



Estd : 2001

**RN SHETTY TRUST®
RNS INSTITUTE OF TECHNOLOGY**

Autonomous Institution affiliated to Visvesvaraya Technological University, Belagavi
Approved By AICTE, New Delhi. Accredited by NAAC 'A'+ Grade
Channasandra, Dr. Vishnuvardhan Road, Bengaluru - 560 098
Ph:(080)28611880,28611881 URL: www.rnsit.ac.in

Department of CSE (Data Science)

**DEEP LEARNING AND REINFORCEMENT LEARNING LAB MANUAL
(BAI701)**

(As per Visvesvaraya Technological University Course type- IPCC)

Compiled by

DEPARTMENT OF CSE (Data Science)

R N S Institute of Technology

Bengaluru-98

Name: _____

USN: _____



RN SHETTY TRUST®
RNS INSTITUTE OF TECHNOLOGY

Autonomous Institution affiliated to Visvesvaraya Technological University, Belagavi
Approved By AICTE, New Delhi. Accredited by NAAC 'A'+ Grade
Channasandra, Dr. Vishnuvardhan Road, Bengaluru - 560 098
Ph:(080)28611880,28611881 URL: www.rnsit.ac.in

Department of CSE (Data Science)

VISION OF THE DEPARTMENT

Empowering students to solve complex real-time computing problems involving high volume multi-dimensional data.

MISSION OF THE DEPARTMENT

- Provide quality education in both theoretical and applied Computer Science to solve real world problems.
- Conduct research to develop algorithms that solve complex problems involving multi-dimensional high-volume data through intelligent inferencing.
- Develop good linkages with industry and research organizations to expose students to global problems and find optimal solutions.
- Creating confident Graduates who can contribute to the nation through high levels of commitment following ethical practices and with integrity.

Disclaimer

The information contained in this document is the proprietary and exclusive property of RNS Institute except as otherwise indicated. No part of this document, in whole or in part, may be reproduced, stored, transmitted, or used for course material development purposes without the prior written permission of RNS Institute of Technology.

The information contained in this document is subject to change without notice. The information in this document is provided for informational purposes only.

Trademark



Edition: 2024- 25

Document Owner

The primary contact for questions regarding this document is:

Author(s): Mrs. Pooja R Rao

Department: CSE (Data Science)

Contact email ids: poojarao@rnsit.ac.in



RN SHETTY TRUST®
RNS INSTITUTE OF TECHNOLOGY

Autonomous Institution affiliated to Visvesvaraya Technological University, Belagavi
Approved By AICTE, New Delhi. Accredited by NAAC 'A+' Grade
Channasandra, Dr. Vishnuvardhan Road, Bengaluru - 560 098
Ph:(080)28611880,28611881 URL: www.rnsit.ac.in

Department of CSE (Data Science)

COURSE OUTCOMES

Course Outcomes: At the end of this course, students are able to:
CO1- Demonstrate the implementation of deep learning techniques.
CO2- Examine various deep learning techniques for solving the real world problems
CO3- Design and implement research-oriented scenario using deep learning techniques in a team

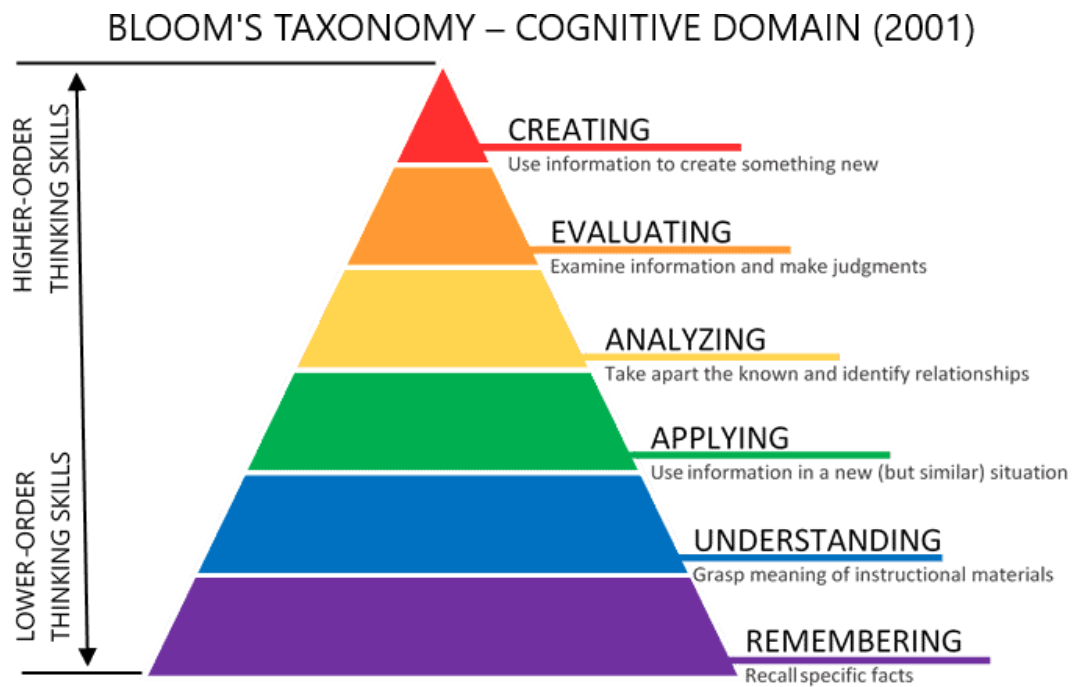
COs and POs Mapping of lab Component

COURSE OUTCOMES	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PSO1	PSO2	PSO3
CO1	3	3	3		2							3	3	2
CO2	3	3	3		2							3	3	2
CO3	3	3	3		2							3	3	2

Mapping of 'Graduate Attributes' (GAs) and 'Program Outcomes' (POs)

Graduate Attributes (GAs) (As per Washington Accord Accreditation)	Program Outcomes (POs) (As per NBA New Delhi)
Engineering Knowledge	Apply the knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems
Problem Analysis	Identify, formulate, review research literature and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences.
Design/Development of solutions	Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate considerations for the public health and safety and the cultural, societal and environmental consideration.
Conduct Investigation of complex problems	Use research – based knowledge and research methods including design of experiments, analysis and interpretation of data and synthesis of the information to provide valid conclusions.
Modern Tool Usage	Create, select and apply appropriate techniques, resources and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
The engineer and society	Apply reasoning informed by the contextual knowledge to assess society, health, safety, legal and cultural issues and the consequential responsibilities relevant to the professional engineering practice.
Environment and sustainability	Understand the impact of the professional engineering solutions in societal and environmental context and demonstrate the knowledge of and need for sustainable development.
Ethics	Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
Individual and team work	Function effectively as an individual and as a member or leader in diverse teams and in multidisciplinary settings.
Communication	Communicate effectively on complex engineering activities with the engineering community and with society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations and give and receive clear instructions.
Project management & finance	Demonstrate knowledge and understanding of the engineering and management principles and apply these to ones won work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
Life Long Learning	Recognize the need for and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

REVISED BLOOMS TAXONOMY (RBT)



PROGRAM LIST

Sl. No.	Program Description	Page No.
1	Design and implement a neural based network for generating word embedding for words in a document corpus.	8
2	Write a program to demonstrate the working of a deep neural network for classification task.	15
3	Desing and implement a Convolutional Neural Network(CNN) for classification of image dataset.	21
4	Build and demonstrate an autoencoder network using neural layers for data compression on image dataset.	27
5	Desing and implement a deep learning network for classification of textual documents.	33
6	Design and implement a deep learning network for forecasting time series data.	39
7	Write a program to enable pre-train models to classify a given image dataset.	45
8	Simple Grid World Problem: Design a custom 2D grid world where the agent navigates from a start position to a goal, avoiding obstacles. Environment: Custom grid (easily implemented in Python).	51
9	Viva Questions	56

Program 1:

Design and implement a neural based network for generating word embedding for words in a document corpus.

```
% Step 1: Load and preprocess the text corpus
textData = fileread('sample_corpus.txt');
textData = lower(textData);
textData = regexprep(textData, '[^a-z\s]', ''); % Remove punctuation
words = strsplit(strtrim(textData));

% Remove stopwords
stopWords =
["the", "is", "and", "to", "with", "this", "as", "on", "for", "it", "i", "of", "a", "in", "b
e"];
words = words(~ismember(words, stopWords));

% Step 2: Build vocabulary with frequency cutoff
% Remove empty or very short tokens
% Clean up the word list
words = words(~cellfun(@isempty, words)); % Remove empty
words = words(cellfun(@(w) length(w) > 1, words)); % Remove single-
letter words

% Count frequencies using map
wordCountsMap = containers.Map();
for i = 1:length(words)
    word = words{i};
    if isKey(wordCountsMap, word)
        wordCountsMap(word) = wordCountsMap(word) + 1;
    else
        wordCountsMap(word) = 1;
    end
end

% Extract words with min frequency
minFreq = 2;
vocabWords = keys(wordCountsMap);
counts = cell2mat(values(wordCountsMap));
keep = counts >= minFreq;

vocabWords = vocabWords(keep);
vocabSize = length(vocabWords);
word2idx = containers.Map(vocabWords, 1:vocabSize);
idx2word = vocabWords;

% Step 3: Generate skip-gram pairs
windowSize = 4;
X = [];
Y = [];

for i = 1:length(words)
    if ~isKey(word2idx, words{i})
        continue;
    end
```



```

end
centerIdx = word2idx(words{i});
for j = max(1, i - windowSize):min(length(words), i + windowSize)
    if j ~= i && isKey(word2idx, words{j})
        contextIdx = word2idx(words{j});
        X = [X; centerIdx];
        Y = [Y; contextIdx];
    end
end
end

%% Step 4: Convert to one-hot vectors
X_onehot = full(ind2vec(X', vocabSize));
Y_onehot = full(ind2vec(Y', vocabSize));

%% Step 5: Split into train/validation sets
numSamples = size(X_onehot, 2);
idx = randperm(numSamples);
trainRatio = 0.8;
numTrain = round(trainRatio * numSamples);

trainIdx = idx(1:numTrain);
valIdx = idx(numTrain+1:end);

XTrain = X_onehot(:, trainIdx)';
YTrain = categorical(vec2ind(Y_onehot(:, trainIdx)), 1:vocabSize);
XVal = X_onehot(:, valIdx)';
YVal = categorical(vec2ind(Y_onehot(:, valIdx)), 1:vocabSize);

%% Step 6: Define the neural network
embeddingDim = 100;

layers = [
    featureInputLayer(vocabSize, "Name", "input")
    fullyConnectedLayer(embeddingDim, "Name", "embedding")
    fullyConnectedLayer(vocabSize, "Name", "fc")
    softmaxLayer("Name", "softmax")
    classificationLayer("Name", "output")
];

%% Step 7: Training options
options = trainingOptions('adam', ...
    'MaxEpochs', 200, ...
    'MiniBatchSize', 256, ...
    'Shuffle', 'every-epoch', ...
    'ValidationData', {XVal, YVal}, ...
    'Verbose', false, ...
    'Plots', 'training-progress');

%% Step 8: Train the network
net = trainNetwork(XTrain, YTrain, layers, options);

%% Step 9: Final evaluation
YPredVal = classify(net, XVal);

```

```

valAccuracy = sum(YPredVal == YVal) / numel(YVal);
fprintf('\nFinal Validation Accuracy: %.2f%%\n', valAccuracy * 100);

YValProb = predict(net, XVal);
YValProb = YValProb';

trueOneHot = full(ind2vec(double(YVal)', vocabSize));
valLoss = crossentropy(YValProb, trueOneHot);
fprintf(' Final Validation Loss: %.4f\n', valLoss);

%% Step 10: Extract word embeddings
embeddingMatrix = net.Layers(2).Weights;

% Display embedding vector for a sample word
sampleWord = 'learning'; % Change as needed
if isKey(word2idx, sampleWord)
    wordIdx = word2idx(sampleWord);
    fprintf('\n Embedding vector for "%s":\n', sampleWord);
    disp(embeddingMatrix(:, wordIdx));
else
    fprintf('\n Word "%s" not found in vocabulary.\n', sampleWord);
end

```

Step-by-Step Explanation

Step	Description	Functions/Commands Used	Variables Used	Notes
1	Load and clean the text corpus	fileread, lower, regexprep, strsplit, strtrim	textData, words, stopWords	Removes punctuation, converts to lowercase, tokenizes words
2	Build vocabulary and filter infrequent words	containers.Map, isKey, cellfun, keys, values	wordCountsMap, vocabWords, word2idx	Constructs a word-index dictionary with frequency filtering
3	Generate skip-gram word pairs	for-loop, max, min, isKey	X, Y, centerIdx, contextIdx	Creates center-context word pairs within a window
4	Convert word indices to one-hot vectors	ind2vec, full	X_onehot, Y_onehot	Converts integer indices to one-hot encoded vectors
5	Split dataset into training and validation sets	randperm, round	trainIdx, valIdx, XTrain, YTrain, XVal, YVal	80-20 train-validation split
6	Define the shallow neural network for word embedding	featureInputLayer, fullyConnectedLayer, softmaxLayer, classificationLayer	layers	Defines a simple 3-layer feedforward network

7	Set training hyperparameters	trainingOptions	options	Uses 'adam' optimizer, plots training progress
8	Train the neural network	trainNetwork	net	Learns embeddings by minimizing classification error
9	Evaluate the trained model	classify, predict, vec2ind, crossentropy	valAccuracy, valLoss, YPredVal, YValProb	Accuracy and loss printed on validation set
10	Extract and display embedding vector for a sample word	net.Layers, isKey, disp	embeddingMatrix, sampleWord	Shows the learned embedding vector from the trained model

Important Variables & Their Roles

Variable	Role/Meaning
textData	Raw text data read from the corpus file
words	Preprocessed list of valid, non-stop words
stopWords	List of common words to exclude
wordCountsMap	Dictionary storing frequency of each word
vocabWords	Final vocabulary after frequency cutoff
word2idx, idx2word	Mapping from word to index and vice versa
X, Y	Integer index pairs of center and context words (training data)
X_onehot, Y_onehot	One-hot encoded vectors of x and y
XTrain, YTrain	Training inputs and labels
XVal, YVal	Validation inputs and labels
layers	Neural network layer architecture
options	Training configuration
net	Trained neural network
YPredVal	Predicted classes on validation set
YValProb	Predicted probabilities on validation set
valAccuracy	Final validation accuracy
valLoss	Final validation loss (cross-entropy)
embeddingMatrix	Learned word embeddings from the first FC layer
sampleWord	Example word used to extract and display its embedding vector

Functions & Their Use

Function/Command	Purpose
fileread	Reads full text from a file
lower, regexprep	Cleans text by converting to lowercase and removing punctuation
strsplit, strtrim	Tokenizes words by whitespace
containers.Map	Creates a dictionary (hash map) of word counts

cellfun, isKey	Applies filtering and lookups over word lists and maps
ind2vec, vec2ind	Converts between integer indices and one-hot vectors
full	Converts sparse matrix to full matrix
randperm, round	Shuffles and splits dataset
featureInputLayer	Defines input layer for one-hot encoded data
fullyConnectedLayer	Learns linear transformations (used for embeddings and output)
softmaxLayer	Converts outputs to class probabilities
classificationLayer	Computes loss during training
trainingOptions	Sets optimizer, epochs, batch size, validation, etc.
trainNetwork	Trains the model using data and layers
classify, predict	Performs inference on validation data
crossentropy	Calculates validation loss

Input: sample_corpus.txt

Deep learning is a subset of machine learning that uses artificial neural networks with multiple layers to model and learn complex patterns in data. Inspired by the structure and function of the human brain, deep learning systems can automatically extract relevant features from raw data without the need for manual feature engineering. These models are particularly effective in handling tasks such as image recognition, natural language processing, and speech analysis. A typical deep neural network includes an input layer, several hidden layers, and an output layer, where each layer transforms the data using activation functions like ReLU or sigmoid. During training, the model learns by adjusting its internal parameters through backpropagation and optimization algorithms such as gradient descent. Deep learning models require large datasets and powerful computing resources, often leveraging GPUs for efficient processing. Widely used frameworks for developing deep learning applications include TensorFlow, PyTorch, and Keras. Due to their ability to learn and generalize from vast amounts of data, deep learning techniques are at the core of advancements in areas like autonomous systems, healthcare, and artificial intelligence research.

