INDEX

PREFACE

Deep learning Processing form the backbone of modern computing, enabling efficient solutions to complex problems in areas like optimization, networks, and data processing. This lab manual for M.Tech CSE I Year I Semester equips students with hands-on skills to implement and analyze key techniques—from brute-force enumeration to sophisticated string matching and flow algorithms—fostering deep insight into performance trade-offs.

The experiments align with core texts like S. Sridhar's *Design and Analysis of Algorithms* and references such as Cormen et al.'s *Introduction to Algorithms*, emphasizing practical coding in C/C++ alongside theoretical rigor. Through these 10 structured labs, students will not only code solutions but also evaluate time/space complexities, preparing them for real-world applications in IT infrastructure and systems design.

ACKNOWLEDGEMENT

It has been a rewarding experience working on the Deep learning Laboratory Manual. We would like to express our sincere gratitude to **Dr.K.Abdul Basith**, Professor and Head of the Department of Computer Science and Engineering, Marri Laxman Reddy Institute of Technology & Management, for his constant encouragement, valuable guidance, and continuous support in the preparation of this manual.

We are deeply indebted and gratefully acknowledge the support and motivation provided by the **Dr.P.Sridhar** Director, Marri Laxman Reddy Institute of Technology & Management, for granting us the opportunity and necessary resources to develop this laboratory manual.

Our heartfelt thanks to **Dr. R. Murali Prasad**, Principal, Marri Laxman Reddy Institute of Technology & Management, for his insightful suggestions, timely feedback, and academic guidance throughout the preparation of this document.

Finally, we extend our sincere appreciation to all the faculty members of the CSE Department whose encouragement, cooperation, and constructive inputs have played a significant role in helping us accomplish this work successfully.

COMPUTER SCIENCE AND ENGINEERING

DEEP LEARNING LAB MANUAL

GENERAL INSTRUCTIONS

- Students are instructed to attend the laboratory on time. Late comers will not be entertained in the lab.
- Students must be punctual. Experiments conducted during the session will not be repeated for those who are absent or late.
- Students are expected to come prepared with the theory and procedure of the experiment scheduled for the day.
- The use of mobile phones inside the lab is strictly prohibited.
- Any damage or loss of system components such as keyboard, mouse, or other peripherals during the lab session will be the responsibility of the student, and a fine or penalty will be imposed accordingly.
- Students must update their lab records and observation books session-wise. Before leaving the lab, the student should get their observation book signed by the concerned faculty member.
- Lab records must be submitted in the next lab session to the respective faculty members in the staffroom for correction and return.
- Students should not move around or disturb others during the lab session.
- In case of any emergency, students must obtain written permission from the concerned faculty member before leaving the laboratory.
- Faculty members reserve the right to suspend any student from the lab session on disciplinary grounds.

COMPUTER SCIENCE AND ENGINEERING
DEEP LEARNING LAB MANUAL

INSTITUTE VISION AND MISSION

Vision:

To be as an ideal academic institution by graduating talented engineers to be ethically strong, competent with quality research and technologies.

Mission:

- Utilize rigorous educational experiences to produce talented engineers
- Create an atmosphere that facilitates the success of students
- Programs that integrate global awareness, communication skills and Leadership qualities
- Education and Research partnership with institutions and industries to prepare the students for interdisciplinary research

COMPUTER SCIENCE AND ENGINEERING

DEEP LEARNING LAB MANUAL

DEPARTMENT VISION AND MISSION

Vision:

To empower the students to be technologically adept, innovative, self-motivated and responsible global citizen possessing human values and contribute significantly towards high quality technical education with ever changing world.

Mission:

- To offer high-quality education in the computing fields by providing an environment where the knowledge is gained and applied to participate in research, for both students and faculty.
- To develop the problem solving skills in the students to be ready to deal with cutting edge technologies of the industry.
- To make the students and faculty excel in their professional fields by inculcating the communication skills, leadership skills, team building skills with the organization of various co-curricular and extra- curricular programs.
- To provide the students with theoretical and applied knowledge, and adopt an education approach that promotes lifelong learning and ethical growth.

COMPUTER SCIENCE AND ENGINEERING

DEEP LEARNING LAB MANUAL

PROGRAM OUTCOMES

PO No.	NBA Statement / Vital Features	No. of Vital Features
PO1.	**An ability to independently carry out research /investigation and development work to solve practical problems.** 1. Research problems in Computer Science and Engineering are clearly identified and defined. 2. Literature review highlights research gaps and suitable methods. 3. Experiments or simulations are conducted using appropriate tools. 4. Data collection, analyses, and interpretation systematically. 5. Innovative approaches are applied to engineering problem-solving. 6. Results are validated against established theories and standards.	6
PO 2.	**An ability to write and present a substantial technical report/document** 1. Technical reports, dissertations, and papers are well-structured. 2. Referencing and academic integrity practices are properly maintained. 3. Content is presented with clarity, precision, and logical flow. 4. Oral communication and presentation skills are effectively demonstrated. 5. Digital tools are used for documentation and visualization. 6. Research findings are communicated to both technical and non-technical audience.	6
PO 3.	**Students should be able to demonstrate advanced proficiency in Computer Science and allied emerging areas of Engineering.** i. In-Depth Technical Knowledge ii. Advanced Problem-Solving and Algorithmic Skills. iii. Hands-On Practical Expertise iv. Research and Innovation Aptitude v. Interdisciplinary Integration	5
PO 4.	**Students should be able to identify, analyze, and effectively solve complex real-world problems by applying advanced computing concepts, while considering solutions from**	

