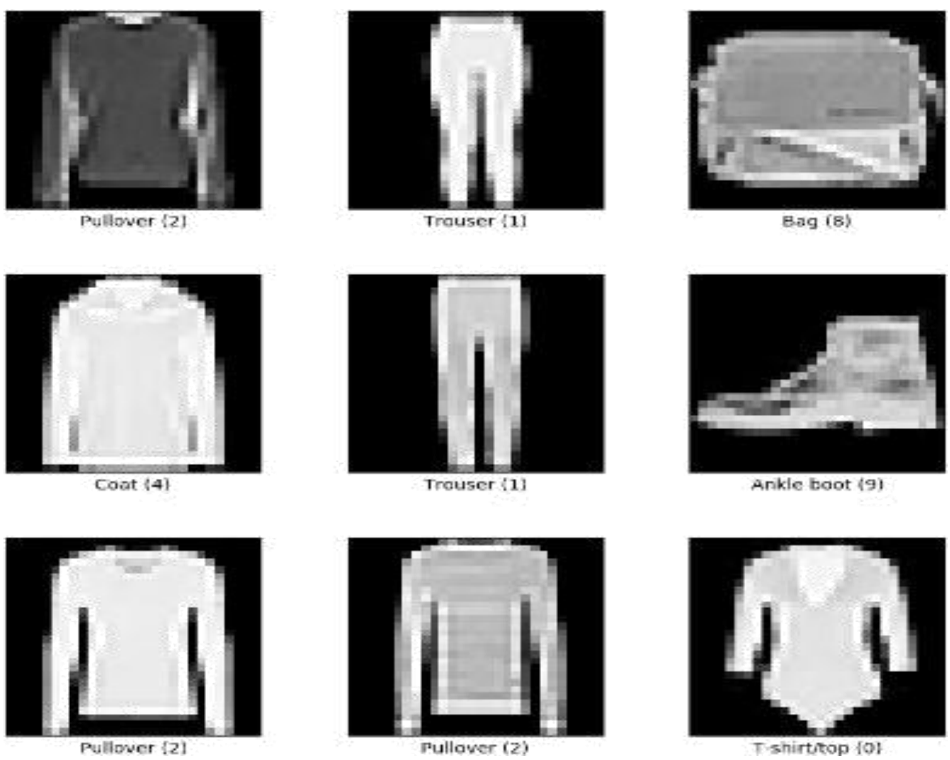
Fully-Connected Layers: These layers connect all the neurons in the previous layer to all the neurons in the current layer, similar to a traditional neural network. They are typically used at the end of the network to perform the final classification.

The architecture of a CNN is typically organized in a series of blocks, each consisting of one or more convolutional layers followed by pooling and activation layers. The output of the final block is then passed through one or more fully-connected layers to produce the final output.

CNNs are trained using backpropagation, which is a process that updates the weights of the network based on the difference between the predicted output and the true output. This process is typically done using a loss function, such as cross-entropy loss, which measures the difference between the predicted output and the true output.In summary, CNNs are a powerful class of neural networks that are specially designed for analyzing and classifying images and other types of multidimensional data.

They achieve this by performing convolutions on the input data using a set of filters, and by using different types of layers to reduce the spatial dimensions of the feature maps, introduce nonlinearity, and perform the final classification.

MNIST fashion dataset example



Dataset information :

The MNIST Fashion Dataset is a widely used benchmark dataset in the field of computer vision and machine learning. It consists of 70,000 grayscale images of clothing items, including dresses, shirts, sneakers, sandals, and more. The dataset is split into 60,000 training images and 10,000 test images, with each image being a 28x28 pixel square.

The dataset is often used as a benchmark for classification tasks in computer vision, particularly for image recognition and classification using neural networks. The dataset is considered relatively easy compared to other image datasets such as ImageNet, but it is still a challenging task due to the variability in the clothing items and the low resolution of the images.

The goal of the MNIST Fashion Dataset is to correctly classify the clothing items into one of the ten categories: T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, and Ankle boot.

The dataset was created as a replacement for the original MNIST handwritten digit dataset, which was becoming too easy for machine learning algorithms to classify accurately. The MNIST Fashion Dataset was created to provide a more challenging classification task while still being a relatively small dataset that can be used for experimentation and testing.

The dataset has been used extensively in the field of computer vision, with researchers and developers using it to test and evaluate new machine learning algorithms and models. The dataset has also been used in educational settings to teach students about machine learning and computer vision.