Output:

Final Validation Accuracy: 19.05%

Final Validation Loss: 1.8237

Embedding vector for "learning":

Columns 1 through 21

```

    0.3830    0.1705   -0.2000   -0.0826   -0.0656   -0.1325   -0.2725   -
0.0725    0.1870   -0.0579    0.2604   -0.2923   -0.0291    0.0395   -0.3456
-0.3406    0.1426    0.2888   -0.3329    0.2349   -0.0116

```

Columns 22 through 42

0.2032	-0.2050	-0.2689	-0.1015	-0.2575	0.1359	-0.0119	-
0.2749	-0.2704	-0.3045	-0.1674	0.2932	-0.2699	0.0419	0.2321
0.1339	0.2158	0.0318	-0.2671	0.0743	0.1030		

Columns 43 through 63

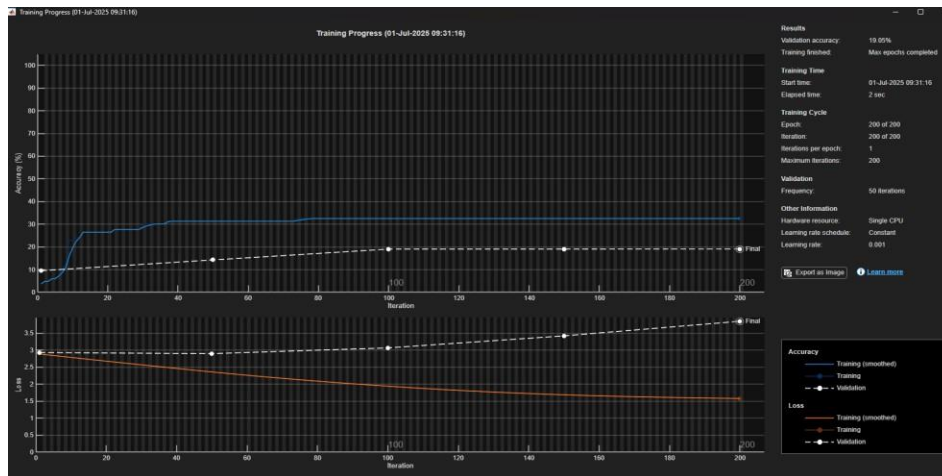
0.0284	0.2895	0.0709	0.1006	-0.2314	-0.2264	-0.0474	-
0.1249	0.2846	-0.0740	0.0455	0.0400	0.3676	0.1208	0.1959
-0.2234	0.2569	0.1923	0.3500	-0.2799	0.2390		


































Columns 64 through 84

-0.2342	0.0505	-0.1059	-0.2783	-0.1884	0.2323	-0.2963	-
0.0685	0.3391	-0.1486	0.3304	-0.1358	0.1331	-0.0175	-0.3249
0.0500	-0.1478	-0.0665	0.2572	0.1576	-0.0051		

Columns 85 through 100

0.1441	-0.1362	-0.0819	-0.0401	-0.0091	-0.3021	-0.0492	
0.2442	0.1343	0.1763	-0.0122	0.2175	0.2802	0.3605	-0.0260
-0.0108							



Name	Value	Size	Class
<input checked="" type="checkbox"/> keep	1×116 logical	1×116	logical
 layers	5×1 Layer	5×1	nnet.cnn.layer.Layer
 minFreq	2	1×1	double
 net	1×1 SeriesNetwork	1×1	SeriesNetwork
 numSamples	104	1×1	double
 numTrain	83	1×1	double
 options	1×1 TrainingOptionsADAM	1×1	nnet.cnn.TrainingOption
 sampleWord	'learning'	1×8	char
 stopWords	1×15 string	1×15	string
 textData	'deep learning is a subset of ...	1×1228	char
 trainIdx	1×83 double	1×83	double
 trainRatio	0.8000	1×1	double
 trueOneHot	18×21 double	18×21	double
 valAccuracy	0.1905	1×1	double
 valIdx	1×21 double	1×21	double
 valLoss	1.8237	1×1	single
 vocabSize	18	1×1	double
 vocabWords	1×18 cell	1×18	cell
 windowSize	4	1×1	double
 word	'research'	1×8	char
 word2idx	18×1 Map	18×1	containers.Map
 wordCountsMap	116×1 Map	116×1	containers.Map
 wordIdx	11	1×1	double
 words	1×145 cell	1×145	cell
 X	104×1 double	104×1	double
 X_onehot	18×104 double	18×104	double
 XTrain	83×18 double	83×18	double
 XVal	21×18 double	21×18	double
 Y	104×1 double	104×1	double
 Y_onehot	18×104 double	18×104	double
 YPredVal	21×1 categorical	21×1	categorical
 YTrain	83×1 categorical	83×1	categorical
 YVal	21×1 categorical	21×1	categorical
 YValProb	18×21 single	18×21	single

Program 2:

Write a program to demonstrate the working of a deep neural network for classification task.

```
%% Step 1: Load the dataset
digitDatasetPath = fullfile(matlabroot, 'toolbox', 'nnnet', 'nndemos', ...
                             'nndatasets', 'DigitDataset');
imds = imageDatastore(digitDatasetPath, ...
                     'IncludeSubfolders', true, ...
                     'LabelSource', 'foldernames');
disp("Total number of images: " + numel(imds.Files));
labelCount = countEachLabel(imds);
disp(labelCount);

%% Visualize sample images
figure;
perm = randperm(numel(imds.Files), 20);
for i = 1:20
    subplot(4,5,i);
    img = readimage(imds, perm(i));
    imshow(img);
    title(char(imds.Labels(perm(i))));
end
sgtitle('Sample Digits from the Dataset');

%% Balance the dataset
minSetCount = min(countEachLabel(imds).Count);
imds = splitEachLabel(imds, minSetCount, 'randomize');

%% Resize and split into training, validation, and test sets
imageSize = [28 28];
imds.ReadFcn = @(filename)imresize(imread(filename), imageSize);

[imdsTrainFull, imdsTest] = splitEachLabel(imds, 0.7, 'randomized');
[imdsTrain, imdsVal] = splitEachLabel(imdsTrainFull, 0.85, 'randomized'); %
85% train, 15% val

%% Define the network
layers = [
    imageInputLayer([28 28 1], 'Name', 'input')

    fullyConnectedLayer(256, 'Name', 'fc1')
    reluLayer('Name', 'relu1')

    fullyConnectedLayer(128, 'Name', 'fc2')
    reluLayer('Name', 'relu2')

    fullyConnectedLayer(64, 'Name', 'fc3')
    reluLayer('Name', 'relu3')

    fullyConnectedLayer(10, 'Name', 'fc4')
    softmaxLayer('Name', 'softmax')
    classificationLayer('Name', 'output')
];
% analyzeNetwork(layers); % Optional: uncomment for architecture view
```

```

%% Training options with validation
options = trainingOptions('adam', ...
    'MaxEpochs', 5, ...
    'MiniBatchSize', 64, ...
    'Shuffle', 'every-epoch', ...
    'ValidationData', imdsVal, ...
    'ValidationFrequency', 30, ...
    'Verbose', false, ...
    'Plots', 'training-progress');

%% Train the network
net = trainNetwork(imdsTrain, layers, options);

%% Evaluate on test set
YPredTest = classify(net, imdsTest);
YTest = imdsTest.Labels;

testAccuracy = sum(YPredTest == YTest) / numel(YTest);
fprintf('\n Test Accuracy: %.2f%%\n', testAccuracy * 100);

%% Evaluate on validation set
YPredVal = classify(net, imdsVal);
YVal = imdsVal.Labels;

valAccuracy = sum(YPredVal == YVal) / numel(YVal);
fprintf('Validation Accuracy: %.2f%%\n', valAccuracy * 100);

%% Confusion matrix
figure;
confusionchart(YTest, YPredTest);
title('Confusion Matrix - Test Data');

```

Step-wise Breakdown

Step	Description	Functions/Commands	Variables Used	Notes
1	Load the digit dataset from MATLAB toolbox	imageDatastore, fullfile	digitDatasetPath, imds	Loads images and labels from folders
2	Display 20 random digit samples	readimage, imshow, subplot	perm, img	Helps visualize dataset diversity
3	Balance the dataset by class size	countEachLabel, splitEachLabel	minSetCount, imds	Ensures equal samples per class
4	Resize and split into train, val, test sets	splitEachLabel, custom ReadFcn	imdsTrain, imdsVal, imdsTest	Resize to 28×28; typical for digit models

5	Define fully connected neural network	imageInputLayer, fullyConnectedLayer, reluLayer, softmaxLayer, classificationLayer	layers	A 4-layer feedforward net
6	Set training options	trainingOptions	options	Uses Adam optimizer, batch size 64
7	Train the model	trainNetwork	net	Uses training data and validation monitoring
8	Evaluate on test data	classify, accuracy calc	YPredTest, YTest, testAccuracy	Accuracy printed for test set
9	Evaluate on validation data	classify, accuracy calc	YPredVal, YVal, valAccuracy	Helps tune model
10	Plot confusion matrix	confusionchart	YPredTest, YTest	Visual insight into misclassifications

Important Variables

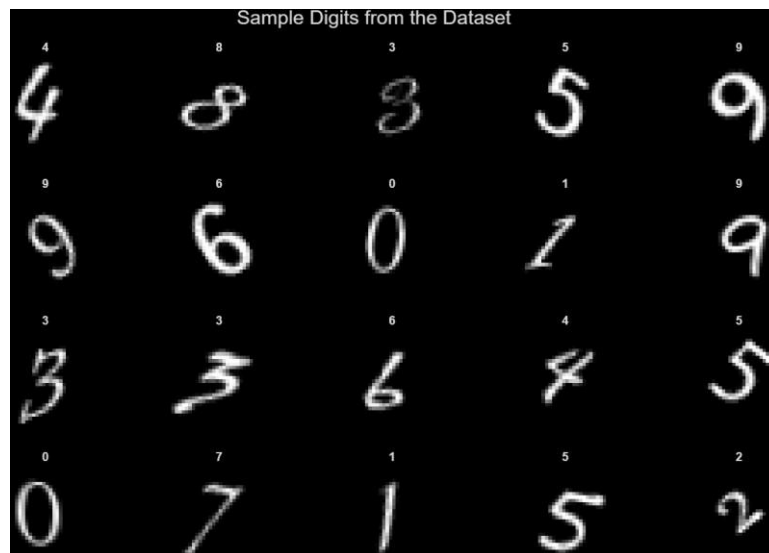
Variable	Role
digitDatasetPath	Path to the MNIST digit image dataset
imds	Complete image datastore including labels
perm	Random permutation index for displaying images
minSetCount	Minimum number of samples per class (for balancing)
imdsTrain, imdsVal, imdsTest	Split datasets for training, validation, and testing
imageSize	Target image resolution for the model
layers	Architecture of the neural network
options	Training parameters like epochs, batch size
net	Trained network object
YPredTest, YPredVal	Predicted labels by the model
YTest, YVal	True labels for test/validation data
testAccuracy, valAccuracy	Accuracy on respective datasets

Key MATLAB Functions Used

Function/Command	Purpose
imageDatastore	Load image data from folders with labels
countEachLabel	Count number of images per label
splitEachLabel	Split dataset per label with balance
readimage	Read a specific image by index

imresize	Resize images to fixed size
randperm	Generate random indices for image selection
imshow, subplot, sgtitle	Display multiple images in grid
imageInputLayer	Define input layer for image size
fullyConnectedLayer	Dense neural network layer
reluLayer	Apply ReLU activation
softmaxLayer	Compute softmax probabilities
classificationLayer	Compute cross-entropy loss
trainingOptions	Set training hyperparameters
trainNetwork	Train the deep learning model
classify	Predict class labels
confusionchart	Plot confusion matrix

Input:



Total number of images: 10000

Label	Count
0	1000
1	1000
2	1000
3	1000
4	1000
5	1000
6	1000
7	1000
8	1000
9	1000

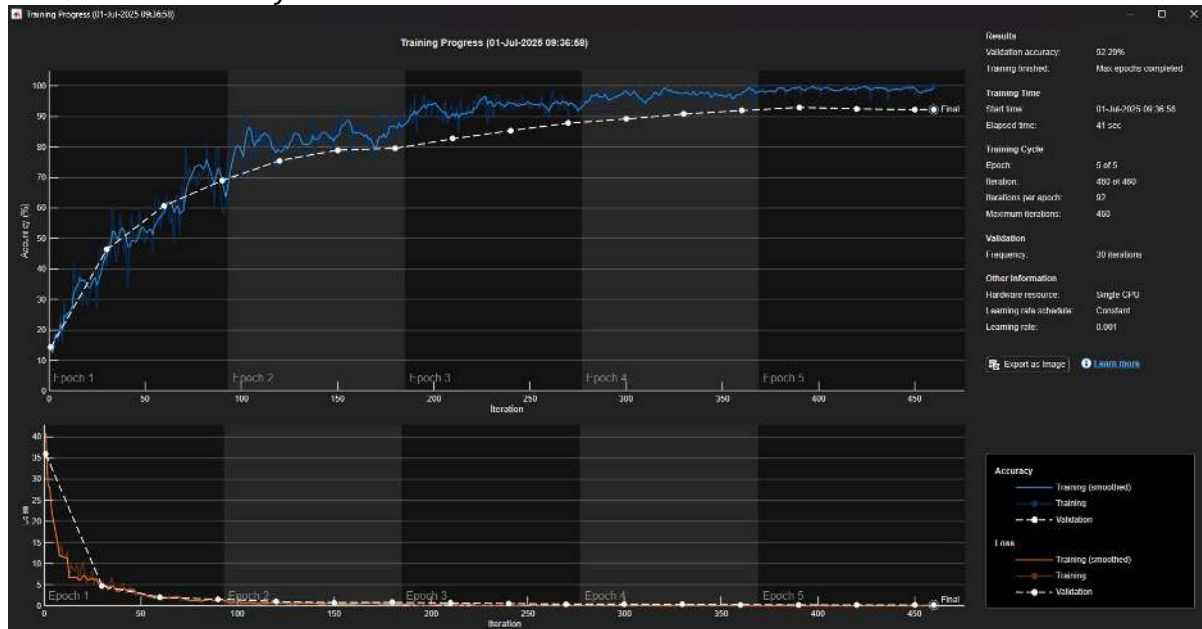
Layers Information

LAYER INFORMATION					
	Name	Type	Activations S (Spatial), T (Time), C (Channel), B (...)	Learnable Sizes	State Sizes
1	input 28×28×1 images with 'zerocenter' norma...	Image Input	28(S) × 28(S) × 1(C) × 1(B)	-	-
2	fc1 256 fully connected layer	Fully Connected	256(C) × 1(B)	Weights 256 × 784 Bias 256 × 1	-
3	relu1 ReLU	ReLU	256(C) × 1(B)	-	-
4	fc2 128 fully connected layer	Fully Connected	128(C) × 1(B)	Weights 128 × 256 Bias 128 × 1	-
5	relu2 ReLU	ReLU	128(C) × 1(B)	-	-
6	fc3 64 fully connected layer	Fully Connected	64(C) × 1(B)	Weights 64 × 128 Bias 64 × 1	-
7	relu3 ReLU	ReLU	64(C) × 1(B)	-	-
8	fc4 10 fully connected layer	Fully Connected	10(C) × 1(B)	Weights 10 × 64 Bias 10 × 1	-
9	softmax softmax	Softmax	10(C) × 1(B)	-	-