	a global perspective i. Ability to break down multifaceted problems into manageable parts and develop effective solutions using advanced computing techniques. ii. Analytical Thinking and Critical Reasoning iii. Advanced Computing Knowledge. iv. Capability to understand and model complex systems, considering interdependencies and dynamic behaviors within global contexts. v. Global and Societal Awareness. vi. Aptitude for developing novel approaches and innovative solutions to address complex challenges. vii. Ability to work effectively in diverse teams and integrate knowledge from multiple disciplines to solve global problems. viii. Skill in clearly presenting problem analyses, solutions, and their implications to technical and non-technical global audience.	**8**
PO 5.	**An ability to acquire and apply advanced technical knowledge, professional skills, and modern computing tools to develop sustainable solutions** i. Ability to continuously learn and apply cutting-edge technical knowledge in computing and related fields. ii. Proficiency with Modern Computing Tools. iii. Capability to design and implement solutions that are environmentally, and economically feasible. iv. Understanding of professional ethics and responsibility in creating technology solutions with long-term positive impact. v. Integration of Multidisciplinary Knowledge. vi. Adaptability to sustainable practices.	**6**
PO 6.	**An Ability to recognize the significance of lifelong learning and actively pursue continuous professional development by adapting technologies in emerging areas.** i. Self-Directed Learning. ii. Skill in quickly learning and integrating new technologies and tools as they emerge in the field. iii. Capability to assess the relevance and impact of emerging technologies and decide on their applicability. iv. Continuous Professional Development Planning. v. Ability to engage with professional communities, attend workshops, and collaborate to stay updated. vi. Habit of regularly reflecting on personal growth, learning experiences, and professional competencies to improve continuously.	**6**

MARRI LAXMAN REDDY
INSTITUTE OF TECHNOLOGY AND MANAGEMENT
(AN AUTONOMOUS INSTITUTION)
(Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad)
Accredited by NBA and NAAC with 'A' Grade & Recognized Under Section2(f) & 12(B)of the UGC act,1956

COMPUTER SCIENCE AND ENGINEERING

DEEP LEARNING LAB MANUAL

PROGRAM EDUCATIONAL OBJECTIVES

Sl. No.	PEOs Name	Program Education Objective Statements
1	PEO – 1	Graduates will achieve professional excellence and success in the field of Computer Science and Engineering by applying strong technical foundations and problem-solving skills to contribute effectively to industry, academia, and entrepreneurship.
2	PEO – 2	Graduates will demonstrate a commitment to lifelong learning by continuously enhancing their knowledge and skills through professional development and self-directed learning to effectively adapt to evolving global challenges.
3	PEO – 3	Graduates of the Computer Science and Engineering program will actively pursue advanced research, contributing to the development of solutions for complex problems and the generation of new knowledge to effectively address real-world challenges.
4	PEO – 4	Graduates will exhibit professionalism, effective communication, leadership skills, and ethical responsibility while working in multidisciplinary teams to deliver computing solutions that address societal needs and contribute to sustainable development.

COMPUTER SCIENCE AND ENGINEERING

DEEP LEARNING LAB MANUAL

PROGRAM SPECIFIC OUTCOMES

PSO1: Applications of Computing: Ability to use knowledge in various domains to provide solution to new ideas and innovations.

PSO2: Programming Skills: Identify required data structures, design suitable algorithms, develop and maintain software for real world problems.

COMPUTER SCIENCE AND ENGINEERING

DEEP LEARNING LAB MANUAL

Course Structure

DEEP LEARNING Lab will have a continuous evaluation during II semester for 40 sessional marks and 60 end semester examination marks.Out of the 40 marks for internal evaluation, day-to-day work in the laboratory shall be evaluated for 20 marks and internal practical examination shall be evaluated for 10 marks conducted by the laboratory teacher concerned.

The end semester examination shall be conducted with an external examiner and internal examiner. The external examiner shall be appointed by the principal / Chief Controller of examinations

COMPUTER SCIENCE AND ENGINEERING

DEEP LEARNING LAB MANUAL

Course Objectives:

1. To Build the Foundation of Deep Learning.
2. To Understand How to Build the Neural Network.
3. To enable students to develop successful machine learning concepts.

Course Outcomes:

1. Implement Python programs and set up IDE environments to execute deep learning workflows for computer vision and NLP tasks.
2. Make use of deep learning libraries such as TensorFlow, Keras, and PyTorch to build and train neural network models for real-world applications.
3. Design convolutional neural networks (CNNs) for image classification tasks such as MNIST digit recognition and other computer vision problems.
4. Utilize recurrent neural network (RNN) models with LSTM/GRU layers to perform sentiment analysis and other sequence modelling tasks in NLP applications.
5. Implement autoencoders and generative adversarial networks (GANs) to perform encoding, image generation, and unsupervised learning tasks in real-world datasets

DEEP LEARNING LAB (LAB - II)

M.Tech CSE/CS I Year I Sem.

L	T	P	C
0	0	4	2

Course Objectives:

4. To Build the Foundation of Deep Learning.
5. To Understand How to Build the Neural Network.
6. To enable students to develop successful machine learning concepts.