One common approach to tackling the MNIST Fashion Dataset is to use convolutional neural networks (CNNs), which are specifically designed to process images. CNNs consist of multiple layers, including convolutional layers, pooling layers, and fully connected layers. The convolutional layers extract features from the images, while the pooling layers downsample the features to reduce the computational complexity. The fully connected layers perform the final classification of the images.

Other approaches to tackling the MNIST Fashion Dataset include using other types of neural networks such as recurrent neural networks (RNNs) and deep belief networks (DBNs), as well as using other machine learning algorithms such as decision trees, support vector machines (SVMs), and k-nearest neighbor (KNN) classifiers.

Overall, the MNIST Fashion Dataset is a valuable benchmark dataset in the field of computer vision and machine learning, and its popularity is likely to continue as new algorithms and models are developed and tested.

Practical implementation of minist classifier is

```
import tensorflow as tf

from tensorflow import keras

import numpy as np
```

```
import matplotlib.pyplot as plt

# Load the dataset

fashion_mnist = keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

# Normalize the images

train_images = train_images / 255.0

test_images = test_images / 255.0

# Define the model

model = keras.Sequential([

   keras.layers.Flatten(input_shape=(28, 28)),

   keras.layers.Dense(128, activation='relu'),

   keras.layers.Dense(10, activation='softmax')

])

#       Compile the model
model.compile(optimizer='adam',

loss='sparse_categorical_crossentropy',

            metrics=['accuracy'])

# Train the model

model.fit(train_images, train_labels, epochs=10)

# Evaluate the model

test_loss, test_acc = model.evaluate(test_images, test_labels)

print('Test accuracy:', test_acc)

# Make predictions
```

```
predictions = model.predict(test_images)

predicted_labels = np.argmax(predictions, axis=1)

#       Show some example images and their predicted labels
num_rows = 5

num_cols = 5

num_images = num_rows * num_cols plt.figure(figsize=(2 *
2 * num_cols, 2 * num_rows)) for i in range(num_images):

   plt.subplot(num_rows, 2 * num_cols, 2 * i + 1)
   plt.imshow(test_images[i], cmap='gray') plt.axis('off')

   plt.subplot(num_rows, 2 * num_cols, 2 * i + 2)
   plt.bar(range(10), predictions[i]) plt.xticks(range(10))

   plt.ylim([0, 1])
   plt.tight_layout()

   plt.title(f"Predicted label: {predicted_labels[i]}")
plt.show()
```

Conclusion:

In this way we are able to implement Convolutional neural network (CNN) Using MNIST Fashion Dataset.

Assignment No.: 4

Title of the Assignment: Recurrent neural network (RNN) Use the Google stock prices dataset and design a time seriesanalysis and prediction system using RNN.

Objective of the Assignment: Students should be able to implement Recurrent Neural Network. Design a time seriesanalysis and prediction system using RNN.

Prerequisite:

5. Basic of Python Programming

6. Good understanding of machine learning algorithms.

7. Knowledge of basic statistics

8. Knowledge about convolution neural network and tensorflow built-in dataset.

Contents for Theory:
What is a Recurrent Neural Network?

A recurrent neural network (RNN) is a type of neural network that is designed to work with sequential data. Unlike traditional feedforward neural networks that only process input data in a single pass, RNNs maintain an internal state or memory that allows them to process sequences of input data.
This makes RNNs well-suited for tasks such as natural language processing, speech recognition, and time series analysis.

Group B

RNNs operate by passing the current input and their internal state through a set of interconnected nodes or "hidden units." Each hidden unit takes in both the current input and the previous hidden state, and produces a new hidden state as output. This process is repeated for each time step in the input sequence, with the output from the final hidden unit being used as the network's overall output.

One of the key advantages of RNNs is their ability to handle variable-length input sequences, which makes them particularly useful for tasks where the length of the input is not fixed. However, RNNs can also be difficult to train, particularly when dealing with long input sequences or complex dependencies between inputs. To address these issues, a number of modifications to the basic RNN architecture have been developed, including long short-term memory (LSTM) networks and gated recurrent units (GRUs).

Here are the steps to implement RNN:

1. Import the required libraries
2. Load the dataset
3. Prepare the data
4. Create the RNN model
5. Train the model
6. Make predictions

Code to implement RNN

Import the required libraries

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler
```