Output:

Test Accuracy: 91.60%

Validation Accuracy: 92.29%



Confusion Matrix - Test Data											
True Class	0	289		2		1	2	4	1		1
	1	1	278	5		2	1	2	11		
	2	3	10	263	5		7		8	2	2
	3		3	11	252	2	21	1	3	3	4
	4		2	2	1	290		3	1	1	
	5	2	2	1	5	1	278	3	1	3	4
	6	5	3	3		8	1	275		1	4
	7	2	7				2		288		1
	8	1	4		3	1	13	13	2	261	2
	9	4	4	1	1	5	2		4	5	274
Predicted Class											

Program 3:

Design and implement a Convolutional Neural Network(CNN) for classification of image dataset.

```
% Step 1: Load dataset
digitDatasetPath = fullfile(matlabroot, 'toolbox', 'nnnet', 'nndemos', ...
                             'nndatasets', 'DigitDataset');
imds = imageDatastore(digitDatasetPath, ...
                     'IncludeSubfolders', true, ...
                     'LabelSource', 'foldernames');

% Step 2: Balance dataset
minSetCount = min(countEachLabel(imds).Count);
imds = splitEachLabel(imds, minSetCount, 'randomize');

% Step 3: Preprocess images
imageSize = [28 28];
imds.ReadFcn = @(filename)imresize(imread(filename), imageSize);

% Step 4: Split data into train, validation, test
[imdsTrainFull, imdsTest] = splitEachLabel(imds, 0.7, 'randomized');
[imdsTrain, imdsVal] = splitEachLabel(imdsTrainFull, 0.85, 'randomized'); %
85% train, 15% val

% Step 5: Define CNN architecture
layers = [
    imageInputLayer([28 28 1], 'Name', 'input')

    convolution2dLayer(3, 8, 'Padding', 'same', 'Name', 'conv_1')
    batchNormalizationLayer('Name', 'batchnorm_1')
    reluLayer('Name', 'relu_1')
    maxPooling2dLayer(2, 'Stride', 2, 'Name', 'maxpool_1')

    convolution2dLayer(3, 16, 'Padding', 'same', 'Name', 'conv_2')
    batchNormalizationLayer('Name', 'batchnorm_2')
    reluLayer('Name', 'relu_2')
    maxPooling2dLayer(2, 'Stride', 2, 'Name', 'maxpool_2')

    fullyConnectedLayer(64, 'Name', 'fc_1')
    reluLayer('Name', 'relu_fc')

    fullyConnectedLayer(10, 'Name', 'fc_out')
    softmaxLayer('Name', 'softmax')
    classificationLayer('Name', 'output')
];

% Step 6: Visualize network
disp('Visualizing CNN Layers...');
analyzeNetwork(layers); % Optional interactive tool
lgraph = layerGraph(layers);
figure;
plot(lgraph);
title('CNN Layer Connectivity');

% Step 7: Set training options (with validation data)
```

```

options = trainingOptions('adam', ...
    'MaxEpochs', 6, ...
    'MiniBatchSize', 64, ...
    'Shuffle', 'every-epoch', ...
    'ValidationData', imdsVal, ...
    'ValidationFrequency', 30, ...
    'Verbose', false, ...
    'Plots', 'training-progress');

%% Step 8: Train network
net = trainNetwork(imdsTrain, layers, options);

%% Step 9: Evaluate on test set
YPredTest = classify(net, imdsTest);
YTrueTest = imdsTest.Labels;
testAccuracy = sum(YPredTest == YTrueTest) / numel(YTrueTest);
fprintf('\nTest Accuracy: %.2f%%\n', testAccuracy * 100);

%% Step 10: Evaluate on validation set
YPredVal = classify(net, imdsVal);
YTrueVal = imdsVal.Labels;
valAccuracy = sum(YPredVal == YTrueVal) / numel(YTrueVal);
fprintf('Validation Accuracy: %.2f%%\n', valAccuracy * 100);

%% Step 11: Confusion Matrix - Test Data
figure;
confusionchart(YTrueTest, YPredTest);
title('Confusion Matrix - CNN Test Data');

```

Step-wise Description:

Step	Description	Functions/Commands Used	Variables Used	Notes
1	Load digit dataset from MATLAB's toolbox	fullfile, imageDatastore	digitDatasetPath, imds	Reads digit images with labels from folder names
2	Balance dataset for equal samples per class	countEachLabel, min, splitEachLabel	minSetCount, imds	Ensures uniform class distribution
3	Preprocess images (resize to 28×28)	imresize, imread, anonymous function	imageSize, imds.ReadFcn	Required for consistent CNN input size
4	Split into train, validation, test sets	splitEachLabel	imdsTrain, imdsVal, imdsTest, imdsTrainFull	Maintains class balance using random sampling
5	Define CNN architecture	imageInputLayer, convolution2dLayer, reluLayer, etc.	layers	2 conv-pool blocks + FC layers
6	Visualize network layers	analyzeNetwork, layerGraph, plot	lgraph	Optional; for architecture inspection

7	Define training options	trainingOptions	options	Uses adam optimizer with validation data
8	Train CNN model	trainNetwork	net	Model is trained using training and validation sets
9	Test model on test set	classify, sum, numel	YPredTest, YTrueTest, testAccuracy	Computes test accuracy
10	Validate model on validation set	classify, sum, numel	YPredVal, YTrueVal, valAccuracy	Computes validation accuracy
11	Plot confusion matrix	confusionchart	YPredTest, YTrueTest	Shows classification performance

Key Functions & Their Use

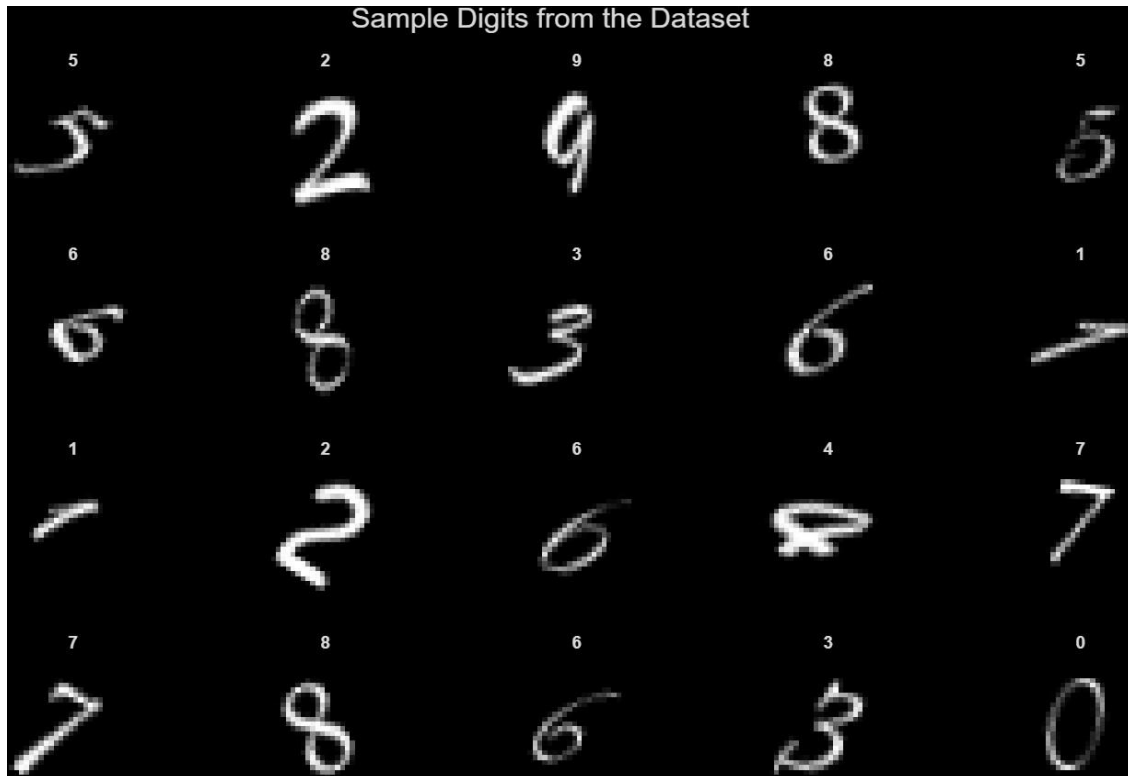
Function/Command	Purpose
imageDatastore	Loads images with labels from folder structure
splitEachLabel	Splits dataset while preserving label proportions
imresize, imread	Resizes and reads images for CNN input
imageInputLayer	Defines input dimensions for CNN
convolution2dLayer	Adds a convolutional layer to extract features
batchNormalizationLayer	Normalizes activations to stabilize training
reluLayer	Applies ReLU activation function
maxPooling2dLayer	Downsamples feature maps
fullyConnectedLayer	Connects previous layer to output
softmaxLayer	Converts scores to probabilities
classificationLayer	Calculates cross-entropy loss for classification
trainingOptions	Sets training parameters
trainNetwork	Trains the defined CNN
classify	Predicts labels using trained model
confusionchart	Visualizes confusion matrix

Important Variables & Their Roles

Variable	Role/Purpose
digitDatasetPath	Path to the digit dataset
imds	Main image datastore containing digit images and labels
minSetCount	Minimum number of images in any class (for balancing)
imageSize	Target image size [28 28] for resizing
imdsTrain, imdsVal, imdsTest	Subsets of data for training, validation, and testing

layers	Stores CNN architecture
options	Training configuration (optimizer, epochs, batch size)
net	Trained convolutional neural network
YPredTest, YPredVal	Predicted labels for test and validation data
YTrueTest, YTrueVal	Ground truth labels for evaluation
testAccuracy, valAccuracy	Performance metrics of the trained model
lgraph	Layer graph of the CNN for visualization

Input:



Total number of images: 10000

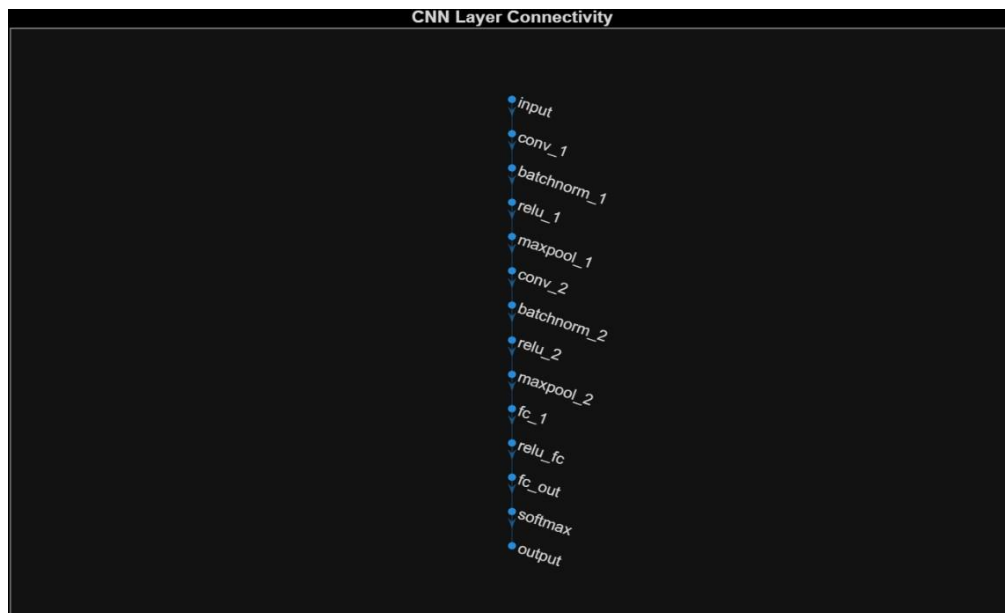
Label	Count
0	1000
1	1000
2	1000
3	1000
4	1000
5	1000
6	1000
7	1000
8	1000
9	1000

Layers Information

LAYER INFORMATION					
	Name	Type	Activations S (Spatial), T (Time), C (Channel), B (...)	Learnable Sizes	State Sizes
1	input 28x28x1 images with 'zerocenter' norma...	Image Input	28(S) × 28(S) × 1(C) × 1(B)	-	-
2	conv_1 8 3x3 convolutions with stride [1 1] and ...	2-D Convolution	28(S) × 28(S) × 8(C) × 1(B)	Weights 3 × 3 × 1 ... Bias 1 × 1 × 8	-
3	batchnorm_1 Batch normalization	Batch Normalization	28(S) × 28(S) × 8(C) × 1(B)	Offset 1 × 1 × 8 Scale 1 × 1 × 8	TrainedMean 1 × 1 × 8 TrainedVariance 1 × 1 × 8
4	relu_1 ReLU	ReLU	28(S) × 28(S) × 8(C) × 1(B)	-	-
5	maxpool_1 2x2 max pooling with stride [2 2] and pa...	2-D Max Pooling	14(S) × 14(S) × 8(C) × 1(B)	-	-
6	conv_2 16 3x3 convolutions with stride [1 1] and...	2-D Convolution	14(S) × 14(S) × 16(C) × 1(B)	Weights 3 × 3 × 8 ... Bias 1 × 1 × 16	-
7	batchnorm_2 Batch normalization	Batch Normalization	14(S) × 14(S) × 16(C) × 1(B)	Offset 1 × 1 × 16 Scale 1 × 1 × 16	TrainedMean 1 × 1 × 16 TrainedVariance 1 × 1 × 16
8	relu_2 ReLU	ReLU	14(S) × 14(S) × 16(C) × 1(B)	-	-
9	maxpool_2 2x2 max pooling with stride [2 2] and pa...	2-D Max Pooling	7(S) × 7(S) × 16(C) × 1(B)	-	-
10	fc_1 64 fully connected layer	Fully Connected	64(C) × 1(B)	Weights 64 × 784 Bias 64 × 1	-
11	relu_fc ReLU	ReLU	64(C) × 1(B)	-	-
12	fc_out 10 fully connected layer	Fully Connected	10(C) × 1(B)	Weights 10 × 64 Bias 10 × 1	-
13	softmax softmax	Softmax	10(C) × 1(B)	-	-