Course Outcomes:

6. Implement Python programs and set up IDE environments to execute deep learning workflows for computer vision and NLP tasks.
7. Make use of deep learning libraries such as TensorFlow, Keras, and PyTorch to build and train neural network models for real-world applications.
8. Design convolutional neural networks (CNNs) for image classification tasks such as MNIST digit recognition and other computer vision problems.
9. Utilize recurrent neural network (RNN) models with LSTM/GRU layers to perform sentiment analysis and other sequence modelling tasks in NLP applications.
10. Implement autoencoders and generative adversarial networks (GANs) to perform encoding, image generation, and unsupervised learning tasks in real-world datasets.

LIST OF EXPERIMENTS:

1. Setting up the Spyder IDE Environment and Executing a Python Program
2. Installing Keras, Tensorflow and Pytorch libraries and making use of them
3. Applying the Convolution Neural Network on computer vision problems
4. Image classification on MNIST dataset (CNN model with Fully connected layer)
5. Applying the Deep Learning Models in the field of DEEP LEARNING
6. Train a sentiment analysis model on IMDB dataset, use RNN layers with LSTM/GRU notes
7. Applying the Autoencoder algorithms for encoding the real-world data
8. Applying Generative Adversial Networks for image generation and unsupervised tasks.

TEXT BOOKS:

1. Deep Learning by Ian Goodfellow, Yoshua Bengio and Aaron Courville, MIT Press.
2. The Elements of Statistical Learning by T. Hastie, R. Tibshirani, and J. Friedman, Springer.
3. Probabilistic Graphical Models. Koller, and N. Friedman, MIT Press.

REFERENCES:

1. Bishop, C, M., Pattern Recognition and Machine Learning, Springer, 2006.
2. Yegnanarayana, B., Artificial Neural Networks PHI Learning Pvt. Ltd, 2009.
3. Golub, G.,H., and Van Loan, C.,F., Matrix Computations, JHU Press,2013.
4. Satish Kumar, Neural Networks: A Classroom Approach, Tata McGraw-Hill Education, 2004.

Extensive Reading:

1. http://www.deeplearning.net
2. https://www.deeplearningbook.org/
3. https://developers.google.com/machine-learning/crash-course/ml-intro
4. www.cs.toronto.edu/~fritz/absps/imagenet.pdf
5. http://neuralnetworksanddeeplearning.com/

Experiment 1

Setting up the Spyder IDE and Executing a Python Program

Installing Spyder using Anaconda

One of the easiest ways to install Spyder IDE is to use Anaconda, a popular Python distribution with many pre-installed packages, including Spyder IDE.

The following steps outline how to install Spyder IDE using Anaconda:

- Download the Anaconda distribution for your operating system from the official website: https://www.anaconda.com/products/individual
- Install Anaconda by following the installation wizard.
- Once installed, open the Anaconda Navigator application.
- Click on the "Environments" tab in the left panel and select the "base (root)" environment.
- In the "base (root)" environment, search for "spyder" in the search bar.
- Click on the "Install" button next to "spyder" to install the latest version of Spyder IDE.

Installing Spyder using pip

- Open the terminal/command prompt on your system.
- Type the following command to install Spyder using pip install Spyder
- Once installed, you can open Spyder IDE by typing the following command in the terminal/command prompt:spyder

Installing Spyder using the official website

Download and install Spyder IDE from the official website: https://www.spyder-ide.org/. The following steps outline how to install Spyder IDE using the official website:

- Go to the official Spyder IDE website: https://www.spyder-ide.org/.
- Choose "Download" from the top menu's button selection.
- Choose the right download link for your operating system.
- Follow the installation wizard to install Spyder IDE.

Running Python scripts in Spyder IDE

An environment for writing, testing, and debugging Python code is provided by the Spyder IDE. The following steps outline how to create a new Python script and run it in Spyder IDE:

- Open the Spyder IDE application.
- To create a new Python script, click the "New file" button on the toolbar or choose "File" > "New file."
- Fill out the new file using your Python code.
- Save the file by going to "File" > "Save as" and give it a proper name with a .py extension.
- Click on the "Run" button on the toolbar or go to "Run" > "Run the file" to run your Python script.
- If your script has any errors, Spyder IDE will show the error message."
- Spyder IDE has a console where you can see the output of your Python code. By selecting the "Console" tab at the bottom of the Spyder IDE window, you can access the console.
- You can interact with the console and execute individual lines of code by typing them directly into the console.
- The Spyder IDE debugger can be used to debug your code. To start the debugger, go to "Run" > "Debug file". This will open the debugger window.

VIVA QUESTIONS

1) What is Spyder and why is it used for Python programming?

2) How do you install Spyder on your system?

3) What are the key components of the Spyder IDE interface?

4) How can you configure the Python interpreter in Spyder?

5) How do you create a new Python file in Spyder?

6) What are the different ways to execute a Python script in Spyder?

7) How can you debug a Python program in Spyder?

8) What is the role of the Variable Explorer in Spyder?

9) How do you manage different Python environments or dependencies in Spyder?

10) How do you integrate Spyder with version control systems like Git?

Experiment 2

Installing Keras, Tensor flow and Pytorch libraries and making use of them

Keras and Tensor Flow are open source Python libraries for working with neural networks, creating machine learning models and performing deep learning. Because Keras is a high level API for TensorFlow, they are installed together.

In general, there are two ways to install Keras and TensorFlow:

- ❖ Install a Python distribution that includes hundreds of popular packages (including Keras andTensorFlow)such as ActivePython.(https://www.activestate.com/products/python/)
- ❖ Use pip to install TensorFlow, which will also install Keras at the same time.

Pip Install TensorFlow:

Instead of pip installing each package separately, the recommended approach is to install Keras as part of the TensorFlow installation. When you install TensorFlow 2.0+, Keras will be automatically installed, as well.

The simplest way to install TensorFlow is to install the binary version using one of the official releases on the Python Package Index (PyPI).

TensorFlow can be run on three different processor platforms, with the main difference being the speed at which your neural network will be trained. Each platform has different hardware requirements and offers different performance:

- CPU – any modern computer can run this version, but it offers the slowest training speeds.
- TPU – only available currently on Google's Colaboratory (Colab) platform, Tensor Processing Units (TPUs) offer the highest training speeds.
- GPU – most high end computers feature a separate Graphics Processing Unit (GPU) from Nvidia or AMD that offer training speeds much faster than CPUs, but not as fast as TPUs.

TensorFlow Requirements:

TensorFlow and Keras require Python 3.6+ (Python 3.8 requires TensorFlow 2.2+) , and the latest version of pip. You can determine the version of Python installed on your computer by running the following command:

python3 --version

Output should be similar to: Python 3.8.2

Run the following command to ensure that the latest version of pip is installed:

pip install --upgrade pip

To install TensorFlow for CPU and GPU processors, run the following command:

pip install tensorflow

If you're fine with using the CPU to train your neural network, your installation is done. If you want to use your GPU to the training, you'll need to do the following:

- For AMD GPUs, refer to the article Install Tensorflow 2 for AMD GPUs
- For Nvidia GPUs:
    1. Ensure you're running a CUDA®-enabled card
    2. Install v11 or later of the CUDA® Toolkit
    3. If you're working with Deep Neural Networks, you'll should also install the latest version of the cuDNN library

The installation installs a slew of TensorFlow and Keras dependencies:

tensorflow

├── absl-py~=0.10

│ └── six

├── astunparse~=1.6.3

│ ├── six<2.0,>=1.6.1

│ └── wheel<1.0,>=0.23.0

├── flatbuffers~=1.12.0

├── gast==0.3.3

├── google-pasta~=0.2

```
│   └── six
├── grpcio~=1.32.0
│   └── six>=1.5.2
├── h5py~=2.10.0
│   ├── numpy>=1.7
│   └── six
├── keras-preprocessing~=1.1.2
│   ├── numpy>=1.9.1
│   └── six>=1.9.0
├── numpy~=1.19.2
├── opt-einsum~=3.3.0
│   └── numpy>=1.7
├── protobuf>=3.9.2
│   └── six>=1.9
├── six~=1.15.0
├── tensorboard~=2.4
│   ├── absl-py>=0.4
│   │   └── six
│   ├── google-auth-oauthlib<0.5,>=0.4.1
│   │   ├── google-auth>=1.0.0
│   │   │   ├── cachetools<5.0,>=2.0.0
│   │   │   ├── pyasn1-modules>=0.2.1
│   │   │   │   └── pyasn1<0.5.0,>=0.4.6
│   │   │   ├── rsa<5,>=3.1.4
│   │   │   │   └── pyasn1>=0.1.3
```

```
│   │   │   ├── setuptools>=40.3.0
│   │   │   └── six>=1.9.0
│   │   └── requests-oauthlib>=0.7.0
│   │       ├── oauthlib>=3.0.0
│   │       └── requests>=2.0.0
│   │           ├── certifi>=2017.4.17
│   │           ├── chardet<5,>=3.0.2
│   │           ├── idna<3,>=2.5
│   │           └── urllib3<1.27,>=1.21.1
│   ├── google-auth<2,>=1.6.3
│   │   ├── cachetools<5.0,>=2.0.0
│   │   ├── pyasn1-modules>=0.2.1
│   │   │   └── pyasn1<0.5.0,>=0.4.6
│   │   ├── rsa<5,>=3.1.4
│   │   │   └── pyasn1>=0.1.3
│   │   ├── setuptools>=40.3.0
│   │   └── six>=1.9.0
│   ├── grpcio>=1.24.3
│   │   └── six>=1.5.2
│   ├── markdown>=2.6.8
│   ├── numpy>=1.12.0
│   ├── protobuf>=3.6.0
│   │   └── six>=1.9
│   ├── requests<3,>=2.21.0
│   │   ├── certifi>=2017.4.17
```

```
│   │   ├── chardet<5,>=3.0.2
│   │   ├── idna<3,>=2.5
│   │   └── urllib3<1.27,>=1.21.1
│   ├── setuptools>=41.0.0
│   ├── six>=1.10.0
│   ├── tensorboard-plugin-wit>=1.6.0
│   ├── werkzeug>=0.11.15
│   └── wheel>=0.26
├── tensorflow-estimator<2.5.0,>=2.4.0
├── termcolor~=1.1.0
├── typing-extensions~=3.7.4
├── wheel~=0.35
└── wrapt~=1.12.1
```

Update Tensorflow and Keras Using Pip

If you already have TensorFlow and Keras installed, they can be updated by running the following command:

- pip install -U tensorflow

You can verify the TensorFlow installation with the following command:

- python -m pip show tensorflow

Output should be similar to:

- Name: tensorflow
- Version: 2.2.0
- Summary: TensorFlow is an open source machine learning framework for everyone.
- Home-page: https://www.tensorflow.org/
- Author: Google Inc.
- Author-email: packages@tensorflow.org
- License: Apache 2.0
- Location: c:\python38\lib\site-packages

How to Import Keras and TensorFlow

Once TensorFlow and Keras are installed, you can start working with them.