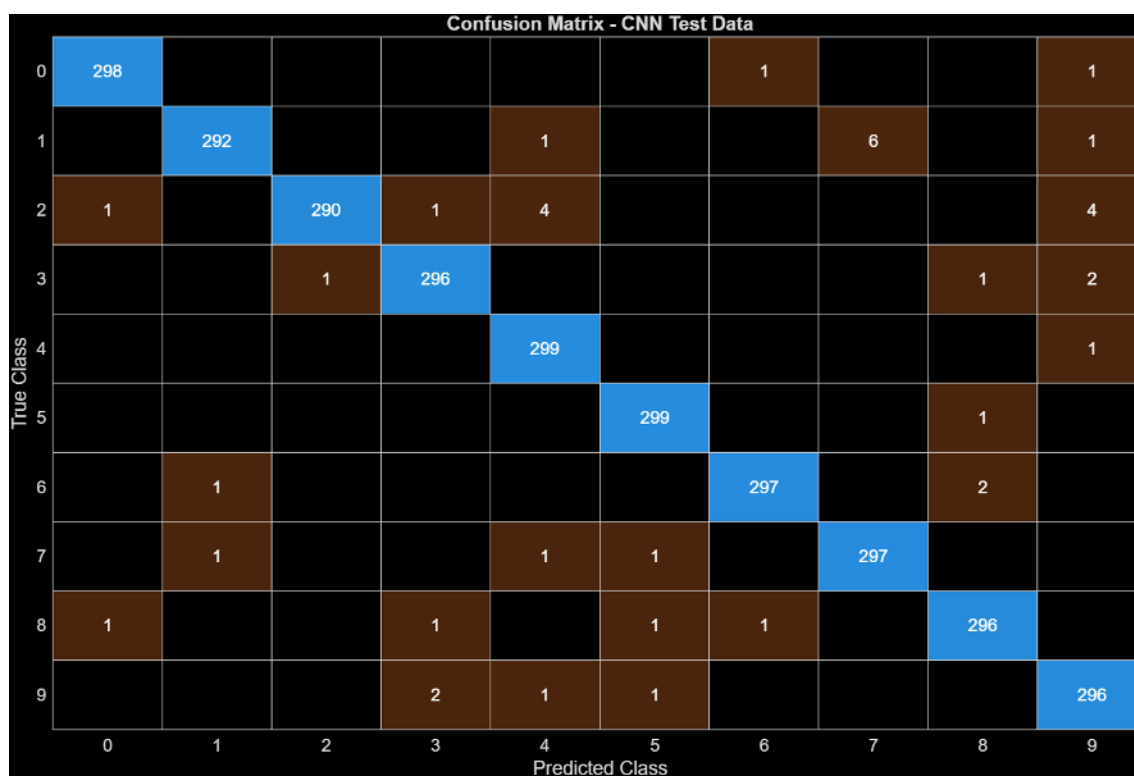
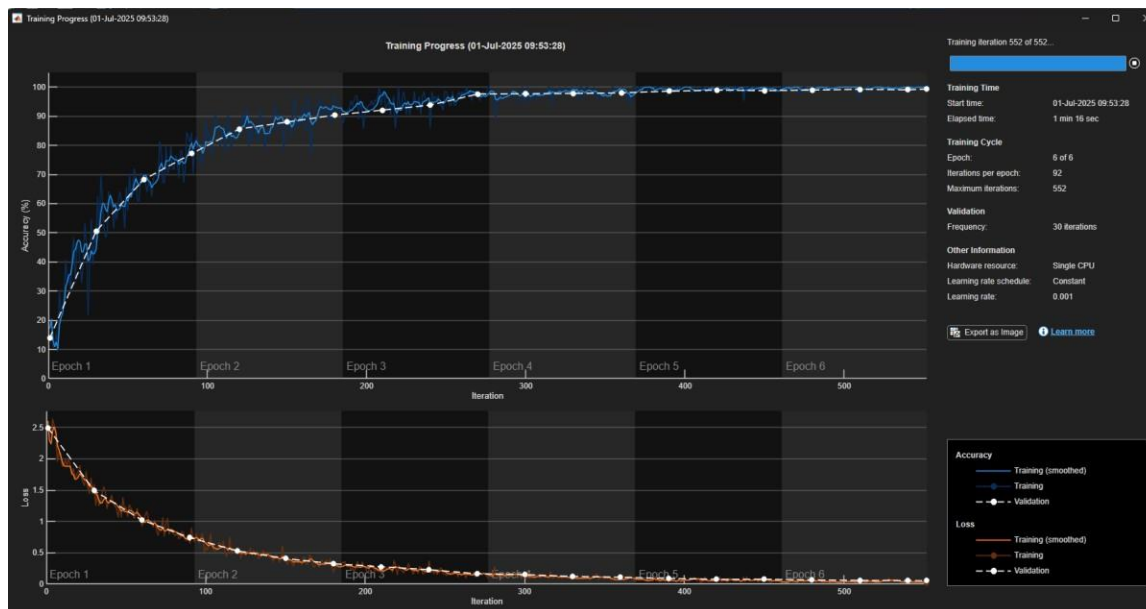
Output:

Visualizing CNN Layers...



Test Accuracy: 98.67%

Validation Accuracy: 99.33%



Program 4:

Build and demonstrate an autoencoder network using neural layers for data compression on image dataset.

```
% Step 1: Load and Balance Digit Dataset
digitDatasetPath = fullfile(matlabroot, 'toolbox', 'nnet', 'nndemos', ...
                             'nndatasets', 'DigitDataset');

imds = imageDatastore(digitDatasetPath, ...
    'IncludeSubfolders', true, ...
    'LabelSource', 'foldernames');

% Balance dataset
minSetCount = min(countEachLabel(imds).Count);
imds = splitEachLabel(imds, minSetCount, 'randomized');
imds = shuffle(imds);

% Step 2: Prepare Input Data
numImages = 1000;
X = zeros(28*28, numImages);
labels = strings(numImages, 1);

for i = 1:numImages
    img = readimage(imds, i);
    img = imresize(img, [28 28]);
    img = im2double(img);
    X(:, i) = img(:);
    labels(i) = string(imds.Labels(i));
end

X = X'; % Shape: [1000 x 784]
X = single(X);

% Step 3: Split into Train and Validation Sets
trainRatio = 0.8;
numTrain = round(trainRatio * numImages);

XTrain = X(1:numTrain, :);
XVal = X(numTrain+1:end, :);
YTrain = XTrain; % Autoencoder: input = target
YVal = XVal;

% Step 4: Define Autoencoder Neural Network Architecture
layers = [
    featureInputLayer(784, 'Name', 'input')

    fullyConnectedLayer(128, 'Name', 'fc1')
    reluLayer('Name', 'relu1')

    fullyConnectedLayer(64, 'Name', 'fc2')
    reluLayer('Name', 'relu2')

    fullyConnectedLayer(32, 'Name', 'bottleneck')

    fullyConnectedLayer(64, 'Name', 'fc3')
```

```

    reluLayer('Name', 'relu3')

    fullyConnectedLayer(128, 'Name', 'fc4')
    reluLayer('Name', 'relu4')

    fullyConnectedLayer(784, 'Name', 'fc5')
    regressionLayer('Name', 'output')
];

% Optional: View network
analyzeNetwork(layers);

%% Step 5: Training Options (with validation)
options = trainingOptions('adam', ...
    'MaxEpochs', 20, ...
    'MiniBatchSize', 64, ...
    'Shuffle', 'every-epoch', ...
    'ValidationData', {XVal, YVal}, ...
    'ValidationFrequency', 30, ...
    'Plots', 'training-progress', ...
    'Verbose', false);

%% Step 6: Train the Autoencoder
net = trainNetwork(XTrain, YTrain, layers, options);

%% Step 7: Predict Reconstructed Output (Train & Val)
YTrainPred = predict(net, XTrain);
YValPred    = predict(net, XVal);

%% Step 8: Visualize Sample Reconstructions (Train Set)
figure;
for i = 1:5
    original = reshape(XTrain(i, :), [28 28]);
    reconstructed = reshape(YTrainPred(i, :), [28 28]);

    subplot(2, 5, i);
    imshow(original);
    title(['Original: ' + labels(i)]);

    subplot(2, 5, i + 5);
    imshow(reconstructed);
    title('Reconstructed');
end
sgtitle('Autoencoder - Training Set Reconstruction');

%% Step 9: Evaluate Train and Validation Error
mseTrain = mean((XTrain(:) - YTrainPred(:)).^2);
mseVal    = mean((XVal(:) - YValPred(:)).^2);

fprintf('\n Mean Squared Reconstruction Error:\n');
fprintf('    Training Set:  %.6f\n', mseTrain);
fprintf('    Validation Set: %.6f\n', mseVal);

```

Step-wise Description

Step	Description	Functions/Commands Used	Variables Used	Notes
1	Load and balance the digit dataset from MATLAB demo folder	fullfile, imageDatastore, countEachLabel, splitEachLabel, shuffle	digitDatasetPath, imds, minSetCount	Ensures class balance by equalizing label count
2	Preprocess images and create input matrix X with labels	readimage, imresize, im2double	X , labels, img, numImages	28×28 grayscale images are flattened into 784-dim vectors
3	Split into training and validation sets	round, matrix slicing	X_{Train} , X_{Val} , Y_{Train} , Y_{Val} , trainRatio, numTrain	Used for training and validating the autoencoder
4	Define autoencoder architecture with encoder-decoder layers	featureInputLayer, fullyConnectedLayer, reluLayer, regressionLayer	layers	Bottleneck of 32 units is the encoded representation
5	Set training options including validation data	trainingOptions	options	Uses adam optimizer and tracks training progress
6	Train the network with training data	trainNetwork	net	Learns to reconstruct input via self-supervised training
7	Predict reconstructed output using trained autoencoder	predict	$Y_{TrainPred}$, $Y_{ValPred}$	Used for error calculation and visualization
8	Visualize original vs. reconstructed images	reshape, imshow, subplot, sgtitle	original, reconstructed	Shows how well the autoencoder replicates inputs
9	Evaluate mean squared error for train & validation sets	mean, element-wise operations	mseTrain, mseVal	Measures reconstruction loss (lower is better)

Key Functions & Their Use

Function/Command	Purpose
fullfile	Constructs file path across OS
imageDatastore	Loads image dataset with labels
countEachLabel	Counts images per label
splitEachLabel	Balances dataset by class
shuffle	Randomizes order of data
readimage	Reads image from datastore
imresize	Resizes image to 28×28
im2double	Converts image to double precision
featureInputLayer	Input layer for flattened image vectors
fullyConnectedLayer	Dense layer with learnable weights
reluLayer	Applies ReLU activation
regressionLayer	Output layer for real-valued output
trainingOptions	Specifies training parameters
trainNetwork	Trains neural network
predict	Predicts output using trained model
reshape	Converts flat vector back to image
imshow, subplot	Displays images for comparison
mean	Calculates mean squared error

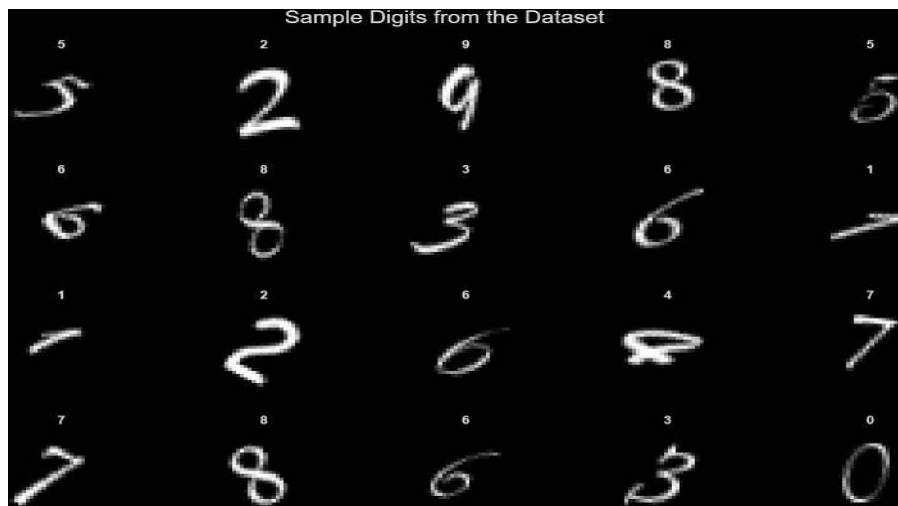
Important Variables & Their Roles

Variable	Role
imds	ImageDatastore containing digit images and labels
X	Input matrix of size [1000 × 784], each row is a flattened image
labels	String labels of each image
XTrain, XVal	Training and validation image inputs
YTrain, YVal	Same as inputs for autoencoder (input = output)
layers	Stores the architecture of the autoencoder
options	Holds training configurations like optimizer and epochs
net	Trained autoencoder model
YTrainPred, YValPred	Predicted reconstructed images
mseTrain, mseVal	Mean squared error for training and validation reconstructions

Layers Information

LAYER INFORMATION					
	Name	Type	Activations S (Spatial), T (Time), C (Channel), B (...)	Learnable Sizes	State Sizes
1	input 784 features	Feature Input	$784(C) \times 1(B)$	-	-
2	fc1 128 fully connected layer	Fully Connected	$128(C) \times 1(B)$	Weights 128×784 Bias 128×1	-
3	relu1 ReLU	ReLU	$128(C) \times 1(B)$	-	-
4	fc2 64 fully connected layer	Fully Connected	$64(C) \times 1(B)$	Weights 64×128 Bias 64×1	-
5	relu2 ReLU	ReLU	$64(C) \times 1(B)$	-	-
6	bottleneck 32 fully connected layer	Fully Connected	$32(C) \times 1(B)$	Weights 32×64 Bias 32×1	-
7	fc3 64 fully connected layer	Fully Connected	$64(C) \times 1(B)$	Weights 64×32 Bias 64×1	-
8	relu3 ReLU	ReLU	$64(C) \times 1(B)$	-	-
9	fc4 128 fully connected layer	Fully Connected	$128(C) \times 1(B)$	Weights 128×64 Bias 128×1	-
10	relu4 ReLU	ReLU	$128(C) \times 1(B)$	-	-
11	fc5 784 fully connected layer	Fully Connected	$784(C) \times 1(B)$	Weights 784×128 Bias 784×1	-

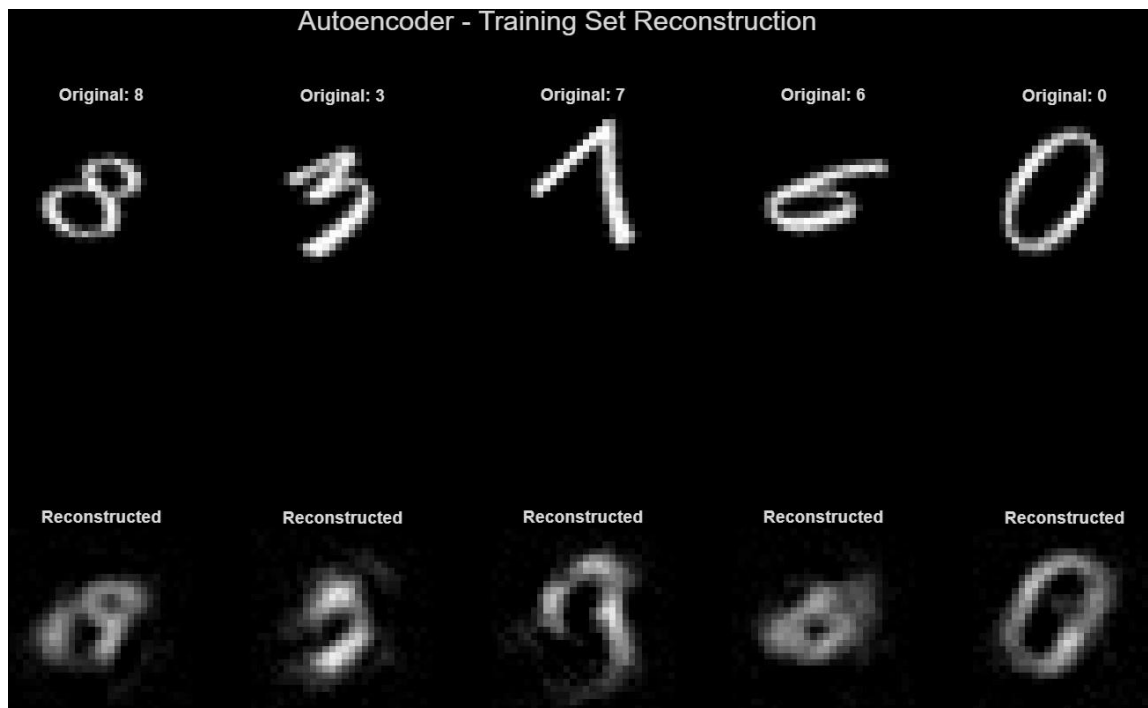
Input:



Total number of images: 10000

Label	Count
0	1000
1	1000
2	1000
3	1000
4	1000
5	1000
6	1000
7	1000
8	1000
9	1000

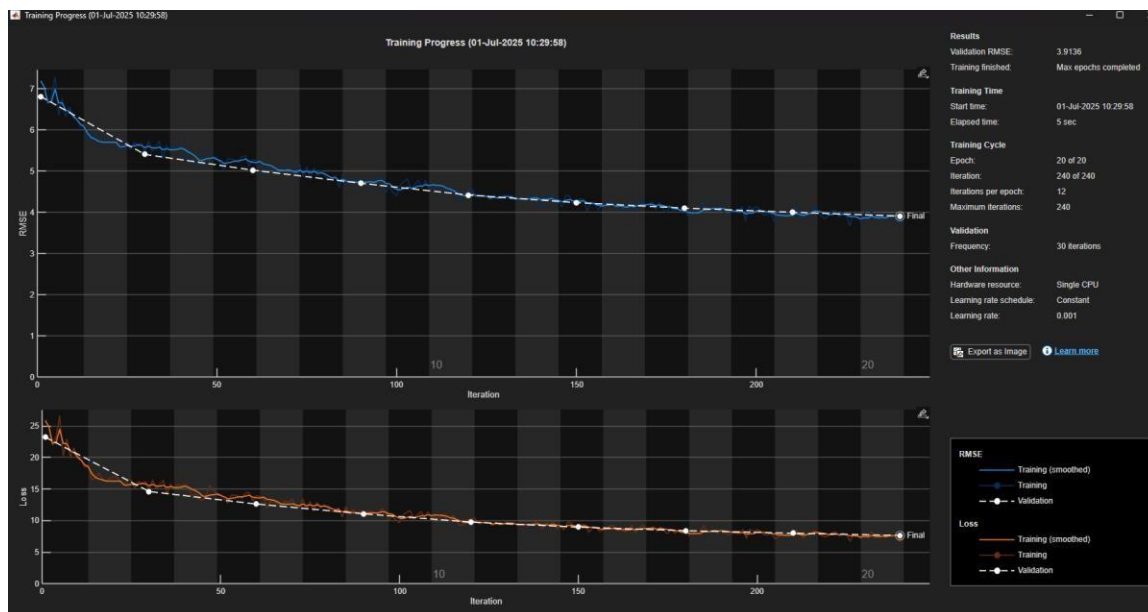
Output



Mean Squared Reconstruction Error:

Training Set: 0.018990

Validation Set: 0.019536



Program 5:

Design and implement a deep learning network for classification of textual documents.

```
%% 1. Load Data from CSV
data = readtable('text_classification_data.csv', 'TextType', 'string');
documents = data.Text;
labels = data.Label;
Y = categorical(labels);

%% 2. Split into Training and Validation (80-20)
cv = cvpartition(Y, 'HoldOut', 0.2);
idxTrain = training(cv);
idxVal = test(cv);

trainDocs = documents(idxTrain);
trainLabels = Y(idxTrain);

valDocs = documents(idxVal);
valLabels = Y(idxVal);

%% 3. Text Preprocessing
trainTokenized = tokenizedDocument(trainDocs);
valTokenized = tokenizedDocument(valDocs);

enc = wordEncoding(trainTokenized);
XTrain = doc2sequence(enc, trainTokenized);
XVal = doc2sequence(enc, valTokenized);

%% 4. Define Network Architecture
inputSize = 1;
embeddingDimension = 50;
numHiddenUnits = 100;
numClasses = numel(categories(Y));

layers = [
    sequenceInputLayer(inputSize)
    wordEmbeddingLayer(embeddingDimension, enc.NumWords)
    lstmLayer(numHiddenUnits, 'OutputMode', 'last')
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer
];

%% 5. Analyze Model Structure
disp('Launching Model Analyzer...')
analyzeNetwork(layers); % GUI analysis

%% 6. Training Options
options = trainingOptions('adam', ...
    'MaxEpochs', 30, ...
    'MiniBatchSize', 4, ...
    'ValidationData', {XVal, valLabels}, ...
    'Shuffle', 'every-epoch', ...
    'Verbose', 0, ...
```

```

        'Plots', 'training-progress');

%% 7. Train the Network
net = trainNetwork(XTrain, trainLabels, layers, options);

%% 8. Compute Accuracy
YPredTrain = classify(net, XTrain);
trainAccuracy = mean(YPredTrain == trainLabels);
fprintf('Training Accuracy: %.2f%%\n', trainAccuracy * 100);

YPredVal = classify(net, XVal);
valAccuracy = mean(YPredVal == valLabels);
fprintf('Validation Accuracy: %.2f%%\n\n', valAccuracy * 100);

%% 9. Show Actual vs Predicted Labels on Validation Set
disp('Validation Set Results (Actual vs Predicted):')
for i = 1:numel(valDocs)
    fprintf('Doc %d: "%s"\n Actual: %s | Predicted: %s\n\n', ...
        i, valDocs(i), string(valLabels(i)), string(YPredVal(i)));
end

%% 10. Visualize Model Architecture
lgraph = layerGraph(layers);
figure;
plot(lgraph);
title('Model Architecture');

%% 11. Test on New Samples
testSamples = [
    "He scored an amazing goal in the final match"
    "The new AI processor improves smartphone performance"
    "They won the football league this year"
    "Innovative apps are changing the mobile industry"
    "The athlete broke the world record in running"
];

testActualLabels = [
    "sports"
    "technology"
    "sports"
    "technology"
    "sports"
];

testActualLabels = categorical(testActualLabels);
testTokenized = tokenizedDocument(testSamples);
XTest = doc2sequence(enc, testTokenized);
YPredTest = classify(net, XTest);

%% 12. Display Test Results: Actual vs Predicted
disp('Test Sample Results (Actual vs Predicted):')
for i = 1:numel(testSamples)
    fprintf('Sample %d: "%s"\n Actual: %s | Predicted: %s\n\n', ...
        i, testSamples(i), string(testActualLabels(i)), string(YPredTest(i)));
end

```

Step-wise Description

Step	Description	Functions/Commands	Variables Used	Notes
1	Load CSV data with text and labels	readtable, categorical	data, documents, labels, Y	Reads raw data from CSV file
2	Split data into 80-20 for training and validation	cvpartition, training, test	cv, idxTrain, idxVal, trainDocs, trainLabels, valDocs, valLabels	Ensures model has unseen validation set
3	Tokenize text documents and encode	tokenizedDocument, wordEncoding, doc2sequence	trainTokenized, valTokenized, enc, XTrain, XVal	Converts text to numeric sequences
4	Define LSTM architecture	sequenceInputLayer, wordEmbeddingLayer, lstmLayer, etc.	layers	Defines model layers for training
5	Analyze network architecture	analyzeNetwork	—	GUI tool to visualize the model
6	Set training options	trainingOptions	options	Chooses Adam optimizer, batch size, etc.
7	Train the network	trainNetwork	net	Learns to map text to labels
8	Evaluate model performance	classify, mean	YPredTrain, trainAccuracy, YPredVal, valAccuracy	Checks how accurate model is
9	Display actual vs predicted labels	fprintf, string	valDocs, valLabels, YPredVal	Human-readable output
10	Visualize model structure	layerGraph, plot	lgraph	Shows layer connectivity
11	Test on new text samples	classify, tokenizedDocument, doc2sequence	testSamples, testTokenized, YPredTest	Predicts categories of new samples
12	Display test predictions	fprintf, string	testActualLabels, YPredTest	Final performance feedback

Key Functions & Their Use

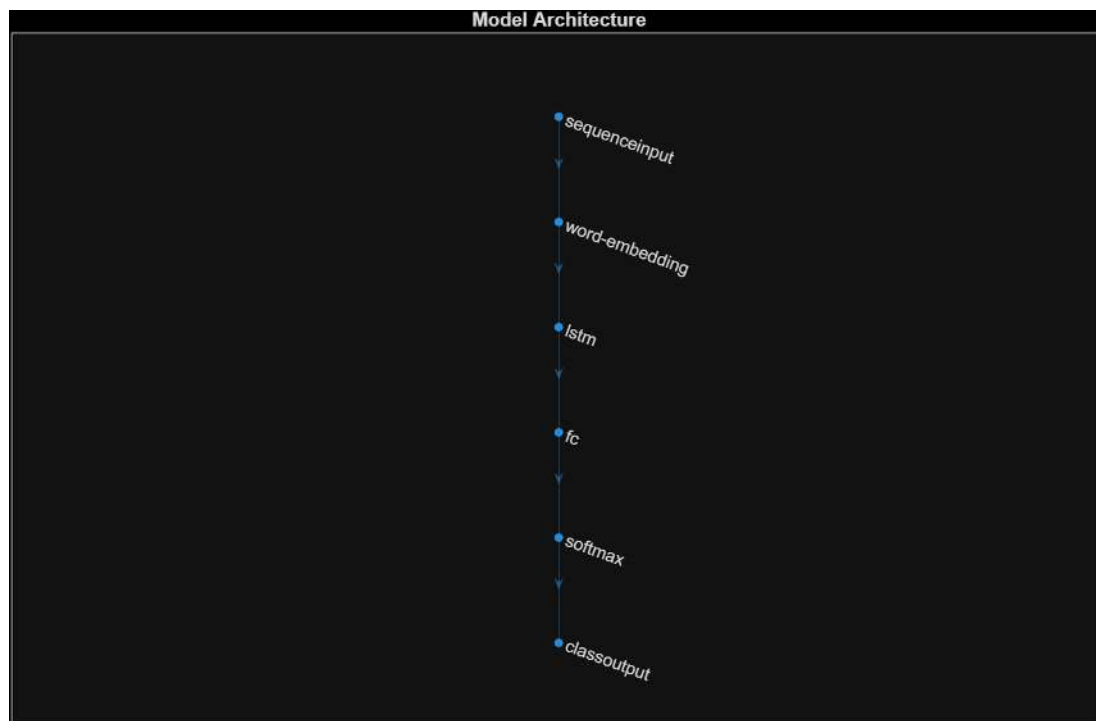
Function/Command	Purpose
readtable	Loads data from CSV file
categorical	Converts string labels to categorical type
cvpartition	Splits data into training and validation sets
tokenizedDocument	Breaks text into tokens (words)
wordEncoding	Maps tokens to unique integers
doc2sequence	Converts documents to numeric sequence inputs
sequenceInputLayer	Defines input layer for sequences
wordEmbeddingLayer	Learns vector representation of words
lstmLayer	Adds LSTM layer for sequence modeling
fullyConnectedLayer	Connects to output layer
softmaxLayer	Outputs class probabilities
classificationLayer	Computes loss and accuracy
trainNetwork	Trains model using training data
classify	Predicts classes for inputs
mean	Calculates average accuracy
analyzeNetwork	GUI to view model architecture
layerGraph, plot	Visualizes layer connections
fprintf	Prints formatted output

Important Variables & Their Roles

Variable	Role
data	Loaded dataset from CSV
documents	Raw text from dataset
labels, Y	Ground truth class labels
trainDocs, valDocs	Training and validation text
trainLabels, valLabels	Training and validation labels
trainTokenized, valTokenized	Tokenized text sequences
enc	Word encoding object
XTrain, XVal	Numeric input sequences
layers	Deep learning network architecture
options	Training configuration
net	Trained deep learning model
YPredTrain, YPredVal	Model predictions
trainAccuracy, valAccuracy	Accuracy metrics
testSamples	New sample text for testing
YPredTest, testActualLabels	Final predictions and ground truth

Layers Information

LAYER INFORMATION					
	Name	Type	Activations S (Spatial), T (Time), C (Channel), B (...)	Learnable Sizes	State Sizes
1	sequenceinput Sequence input with 1 dimensions	Sequence Input	$1(C) \times 1(B) \times 1(T)$	-	-
2	word-embedding Word embedding layer with 50 dimension...	Word Embedding La...	$50(C) \times 1(B) \times 1(T)$	Weights 50×45	-
3	lstm LSTM with 100 hidden units	LSTM	$100(C) \times 1(B)$	InputWeigh... $400 \times \dots$ RecurrentW... $400 \times \dots$ Bias $400 \times \dots$	HiddenState 100×1 CellState 100×1
4	fc 2 fully connected layer	Fully Connected	$2(C) \times 1(B)$	Weights 2×100 Bias 2×1	-
5	softmax softmax	Softmax	$2(C) \times 1(B)$	-	-



Input: text_classification_data.csv

Text	Label
The team won the championship with a great goal	sports
The match was exciting and full of energy	sports
The player scored a hat trick in the game	sports
The new smartphone has advanced technology	technology
Artificial intelligence is changing the tech world	technology
The computer has a powerful new processor	technology
He scored an amazing goal in the final match	sports
The athlete broke the world record in running	sports
Innovative apps are changing the mobile industry	technology
The new AI processor improves smartphone performance	technology

Output:

Training Accuracy: 100.00%

Validation Accuracy: 100.00%

Validation Set Results (Actual vs Predicted):

Doc 1: "The player scored a hat trick in the game"

Actual: sports | Predicted: sports

Doc 2: "The new AI processor improves smartphone performance"

Actual: technology | Predicted: technology

Test Sample Results (Actual vs Predicted):

Sample 1: "He scored an amazing goal in the final match"

Actual: sports | Predicted: sports

Sample 2: "The new AI processor improves smartphone performance"

Actual: technology | Predicted: technology

Sample 3: "They won the football league this year"