 # Begin a Keras script by importing the Keras library:

import keras

or

from tensorflow import keras

 # Import TensorFlow:

import tensorflow as tf

It's not necessary to import all of the Keras and Tensorflow library functions. Instead, import just the function(s) you need for your project.

 # Import the Sequential model class from Keras

 # to form the framework for a Sequential neural network:

from keras.models import Sequential

VIVA QUESTIONS

1) How do you install TensorFlow in your Python environment?

2) What is the command to install Keras, and how does it relate to TensorFlow?

3) How do you install PyTorch, and what command would you use?

4) How do you import TensorFlow and check its version in a Python script?

5) How do you import Keras and define a simple neural network model using it?

6) How do you define and train a simple neural network model in PyTorch?

7) How would you save and load a model in TensorFlow/Keras?

8) How do you save and load a model in PyTorch?

9) What are some common use cases for TensorFlow, Keras, and PyTorch in machine learning and deep learning?

10) Compare TensorFlow/Keras and PyTorch in terms of their flexibility and ease of use.

EXPERIMENT 3

Implement Simple Programs like vector addition in Tensor Flow.

TensorFlow Basic Operations:

- Add
- Subtract
- Multiply
- Divide
- Square
- Reshape

1.Tensor Addition:

We can add two tensors using *tensorA.add(tensorB):*

```
const tensorA = tf.tensor([[1, 2], [3, 4], [5, 6]]);
const tensorB = tf.tensor([[1,-1], [2,-2], [3,-3]]);

// Tensor Addition
const tensorNew = tensorA.add(tensorB);
```

Output: [[2, 1], [5, 2], [8, 3]]

2.Tensor Subtract:

We can subtract two tensors using tensorA.sub(tensorB):

```
const tensorA = tf.tensor([[1, 2], [3, 4], [5, 6]]);
const tensorB = tf.tensor([[1,-1], [2,-2], [3,-3]]);

// Tensor Subtraction
const tensorNew = tensorA.sub(tensorB);
```

Output:[[0, 3], [1, 6], [2, 9]]

3.Tensor Multiplication:

We can multiply two tensors using tensorA.mul(tensorB):

```
const tensorA = tf.tensor([1, 2, 3, 4]);
const tensorB = tf.tensor([4, 4, 2, 2]);

// Tensor Multiplication
const tensorNew = tensorA.mul(tensorB);
```

Output: [4, 8, 6, 8]

4.Tensor Division:

We can divide two tensors using tensorA.div(tensorB):

```
const tensorA = tf.tensor([2, 4, 6, 8]);

const tensorB = tf.tensor([1, 2, 2, 2]);


// Tensor Division

const tensorNew = tensorA.div(tensorB);
```

Output: [2, 2, 3, 4]

VIVA QUESTIONS

1) What is the purpose of the tf.constant function in the provided code?

2) How does tf.add work in TensorFlow, and what is its role in the code?

3) What does the .numpy() method do in the context of TensorFlow tensors?

4) Why is the data type of the tensors specified as tf.float32, and what could be the implications of not specifying it?

5) How would you modify the code to perform vector addition with integer tensors instead of floating-point tensors?

6) What are the key differences between TensorFlow 1.x and TensorFlow 2.x when it comes to session management and tensor operations?

7) Explain the significance of TensorFlow's tensor operations in machine learning workflows.

8) How would you handle cases where the vectors have different lengths or shapes?

9) What are some potential use cases of TensorFlow’s vector operations in real-world applications?

10) How would you debug or troubleshoot issues if the tensor operations do not produce the expected results?

EXPERIMENT 4

Implement a simple problem like regression model in Keras

Introduction:

- Keras regression is the type of algorithm of supervised machine learning which was used to predict the label which was continuous.
- The goal of producing the model which was representing the best fit is to observe the data as per the evaluation criterion.
- The architecture of the neural network consists hidden layer, an input layer, and an output layer. In the regression, the main aim is for predicting the continuous value as output.

Overview of Keras Regression:

- Regression is enabling the continuous values in the keras dataset. The goal of regression is to produce the model which was representing the best fit in the observed data and on the evaluation criterion.
- The regression has three main components as follows:
- Input Layer: This layer is where our observation training will be fed. We need to specify the number of predictor variables by using neurons.
- Hidden Layer: This layer is nothing but the intermediate layer between output and input layers. The neural network will learn about the relationship which was involved in the component of data.
- Output Layer: In this layer, the final output is extracted which was happening in previous layers. If suppose we have observed any problem in regression then the output will contain the neuron.
- It is used to predict the values. While performing the classification our model will predict the values.

How to Use Keras with Regression?

The below steps show how we can use the keras with regression as follows. In the first step, we are importing all the required modules.

1.While using keras with regression in the first step we are importing all the modules which were required in keras regression. We are importing all the modules by using the import keyword as follows.

Code:

import keras

from sklearn import preprocessing

from sklearn.preprocessing import scale

Output:Keras Regression –Importing.

```
>>>
>>>
>>> import keras
>>> from keras.datasets import boston_housing
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> from keras.optimizers import RMSprop
>>> from keras.callbacks import EarlyStopping
>>> from sklearn import preprocessing
>>> from sklearn.preprocessing import scale
>>>
>>>
>>>
```

2. After importing the module in this step we are loading the data. We are loading the dataset name as boston_housing.

Code:

(x_train, y_train) … = boston_housing.load_data()

Output:Keras Regression - Loading

```
>>>
>>>
>>>
>>> (x_train, y_train), (x_test, y_test) = boston_housing.load_data()
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/boston_housing.npz
57026/57026 [==============================] - 0s 0us/step
>>>
>>>
>>>
```

3. After loading the dataset in this step we are processing the data as per our model as follows.

Code:

x_trai = preprocessing.scale(x_train)

sclr = preprocessing.StandardScaler().fit(x_train)

x_tes = sclr.transform(x_test)

Output:processing the data:

```
>>>
>>>
>>> x_trai = preprocessing.scale(x_train)
>>> sclr = preprocessing.StandardScaler().fit(x_train)
>>> x_tes = sclr.transform(x_test)
>>>
>>>
>>>
```

4. After loading the dataset now in this step we are creating the actual model of regression.

Code:

mod = Sequential()

mod.add(Dense(….))

mod.add(Dense(32, activation = 'relu'))

mod.add(Dense(1))

Output:Keras Regression - Creating

```
>>>
>>>
>>> mod = Sequential()
>>> mod.add(Dense(32, kernel_initializer = 'normal', activation = 'relu', input_shape = (13,)))
>>> mod.add(Dense(32, activation = 'relu'))
>>> mod.add(Dense(1))
>>>
```

5. After creating the model, now in this step we are compiling the model by using the loss function.

Code:

mod.compile (

loss = 'mse',

optimizer = RMSprop(),

metrics = [‘Error']

)

Output:compiling the model

```
>>>
>>> mod.compile(
...     loss = 'mse',
...     optimizer = RMSprop(),
...     metrics = ['error']
... )
>>>
>>>
```

6. After compiling the model in this step we are training the keras regression model.

Code:

his = mod.fit(

x_trai, y_train,

batch_size = 32,

epochs = 300,

verbose = 1,

validation_split = 0.3)

Output:

```
>>>
>>>
>>> his = mod.fit(
...     x_trai, y_train,
...     batch_size = 32,
...     epochs = 300,
...     verbose = 1,
...     validation_split = 0.3)
Epoch 1/300
```

7. After training the model now in this step we are predicting the model by using test data as follows.

Code:

```
pred = mod.predict(x_tes)

print(pred.flatten())

print(y_test)
```

Output:

```
>>>
>>> pred = mod.predict(x_tes)
4/4 [==============================] - 1s 6ms/step
>>> print(pred.flatten())
[-0.02613243 -0.03150093  0.00388797 -0.15836686  0.01196963 -0.02115891
 -0.05061442 -0.06047193 -0.01590732  0.18354216 -0.02463401  0.1048634
  0.10055219 -0.02132798  0.01755312 -0.0389421  -0.07903135 -0.08860285
  0.20760176  0.10447823  0.17851149 -0.13674062  0.00766724  0.05479153
 -0.01654451  0.12535214 -0.05044956  0.08275335 -0.0658207  -0.06760225
 -0.02703886 -0.03258924 -0.09384736 -0.07584611  0.10460332  0.17350915
 -0.17872201 -0.1342265   0.10667339  0.0355618  -0.01770161 -0.01040842
  0.17300722 -0.07473354  0.00299511 -0.00994735 -0.02626871 -0.01301824
  0.13475838 -0.02556396  0.06145026 -0.05021334  0.18924521  0.11257436
 -0.09905939  0.00650238  0.16617644 -0.12135176 -0.09290744  0.02574837
  0.08840709  0.14928383  0.02022238 -0.00693979 -0.00234615  0.00168027
  0.12881143 -0.05086273  0.15878248  0.18640935  0.10322411 -0.09141395
  0.22373706  0.16167687 -0.08801948 -0.0474164  -0.06101973 -0.0410192
 -0.08438651  0.19731079 -0.00809717 -0.16350505  0.0441207   0.15555078
  0.00753694  0.02975911  0.10143512 -0.01614413 -0.03521812  0.11212781
  0.03546121 -0.04874603 -0.03802439 -0.09594042 -0.08911604 -0.17138678
 -0.16008608 -0.02298233 -0.07229502 -0.1753512  -0.04225217 -0.1226924 ]
>>> print(y_test)
[ 7.2 18.8 19.  27.  22.2 24.5 31.2 22.9 20.5 23.2 18.6 14.5 17.8 50.
 20.8 24.3 24.2 19.8 19.1 22.7 12.  10.2 20.  18.5 20.9 23.  27.5 30.1
  9.5 22.  21.2 14.1 33.1 23.4 20.1  7.4 15.4 23.8 20.1 24.5 33.  28.4
 14.1 46.7 32.5 29.6 28.4 19.8 20.2 25.  35.4 20.3  9.7 14.5 34.9 26.6
  7.2 50.  32.4 21.6 29.8 13.1 27.5 21.2 23.1 21.9 13.  23.2  8.1  5.6
 21.7 29.6 19.6  7.  26.4 18.9 20.9 28.1 35.4 10.2 24.3 43.1 17.6 15.4
 16.2 27.1 21.4 21.5 22.4 25.  16.6 18.6 22.  42.8 35.1 21.5 36.  21.9
 24.1 50.  26.7 25. ]
>>>
```

VIVA QUESTIONS

1) What is the purpose of the Dense layer in the Keras model?