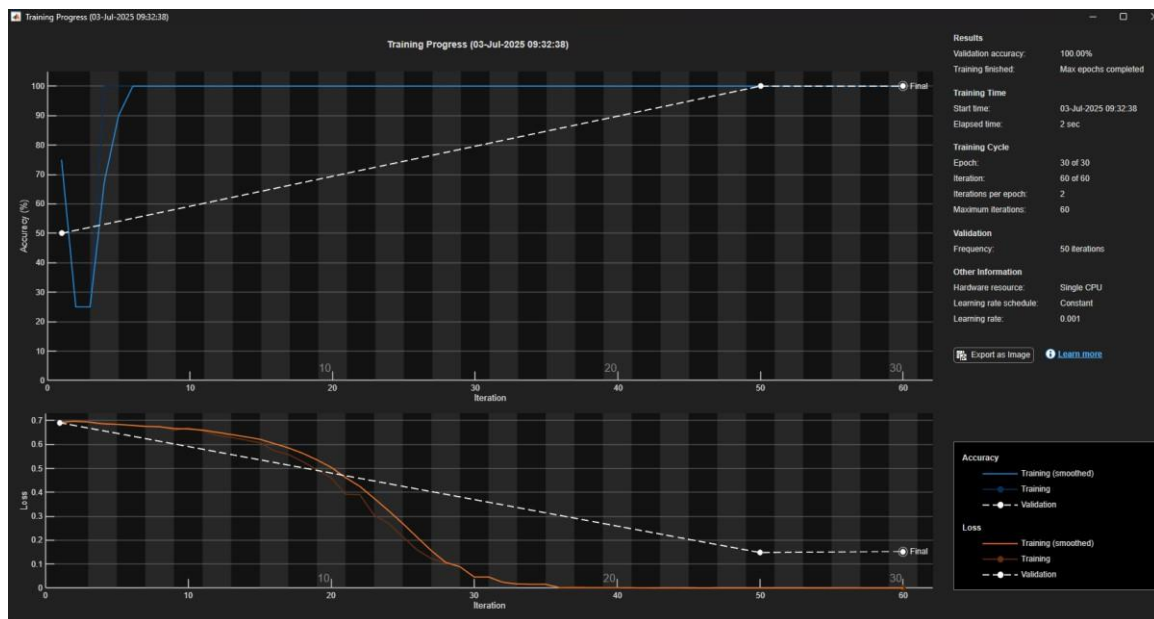
Actual: sports | Predicted: technology

Sample 4: "Innovative apps are changing the mobile industry"

Actual: technology | Predicted: technology

Sample 5: "The athlete broke the world record in running"

Actual: sports | Predicted: sports



Program 6:

Design and implement a deep learning network for forecasting time series data.

```
% 1. Simulated Time Series Data
numTimeSteps = 200;
t = (1:numTimeSteps)';
data = sin(0.05*t) + 0.1*randn(numTimeSteps,1);

figure;
plot(t, data);
title('Original Time Series');
xlabel('Time Step');
ylabel('Value');
grid on;

% 2. Define Sequence Inputs and Targets
X = data(1:end-1)';
Y = data(2:end)';

X = {X};
Y = {Y};

% 3. Split Data: Train (80%), Val (10%), Test (10%)
totalSteps = numel(X{1});
numTrain = floor(0.8 * totalSteps);
numVal = floor(0.1 * totalSteps);

XTrain = {X{1}(:, 1:numTrain)};
YTrain = {Y{1}(:, 1:numTrain)};

XVal = {X{1}(:, numTrain+1:numTrain+numVal)};
YVal = {Y{1}(:, numTrain+1:numTrain+numVal)};

XTest = {X{1}(:, numTrain+numVal+1:end)};
YTest = {Y{1}(:, numTrain+numVal+1:end)};

% 4. Define LSTM Network
numFeatures = 1;
numResponses = 1;
numHiddenUnits = 100;

layers = [
    sequenceInputLayer(numFeatures)
    lstmLayer(numHiddenUnits)
    fullyConnectedLayer(numResponses)
    regressionLayer
];

% 5. Visualize Model Architecture
disp('Opening Model Analyzer...');
analyzeNetwork(layers);
```

```

lgraph = layerGraph(layers);
figure;
plot(lgraph);
title('LSTM Network Architecture');

%% 6. Training Options
options = trainingOptions('adam', ...
    'MaxEpochs', 200, ...
    'GradientThreshold', 1, ...
    'InitialLearnRate', 0.005, ...
    'MiniBatchSize', 1, ...
    'Shuffle', 'never', ...
    'ValidationData', {XVal, YVal}, ...
    'Verbose', 0, ...
    'Plots', 'training-progress');

%% 7. Train the Network
net = trainNetwork(XTrain, YTrain, layers, options);

%% 8. Predict on Validation Data
YPredVal = predict(net, XVal, 'MiniBatchSize', 1);
rmseVal = sqrt(mean((YPredVal{1} - YVal{1}).^2));
fprintf('Validation RMSE: %.4f\n', rmseVal);

figure;
plot(YVal{1}, '-o', 'DisplayName', 'Actual');
hold on;
plot(YPredVal{1}, '-x', 'DisplayName', 'Predicted');
title('Validation Forecast');
xlabel('Time Step');
ylabel('Value');
legend;
grid on;

%% 9. Predict on Test Data
YPredTest = predict(net, XTest, 'MiniBatchSize', 1);
rmseTest = sqrt(mean((YPredTest{1} - YTest{1}).^2));
fprintf('Test RMSE: %.4f\n', rmseTest);

%% 10. Plot Test Results
figure;
plot(YTest{1}, '-o', 'DisplayName', 'Actual');
hold on;
plot(YPredTest{1}, '-x', 'DisplayName', 'Predicted');
title('Test Forecast');
xlabel('Time Step');
ylabel('Value');
legend;
grid on;

```


Step-wise Description

Step	Description	Functions/Commands	Variables Used	Notes
1	Generate noisy sine wave as synthetic time series data	sin, randn, plot	t, data	Used for simulating real-world time series
2	Prepare sequential input and target values	Indexing, cell conversion {}	X, Y	Input is previous value, target is next value
3	Split data into training, validation, and test sets	floor, indexing	XTrain, YTrain, XVal, YVal, XTest, YTest	80-10-10 split
4	Define LSTM network for regression	sequenceInputLayer, lstmLayer, etc.	layers, numFeatures, numHiddenUnits	Output is a continuous value (regression)
5	Visualize and analyze model architecture	analyzeNetwork, layerGraph, plot	layers, lgraph	Helpful for understanding LSTM structure
6	Set training parameters	trainingOptions	options	Optimizer, epochs, learning rate etc.
7	Train LSTM model using training data	trainNetwork	net	Takes training data, layers, options
8	Predict and evaluate model on validation data	predict, sqrt, mean, plot	YPredVal, rmseVal, YVal	Calculates RMSE and plots forecast
9	Predict and evaluate model on unseen test data	predict, sqrt, mean, plot	YPredTest, rmseTest, YTest	Assesses generalization ability
10	Plot and compare actual vs predicted test results	plot, legend, grid on	YTest, YPredTest	Final performance visualization

Key Functions & Their Use

Function/Command	Purpose
sin, randn	Generates base signal and adds noise to simulate real data
floor	Rounds down to determine split sizes
trainNetwork	Trains deep learning model using LSTM
sequenceInputLayer	Accepts sequential time-series data
lstmLayer	Captures temporal dependencies in sequences
fullyConnectedLayer	Maps hidden units to output value
regressionLayer	Computes loss for continuous output

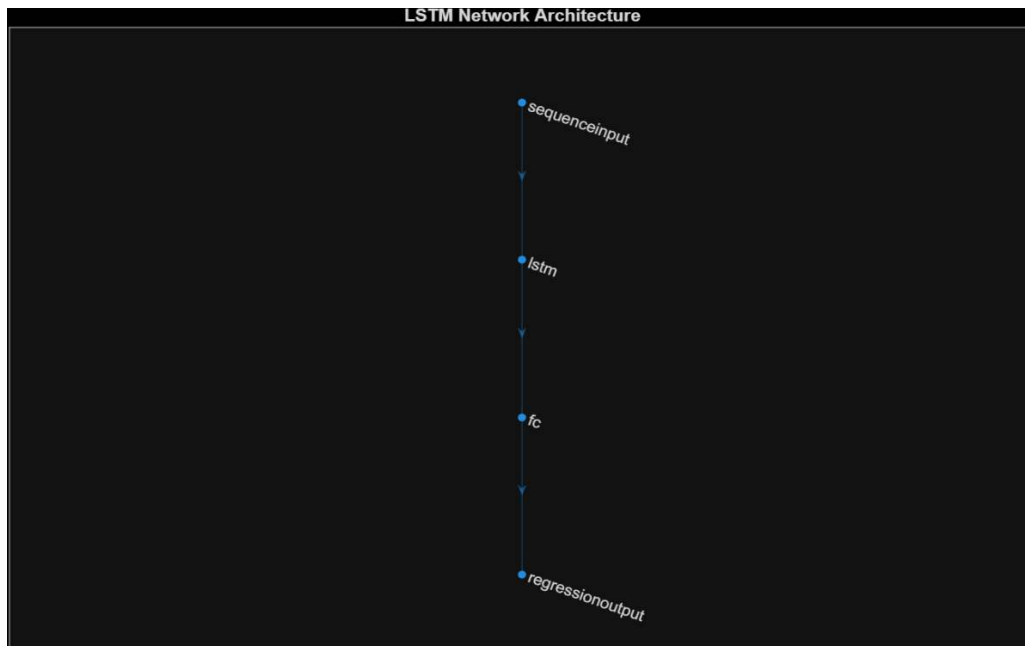
trainingOptions	Sets training parameters like optimizer, epochs, validation, etc.
analyzeNetwork	Opens GUI to analyze neural network layers
layerGraph, plot	Visualizes network layer connections
predict	Performs forward pass prediction using trained model
sqr, mean	Used to compute Root Mean Square Error (RMSE)

Important Variables & Their Roles

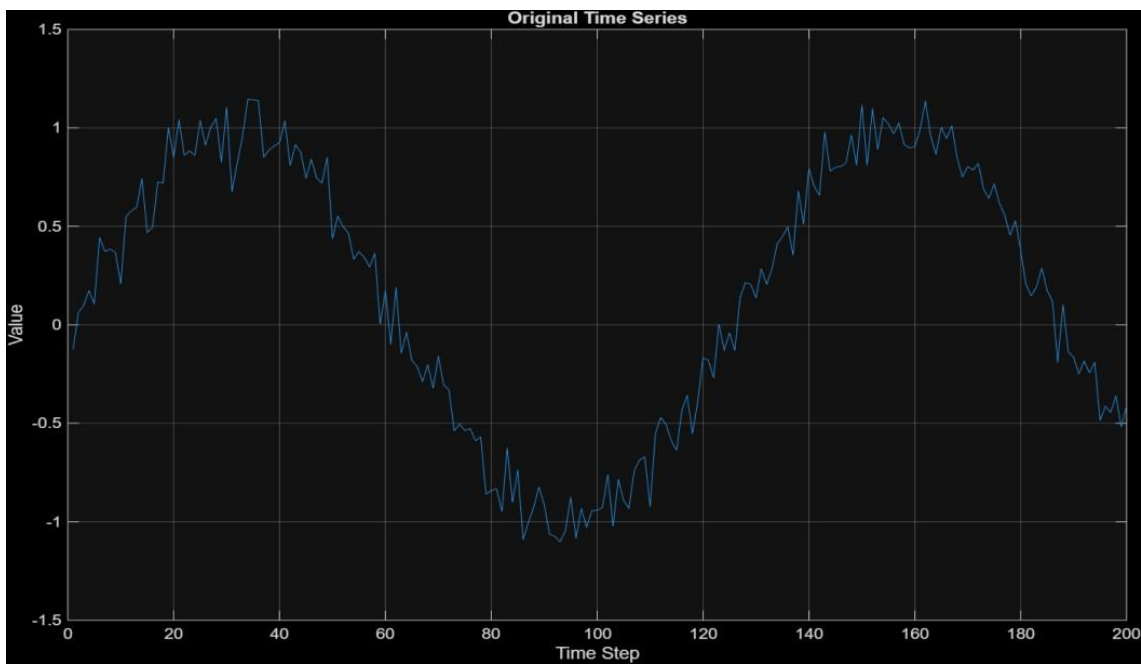
Variable	Role
data	Simulated noisy sine time series
t	Time steps vector (1 to 200)
X, Y	Input-output pairs for time series prediction (cell arrays)
XTrain, YTrain	Training data (80%)
XVal, YVal	Validation data (10%)
XTest, YTest	Test data (10%)
layers	Neural network architecture
net	Trained LSTM model
YPredVal	Model predictions on validation set
YPredTest	Model predictions on test set
rmseVal, rmseTest	Performance evaluation metrics (error values)
options	Training configuration
numHiddenUnits	Number of neurons in LSTM layer
numFeatures	Input sequence size (1, since univariate)

Layers Information

LAYER INFORMATION					
	Name	Type	Activations S (Spatial), T (Time), C (Channel), B (...)	Learnable Sizes	State Sizes
1	sequenceinput Sequence input with 1 dimensions	Sequence Input	1(C) × 1(B) × 1(T)	-	-
2	lstm LSTM with 100 hidden units	LSTM	100(C) × 1(B) × 1(T)	InputWeigh. 400 × RecurrentW. 400 × Bias 400 ×	HiddenState 100 × 1 CellState 100 × 1
3	fc 1 fully connected layer	Fully Connected	1(C) × 1(B) × 1(T)	Weights 1 × 100 Bias 1 × 1	-