2) Why did you use the mean_squared_error loss function for this regression model?

3) How does the Adam optimizer work, and why is it chosen here?

4) Explain the significance of the input_dim parameter in the Dense layer.

5) What role does the epochs parameter play in the fit method?

6) How does the activation='linear' function impact the behavior of the Dense layer?

7) How would you modify this code to handle a multi-dimensional input feature space?

8) What is the purpose of using np.random.normal in generating y_train values?

9) What might you do if the model's performance (e.g., MSE) is not satisfactory?

10) How would you save and load this trained model for later use?

Experiment 5

Implement a perceptron in Tensor Flow / Keras Environment.

```
import tensorflow as tf
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Activation
import matplotlib.pyplot as plt
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
# Cast the records into float values
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
  # normalize image pixel values by dividing
# by 255
gray_scale = 255
x_train /= gray_scale
x_test /= gray_scale
print("Feature matrix:", x_train.shape)
```

```
print("Target matrix:", x_test.shape)
print("Feature matrix:", y_train.shape)
print("Target matrix:", y_test.shape)
fig, ax = plt.subplots(10, 10)
k = 0
for i in range(10):
    for j in range(10):
        ax[i][j].imshow(x_train[k].reshape(28, 28),
                        aspect='auto')
        k += 1
plt.show()
model = Sequential([
        # reshape 28 row * 28 column data to 28*28 rows
    Flatten(input_shape=(28, 28)),
         # dense layer 1
    Dense(256, activation='sigmoid'),
        # dense layer 2
    Dense(128, activation='sigmoid'),
         # output layer
    Dense(10, activation='sigmoid'),
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
model.fit(x_train, y_train, epochs=10,
        batch_size=2000,
        validation_split=0.2)
results = model.evaluate(x_test,  y_test, verbose = 0)
print('test loss, test acc:', results)
```

Output:

```
Epoch 1/10
24/24 [==============================] - 2s 43ms/step - loss: 2.0848 - accuracy: 0.3766 - val_loss: 1.7302 - val_accuracy: 0.65
64
Epoch 2/10
24/24 [==============================] - 1s 37ms/step - loss: 1.3903 - accuracy: 0.7211 - val_loss: 1.0328 - val_accuracy: 0.80
33
Epoch 3/10
24/24 [==============================] - 1s 28ms/step - loss: 0.8634 - accuracy: 0.8146 - val_loss: 0.6720 - val_accuracy: 0.86
03
Epoch 4/10
24/24 [==============================] - 1s 31ms/step - loss: 0.6047 - accuracy: 0.8661 - val_loss: 0.4967 - val_accuracy: 0.88
77
Epoch 5/10
24/24 [==============================] - 1s 28ms/step - loss: 0.4724 - accuracy: 0.8869 - val_loss: 0.4066 - val_accuracy: 0.89
93
Epoch 6/10
24/24 [==============================] - 1s 30ms/step - loss: 0.4006 - accuracy: 0.8977 - val_loss: 0.3559 - val_accuracy: 0.90
74
Epoch 7/10
24/24 [==============================] - 1s 26ms/step - loss: 0.3564 - accuracy: 0.9051 - val_loss: 0.3221 - val_accuracy: 0.91
47
Epoch 8/10
24/24 [==============================] - 1s 30ms/step - loss: 0.3256 - accuracy: 0.9116 - val_loss: 0.2989 - val_accuracy: 0.91
95
Epoch 9/10
24/24 [==============================] - 1s 29ms/step - loss: 0.3029 - accuracy: 0.9164 - val_loss: 0.2807 - val_accuracy: 0.92
33
Epoch 10/10
24/24 [==============================] - 1s 28ms/step - loss: 0.2847 - accuracy: 0.9205 - val_loss: 0.2662 - val_accuracy: 0.92
64
<tensorflow.python.keras.callbacks.History at 0x25dcad04048>
```

test loss, test acc: [0.27210235595703125, 0.9223999977111816]

VIVA QUESTIONS

1) What is the purpose of the Dense layer in the perceptron model?

2) Why did you use the sigmoid activation function in the perceptron model?

3) What is the role of binary_crossentropy as the loss function?

4) Explain the choice of the SGD optimizer for this model.

5) How does the input_dim parameter affect the Dense layer?

6) What does the StandardScaler do, and why is it used here?

7) What is the purpose of the validation_data parameter in the fit method?

8) How would you modify the code to handle multi-class classification instead of binary classification?

9) What is the significance of the epochs parameter in the fit method?

10) How would you assess whether the perceptron model is overfitting or underfitting?