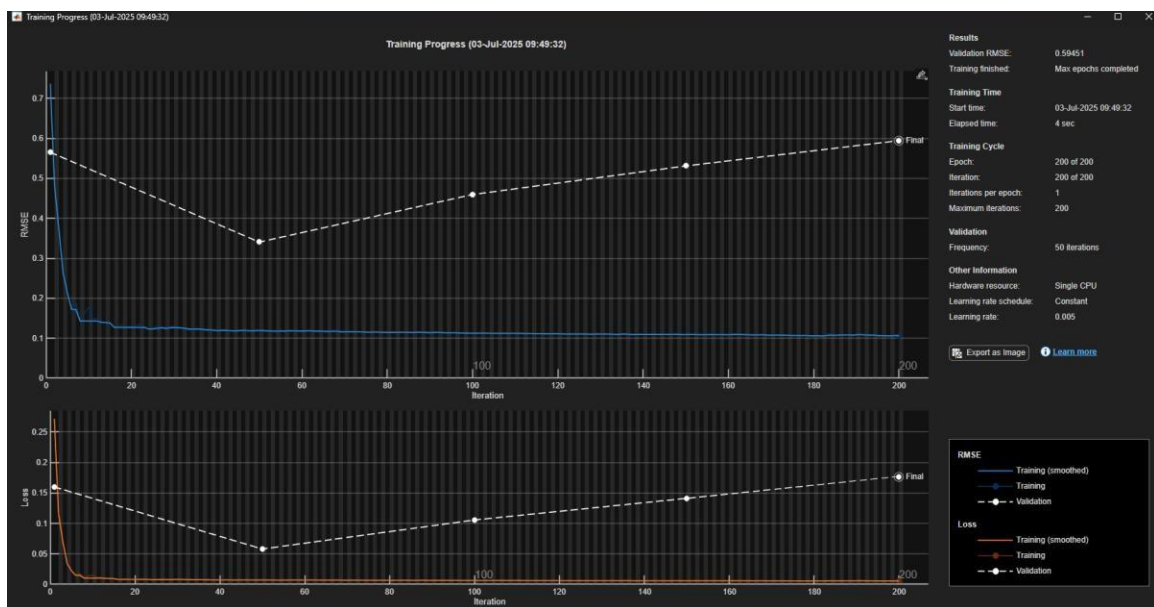
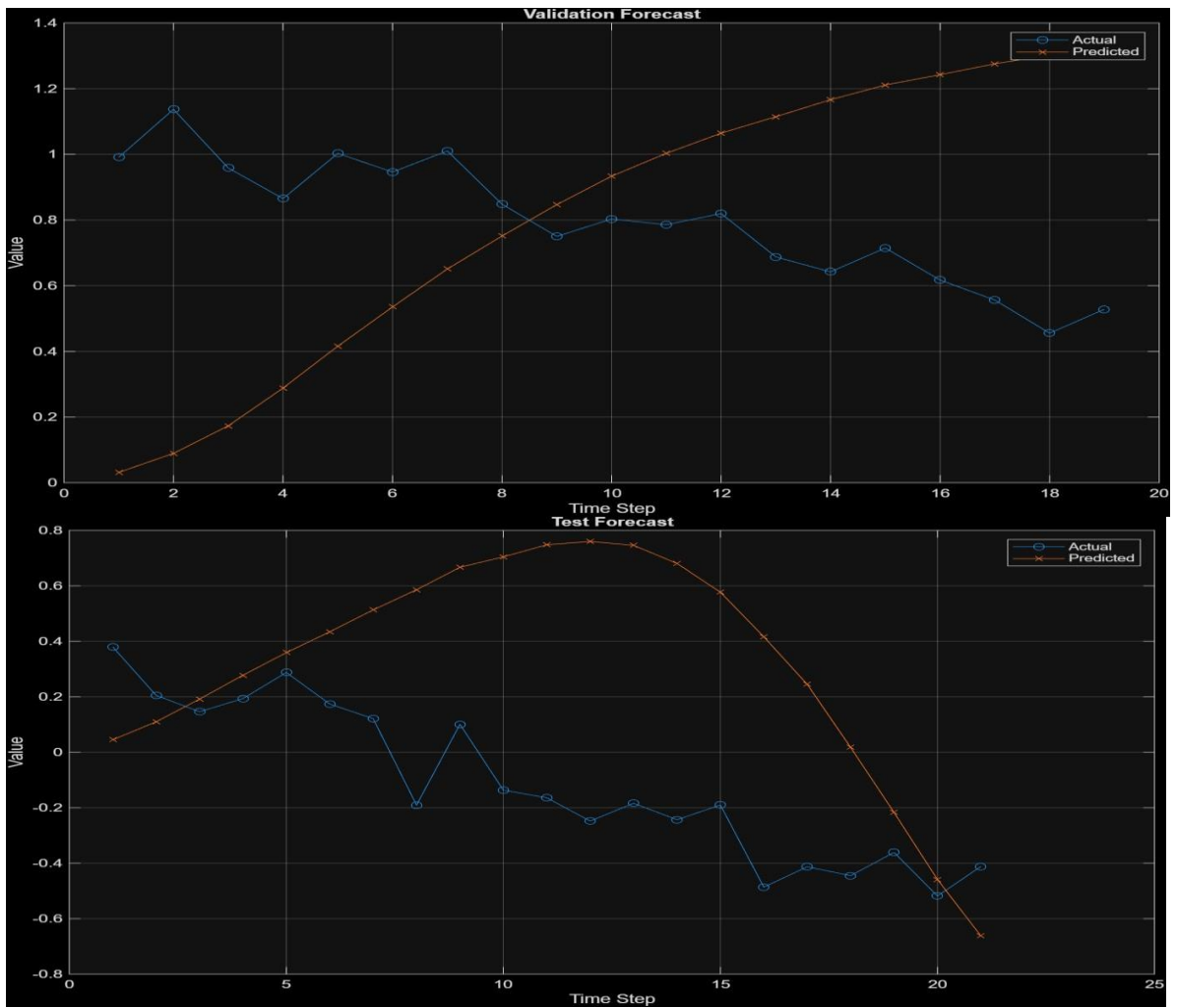
Input



Output

Validation RMSE: 0.5945

Test RMSE: 0.6052



Program 7:

Write a program to enable pre-train models to classify a given image dataset.

```
%% 1. Set Dataset Path
datasetPath = 'D:\RNSIT\AIML\2025-26_ODD_sem\Deep Learning\Lab
Programs\flowers'; % <<< Change this to your dataset folder

imds = imageDatastore(datasetPath, ...
    'IncludeSubfolders', true, ...
    'LabelSource', 'foldernames');

% Display label count
disp(countEachLabel(imds));

%% 2. Split into Training and Validation Sets
[imdsTrainRaw, imdsValRaw] = splitEachLabel(imds, 0.8, 'randomized');
numClasses = numel(categories(imdsTrainRaw.Labels));

%% 3. Resize & Augment Images
inputSize = [224 224 3];
imdsTrain = augmentedImageDatastore(inputSize, imdsTrainRaw);
imdsVal = augmentedImageDatastore(inputSize, imdsValRaw);

%% 4. Load Pre-trained ResNet-18
try
    net = resnet18();
catch
    error(['ResNet-18 not installed. Go to: Add-Ons > Get Add-Ons > ' ...
        '"Deep Learning Toolbox Model for ResNet-18 Network'.']);
end

%% 5. Modify Final Layers for Your Dataset
lgraph = layerGraph(net);
lgraph = removeLayers(lgraph, {'fc1000', 'prob',
    'ClassificationLayer_predictions'});

newLayers = [
    fullyConnectedLayer(numClasses, 'Name', 'fcNew')
    softmaxLayer('Name', 'softmax')
    classificationLayer('Name', 'output')
];

lgraph = addLayers(lgraph, newLayers);
lgraph = connectLayers(lgraph, 'pool5', 'fcNew');

%% 6. Training Options
options = trainingOptions('adam', ...
    'MiniBatchSize', 32, ...
    'MaxEpochs', 5, ...
    'Shuffle', 'every-epoch', ...
    'ValidationData', imdsVal, ...
    'ValidationFrequency', 20, ...
    'Verbose', false, ...
    'Plots', 'training-progress');
```

```

%% 7. Train the Network
trainedNet = trainNetwork(imdsTrain, lgraph, options);

%% 8. Evaluate Model on Entire Validation Set
YPred = classify(trainedNet, imdsVal);
YTrue = imdsValRaw.Labels;

% Accuracy
accuracy = mean(YPred == YTrue);
fprintf('Validation Accuracy: %.2f%%\n\n', accuracy * 100);

% Display per-class prediction count
classes = categories(YTrue);
confMat = confusionmat(YTrue, YPred);

fprintf('Prediction Counts Per Class:\n');
for i = 1:numel(classes)
    fprintf('Class: %-15s Correct: %-3d Total: %-3d\n', ...
        string(classes{i}), confMat(i,i), sum(confMat(i,:)));
end

%% 9. Confusion Matrix Plot
figure;
confusionchart(YTrue, YPred);
title('Confusion Matrix');

%% 10. Show One Sample per Class with Prediction
figure;
uniqueClasses = categories(YTrue);
shown = false(size(uniqueClasses));
shownCount = 0;

for i = 1:numel(imdsValRaw.Files)
    img = readimage(imdsValRaw, i);
    label = YTrue(i);
    pred = classify(trainedNet, imresize(img, inputSize(1:2)));

    idx = find(uniqueClasses == label);
    if ~shown(idx)
        shownCount = shownCount + 1;
        subplot(ceil(numel(uniqueClasses)/3), 3, shownCount);
        imshow(img);
        title(sprintf('True: %s\nPred: %s', string(label), string(pred)));
        shown(idx) = true;
    end

    if all(shown)
        break;
    end
end
end

```

Step-wise Description

Step	Description	Functions / Commands	Variables Used	Notes
1	Set dataset path and load images using labels from folder names	imageDatastore()	datasetPath, imds	Automatically labels images using folder names
2	Split data into training and validation sets	splitEachLabel()	imdsTrainRaw, imdsValRaw	80% train, 20% validation
3	Resize and prepare images using augmentation pipeline	augmentedImageDatastore()	inputSize, imdsTrain, imdsVal	Prepares images for ResNet input size
4	Load pre-trained ResNet-18 model	resnet18()	net	Requires Deep Learning Toolbox Add-On
5	Modify the final layers to match your dataset classes	layerGraph(), removeLayers(), addLayers()	lgraph, newLayers, numClasses	Replaces default 1000-class layer
6	Define training options	trainingOptions()	options	Uses Adam optimizer and training settings
7	Train the network	trainNetwork()	trainedNet	Trains using training datastore
8	Predict and calculate accuracy on validation data	classify(), mean()	YPred, YTrue, accuracy	Also displays per-class stats
9	Visualize performance using confusion matrix	confusionchart(), confusionmat()	confMat, classes	Useful to analyze prediction strengths
10	Display one image per class with true and predicted label	readimage(), classify(), imshow()	shown, label, pred	Helpful for qualitative validation

Key Functions & Their Use

Function/Command	Purpose
imageDatastore()	Loads images with labels based on folder structure
splitEachLabel()	Randomly splits datastore into training and validation sets
augmentedImageDatastore()	Resizes and preprocesses images on-the-fly
resnet18()	Loads the pre-trained ResNet-18 model
layerGraph(), addLayers()	Modifies deep learning network architecture
trainingOptions()	Sets hyperparameters and training configurations
trainNetwork()	Trains the model using the given layers and options
classify()	Predicts labels for input images
mean()	Calculates overall prediction accuracy
confusionmat(), confusionchart()	Generates confusion matrix for evaluating classification results
readimage()	Reads a specific image from datastore
imshow()	Displays an image in MATLAB figure

Important Variables & Their Roles

Variable	Role
datasetPath	File path to the dataset containing images organized by class
imds	Full image datastore with labels
imdsTrainRaw	Training subset of the datastore
imdsValRaw	Validation subset of the datastore
inputSize	Target input size expected by ResNet (224×224×3)
net	Original ResNet-18 model loaded from MATLAB
lgraph	Modified network architecture (layer graph)
newLayers	Custom layers for replacing ResNet's original final layers
options	Training configuration used in <code>trainNetwork()</code>
trainedNet	Final trained model after fine-tuning
YPred	Predicted labels for validation images
YTrue	Actual ground truth labels from validation set
confMat	Confusion matrix values (True vs Predicted)

Output

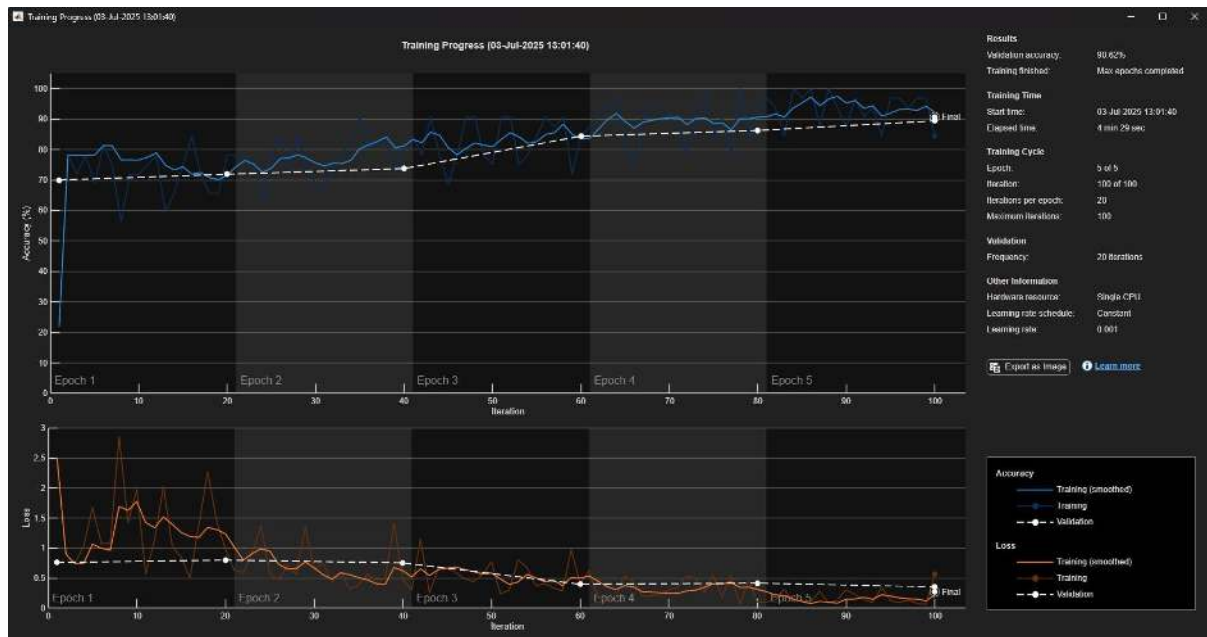
Label	Count
daisy	160
dandelion	160
rose	160
sunflower	160
tulip	160

Validation Accuracy: 90.62%

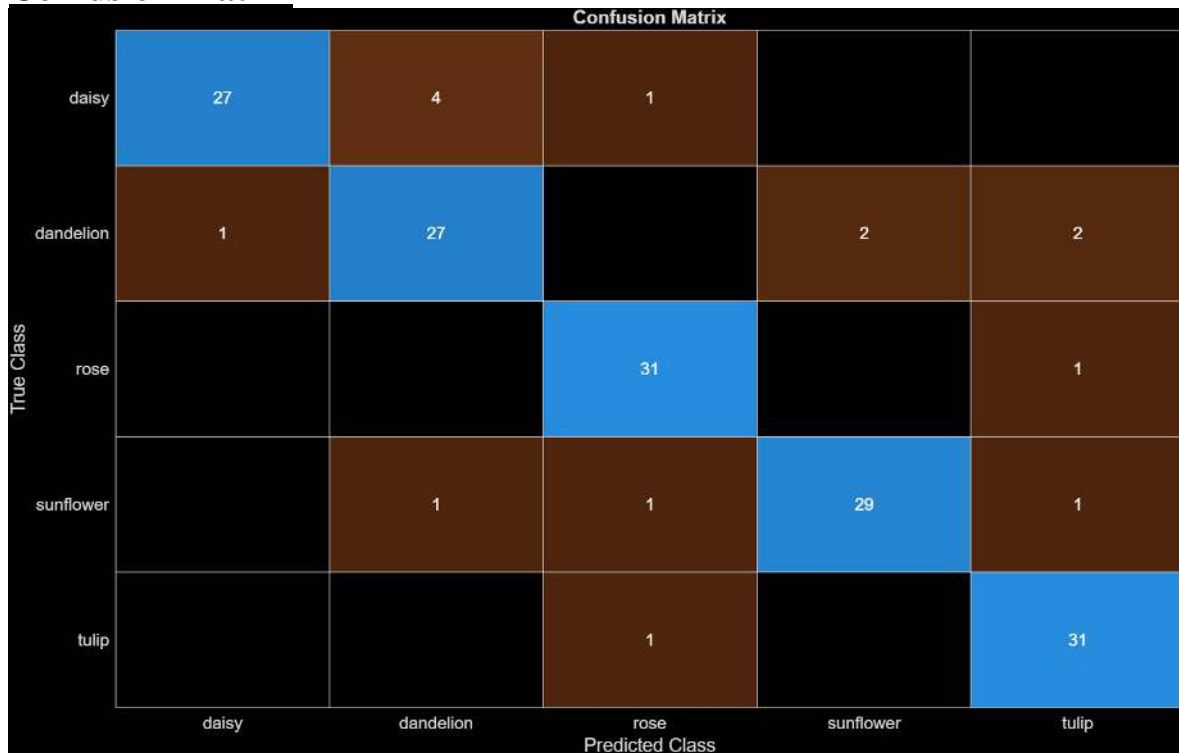
Prediction Counts Per Class:

Class: daisy	Correct: 27	Total: 32
Class: dandelion	Correct: 27	Total: 32
Class: rose	Correct: 31	Total: 32
Class: sunflower	Correct: 29	Total: 32
Class: tulip	Correct: 31	Total: 32

Training Progress



Confusion Matrix



Prediction Result

True: daisy
Pred: daisy



True: dandelion
Pred: dandelion



True: rose
Pred: rose



True: sunflower
Pred: sunflower



True: tulip
Pred: tulip



Program 8:

Simple Grid World Problem: Design a custom 2D grid world where the agent navigates from a start position to a goal, avoiding obstacles. **Environment:** Custom grid.

```
%% 1. Parameters
gridSize = [6, 6];           % Grid dimensions [Rows, Columns]
startPos = [1, 1];           % Starting position [row, col]
goalPos  = [6, 6];           % Goal position [row, col]

% Define obstacles as [row, col] pairs
obstacles = [2 2; 2 3; 3 3; 4 5; 5 2];

%% 2. Create Grid Matrix
gridWorld = ones(gridSize);   % 0: free space
for i = 1:size(obstacles,1)
    gridWorld(obstacles(i,1), obstacles(i,2)) = 2; % 2: obstacle
end
gridWorld(goalPos(1), goalPos(2)) = 3;           % 3: goal

%% 3. Display the Grid
figure;
imagesc(gridWorld);
colormap([1 1 1; 0 0 0; 0 1 0]); % white=free, black=obstacle, green=goal
axis equal tight;
xticks(1:gridSize(2)); yticks(1:gridSize(1));
grid on;
xlabel('Column'); ylabel('Row');
title('Custom 2D Grid World');

% Mark start and goal
hold on;
plot(startPos(2), startPos(1), 'ro', 'MarkerSize', 12, 'LineWidth', 2);
text(startPos(2)+0.1, startPos(1), 'Start', 'Color', 'r', 'FontSize', 10);
text(goalPos(2)+0.1, goalPos(1), 'Goal', 'Color', 'g', 'FontSize', 10);

%% 4. Simulate Agent Movement (Random Walk)
agentPos = startPos;
path = agentPos;

while ~isequal(agentPos, goalPos)
    pause(0.2); % For visualization

    % Moves: [Up, Down, Left, Right]
    moves = [ -1 0; 1 0; 0 -1; 0 1 ];
    nextMoves = agentPos + moves;

    % Validate moves (within bounds and not obstacle)
    valid = [];
    for i = 1:4
        r = nextMoves(i,1); c = nextMoves(i,2);
        if r >= 1 && r <= gridSize(1) && c >= 1 && c <= gridSize(2)
            if gridWorld(r,c) ~= 2
                valid = [valid; r c];
            end
        end
    end
```

```

        end
    end

    % No valid moves? Exit
    if isempty(valid)
        disp('Agent is stuck! No valid moves.');
```

break;

```

    end

    % Choose a random valid move (can replace with A* or Q-learning)
    nextPos = valid(randi(size(valid,1)), :);
    agentPos = nextPos;
    path = [path; agentPos];

    % Plot agent movement
    plot(agentPos(2), agentPos(1), 'bs', 'MarkerSize', 10, 'LineWidth', 1.5);
    drawnow;
end

% Plot the entire path
plot(path(:,2), path(:,1), 'r--', 'LineWidth', 2);
title('Agent Path to Goal');

%% 5. Print Agent Path
fprintf('\nAgent Path from Start to Goal:\n');
for i = 1:size(path,1)
    fprintf('Step %2d: Row = %d, Column = %d\n', i, path(i,1), path(i,2));
end

% Optional: save path to CSV
% writematrix(path, 'agent_path.csv');
```

Step-wise Description

Step	Description	Functions / Commands Used	Variables Used	Notes
1	Define the grid size, agent's start and goal positions, and obstacle locations	[] (assignment)	gridSize, startPos, goalPos, obstacles	Sets the environment dimensions and constraints
2	Initialize the grid with free spaces (1), obstacles (2), and goal (3)	ones, size, assignment	gridWorld, obstacles, goalPos	Prepares environment with special cells
3	Display the grid with color-coded cells for free space,	figure, imagesc, colormap, axis equal, xticks, yticks, xlabel,	gridWorld, startPos, goalPos	Visualizes the grid using imagesc. White = free, Black =

	obstacles, and the goal	ylabel, title, plot, text, grid on		obstacle, Green = goal, Red circle = start
4	Simulate agent movement using a random walk algorithm	pause, randi, isequal, for, if, plot, drawnow, disp, break	agentPos, moves, nextMoves, valid, nextPos, path, gridWorld	Agent moves randomly until it reaches goal or gets stuck
5	Print the entire path taken by the agent step-by-step	fprintf, size	path	Logs each position in the agent's trajectory
6 (optional)	Save the path to a CSV file for external analysis	writematrix (commented)	path	Export functionality if needed

Display the Grid – Detailed Description

Aspect	Explanation
Objective	Visually represent the 2D grid world using MATLAB plotting tools so that obstacles, free spaces, goal, and start positions are clearly visible.
Why It's Important	This step helps in debugging and understanding the environment layout. It makes the agent's movement more intuitive and observable as it progresses toward the goal.
Key Functions Used	figure, imagesc, colormap, axis, grid, xticks, yticks, xlabel, ylabel, plot, text
Grid Plot	imagesc(gridWorld); displays the 2D grid where each cell's color represents its value (free, obstacle, or goal).
Color Mapping	colormap([1 1 1; 0 0 0; 0 1 0]); assigns specific colors: white (free = 0), black (obstacle = -1), green (goal = 10).
Axis Formatting	axis equal tight; ensures cells are square and the figure fits tightly around the grid.
Grid Lines	xticks(1:gridSize(2)); and yticks(1:gridSize(1)); put tick marks for each grid column and row. grid on; enables grid lines.
Labeling Axes	xlabel('Column'); ylabel('Row'); adds axis labels for clarity.
Start & Goal Marker	plot(startPos(2), startPos(1), 'ro', ...) places a red circle at the start position. text(..., 'Start') and text(..., 'Goal') annotate the grid for clarity.

Key Functions & Their Use

Function/Command	Purpose
------------------	---------

ones	Initialize empty grid
imagesc	Display grid as image with scaled colors
colormap	Set color mapping for grid values
axis equal tight, xticks, yticks, grid on	Grid visualization formatting
plot	Mark agent's movement and positions
text	Add labels for Start and Goal
pause	Delay for visualization of movement
randi	Select random move from valid options
fprintf	Display steps in console
writematrix (<i>optional</i>)	Save path to CSV file

Important Variables & Their Roles

Variable	Role
gridSize	Defines the 2D grid dimensions (e.g., [6,6])
startPos	Starting position of agent
goalPos	Target goal position to reach
obstacles	Array of obstacle coordinates
gridWorld	Matrix representing the grid layout (0=free, -1=obstacle, 10=goal)
agentPos	Current position of the agent during simulation
path	Sequence of positions traversed by the agent
moves	Possible move directions (up, down, left, right)
valid	Valid moves filtered based on bounds and obstacle check
nextMoves	All possible next positions before filtering

Visual Example Interpretation

- White squares → Free path
- **Black squares → Obstacles (visible now)**
- Green square → Goal
- Red circle → Start

Output

Agent Path from Start to Goal:

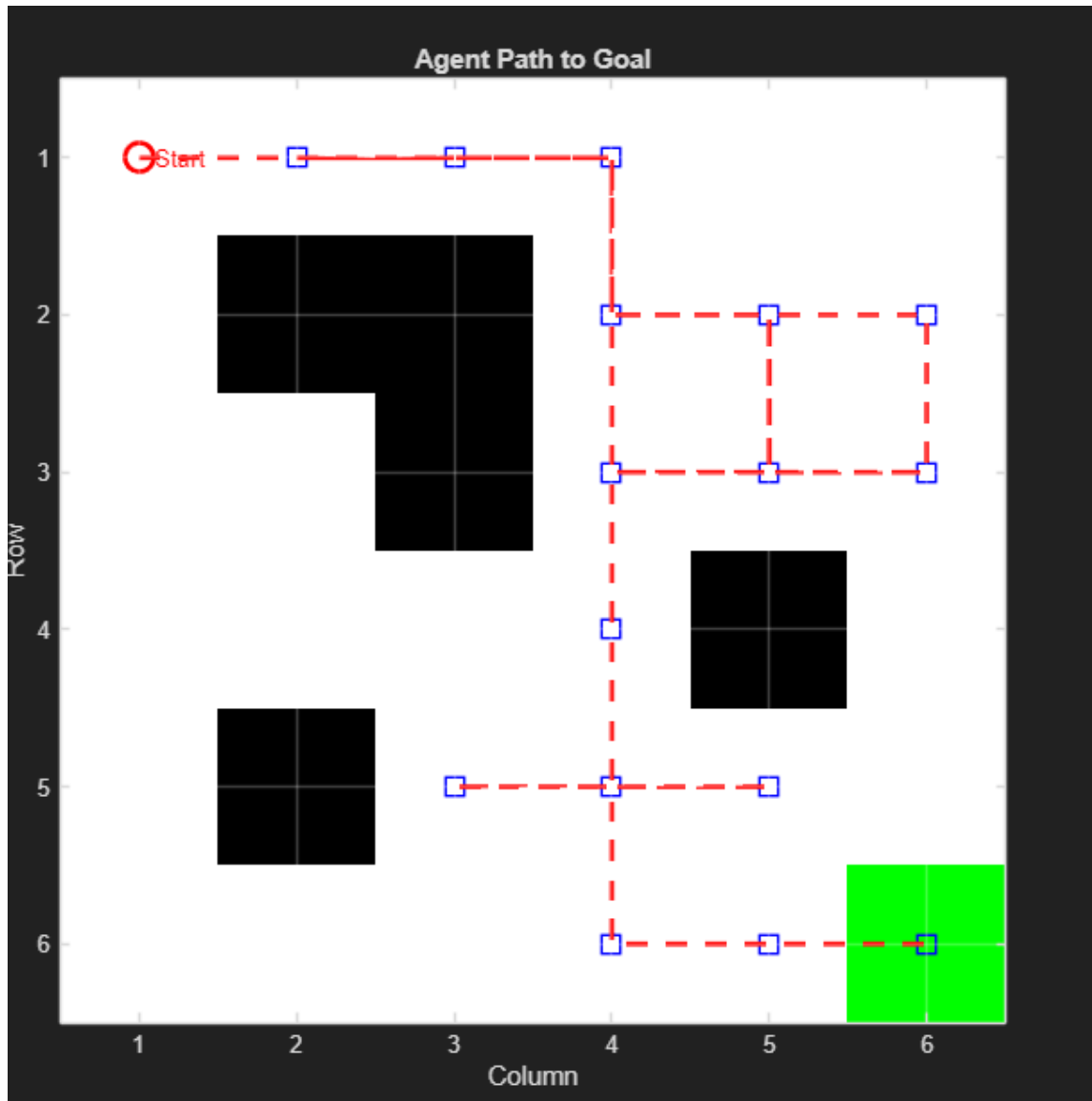
Step 1: Row = 1, Column = 1

Step 2: Row = 1, Column = 2

Step 3: Row = 1, Column = 3

•
•
•

Step 99: Row = 5, Column = 5
Step 100: Row = 5, Column = 6
Step 101: Row = 5, Column = 5
Step 102: Row = 5, Column = 6
Step 103: Row = 6, Column = 6



Sample Viva Questions

BASIC CONCEPTUAL QUESTIONS

1. What is word embedding?

Answer: Word embedding is a way of representing words as continuous vectors in a high-dimensional space, capturing their syntactic and semantic meanings.

2. What is the Skip-Gram model?

Answer: The Skip-Gram model is a neural network-based method that predicts context words given a center word, used for learning word embeddings.

3. Why do we use one-hot encoding in this lab?

Answer: One-hot encoding is used to represent words as binary vectors so they can be processed by the neural network.

4. What is the role of the embedding layer?

Answer: It transforms sparse one-hot vectors into dense vectors of fixed size, capturing relationships among words.

5. Why is softmax used in the final layer?

Answer: Softmax converts the output logits into probabilities across the vocabulary, used for classification.

6. What does the output of the embedding layer represent?

Answer: It represents the word's vector embedding in a lower-dimensional semantic space.

7. Why is context important in word embedding?

Answer: Context helps capture the semantic meaning of words based on how they are used in sentences.

8. What is the loss function used in this program?

Answer: Categorical cross-entropy loss is used, which compares the predicted distribution with the actual one-hot vector.

9. What is the window size in the Skip-Gram model?

Answer: It's the number of words to the left and right of the center word considered as context. In the program, it's 4.

10. What are stopwords? Why do we remove them?

Answer: Stopwords are common words (like “the”, “is”) that carry little meaning. Removing them helps reduce noise.

MATLAB-SPECIFIC / PROGRAMMING

11. What does the `containers.Map` do in MATLAB?

Answer: It creates a dictionary-like structure to map words to indices.

12. Why did we replace `accumarray` with `containers.Map`?

Answer: Because `accumarray` fails with large vocabularies; `containers.Map` is more memory-efficient and safer.

13. What does `featureInputLayer` do?

Answer: It defines the input size for the network, in this case, equal to the vocabulary size.

14. How are skip-gram pairs created in the code?

Answer: For each word, all surrounding context words within a defined window are paired with it.

15. What is the purpose of `ind2vec` function?

Answer: It converts integer class labels into one-hot encoded vectors.

16. What does the `trainNetwork` function return?

Answer: It returns the trained neural network model.

17. What are `XTrain` and `YTrain` in this code?

Answer: `XTrain` is the one-hot encoded input vectors, and `YTrain` is the corresponding categorical context words.

18. How is validation accuracy computed?

Answer: By comparing predicted labels (`YPredVal`) with actual validation labels (`YVal`).

19. How are embeddings saved after training?

Answer: By extracting the weights from the embedding layer and saving them using `save()`.

20. What is the dimension of each word vector in this implementation?

Answer: It is 100, as specified by the `embeddingDim` variable.

ADVANCED / EXTENSION QUESTIONS

21. How can you visualize word embeddings?

Answer: Using dimensionality reduction techniques like PCA or t-SNE to project embeddings to 2D/3D for plotting.

22. Can we use this model to find similar words?

Answer: Yes. By computing cosine similarity between word vectors, similar words can be identified.

23. What are negative samples in `word2vec`, and are they used here?

Answer: Negative sampling is a technique to speed up training by only updating a few output neurons. It is not used in this implementation.

24. What's the difference between Skip-Gram and CBOW?

Answer: Skip-Gram predicts context from the center word; CBOW predicts center word from the context.

25. What will happen if embedding dimension is too low?

Answer: The model may not capture enough semantic information, reducing performance.

26. What will happen if embedding dimension is too high?

Answer: It may overfit the data and increase computational cost.

27. What changes if you use a pre-trained embedding?

Answer: Training time reduces and performance may improve, especially with small datasets.

28. Can this model work for other languages?

Answer: Yes, provided text is tokenized and cleaned properly for that language.

29. What happens if you include very rare words in the vocabulary?

Answer: It increases sparsity and memory usage, and the model may not learn meaningful embeddings for them.

30. How can you improve this word embedding model further?

Answer: By using deeper layers, dropout, pre-trained embeddings, larger corpora, or switching to contextual embeddings like BERT.