EXPERIMENT 6

Implement a Feed-Forward Network in Tensor Flow/Keras.

```
import pandas as pd
import numpy as np
import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt


loans = pd.read_csv("loans.csv")
loans = loans[['created_at','amount']]
loans['created_at'] = pd.DatetimeIndex(loans['created_at'])
loans = loans.groupby(['created_at']).amount.sum().reset_index()
loans.sort_values(by=['created_at'], inplace=True)
loans = loans.set_index('created_at')


scaler = MinMaxScaler(feature_range=(0,1))
```

```
scaled_loans = scaler.fit_transform(loans)


x_train = []
y_train = []
for i in range(60, 1955):
    x_train.append(scaled_loans[i-60:i, 0])
    y_train.append(scaled_loans[i, 0])


y_train = np.array(y_train)
x_train = np.array(x_train)
x_train = np.reshape(x_train, (np.shape(x_train)[0], np.shape(x_train)[1], 1 ))


regressor  = Sequential()
regressor.add(LSTM(units=50, return_sequences=True, input_shape=(np.shape(x_train)[1],1)))
regressor.add(Dropout(0.2))


regressor.add(LSTM(units=50, return_sequences=True))
regressor.add(Dropout(0.2))


regressor.add(LSTM(units=50, return_sequences=True))
regressor.add(Dropout(0.2))


regressor.add(LSTM(units=50))
regressor.add(Dropout(0.2))
```

```
regressor.add(Dense(units=1))
regressor.add(Dropout(0.2))

regressor.add(Dense(units=1))

regressor.summary()

lr_schedule = keras.callbacks.LearningRateScheduler(lambda epoch:1e-7 * 10**(epoch/20))

opt = tensorflow.keras.optimizers.Adam(learning_rate=1e-7)

regressor.compile(optimizer=opt, loss='mse', metrics=['mae','mape'])

early_stopping = tf.keras.callbacks.EarlyStopping(monitor='loss', mode='min',patience=20)

mc = tf.keras.callbacks.ModelCheckpoint('best_model.h5', monitor='loss', mode='min',
verbose=0, save_best_only=True)

hist = regressor.fit(x_train,y_train, epochs=150, batch_size=32, callbacks=[mc, lr_schedule,
early_stopping])

prediction = regressor.predict( np.array( [x_train[0],]))
```

OUTPUT:

```
60/60 [==============================] - 23s 195ms/step - loss: 0.0013 - mae: 0.0126 - mape: 14997.7793 - lr: 1.0000e-07
Epoch 2/150
60/60 [==============================] - 12s 204ms/step - loss: 0.0013 - mae: 0.0126 - mape: 38157.1758 - lr: 1.1220e-07
Epoch 3/150
60/60 [==============================] - 16s 264ms/step - loss: 0.0013 - mae: 0.0125 - mape: 68237.7188 - lr: 1.2589e-07
Epoch 4/150
60/60 [==============================] - 13s 216ms/step - loss: 0.0013 - mae: 0.0125 - mape: 98987.6484 - lr: 1.4125e-07
Epoch 5/150
60/60 [==============================] - 13s 216ms/step - loss: 0.0013 - mae: 0.0125 - mape: 135317.9219 - lr: 1.5849e-07
Epoch 6/150
60/60 [==============================] - 14s 232ms/step - loss: 0.0013 - mae: 0.0125 - mape: 172366.9844 - lr: 1.7783e-07
Epoch 7/150
60/60 [==============================] - 13s 219ms/step - loss: 0.0013 - mae: 0.0124 - mape: 216284.2031 - lr: 1.9953e-07
Epoch 8/150
60/60 [==============================] - 13s 209ms/step - loss: 0.0013 - mae: 0.0124 - mape: 257962.3125 - lr: 2.2387e-07
Epoch 9/150
60/60 [==============================] - 12s 201ms/step - loss: 0.0013 - mae: 0.0123 - mape: 304780.3750 - lr: 2.5119e-07
Epoch 10/150
60/60 [==============================] - 12s 200ms/step - loss: 0.0013 - mae: 0.0123 - mape: 367604.4062 - lr: 2.8184e-07
```

```
y_preds = regressor.predict(x_train)
```

```
52] y_preds

    array([[0.00309422],
           [0.00310009],
           [0.00310448],
           ...,
           [0.03266592],
           [0.03250946],
           [0.03218953]], dtype=float32)
```

VIVA QUESTIONS

1) What is the role of the `Sequential` model in TensorFlow/Keras?

2) Explain the purpose of the `Dense` layer and how it is used in this example.

3) Why is the `relu` activation function used in the hidden layers?

4) What is the function of the `sigmoid` activation function in the output layer?

5) Why is the `binary_crossentropy` loss function used in this model?

6) What does the `Adam` optimizer do, and why is it used here?

7) Explain the purpose of the `batch_size` parameter in the `fit` method.

8) How does the `validation_data` parameter benefit the training process?

9) How would you modify this code to handle a multi-class classification problem?

10) What steps can you take if the model's performance is unsatisfactory?

EXPERIMENT 7

Applying the Convolution Neural Network on computer vision.

```
import numpy as np

import pandas as pd

import os

from pathlib import Path

import glob


import seaborn as sns

import matplotlib.pyplot as plt


import tensorflow as tf

from tensorflow.keras import layers

from tensorflow.keras import Model

from tensorflow.keras.optimizers import RMSprop

from keras_preprocessing.image import ImageDataGenerator


data_dir = Path('../input/cat-and-dog') # data directory

train_dir = data_dir / "training_set/training_set"

test_dir = data_dir / "test_set/test_set"
```

```
cat_samples_dir_train = train_dir / "cats" # directory for cats images
dog_samples_dir_train = train_dir / "dogs" # directory for dogs images
def make_csv_with_image_labels(CATS_PATH, DOGS_PATH):

    '''

    Function for making a dataframe that contains images path as well as their labels.

    Parameters:-
     - CATS_PATH - Path for Cats  Images
     - DOGS_PATH - Path for Dogs Images
cat_images = CATS_PATH.glob('*.jpg')
    dog_images = DOGS_PATH.glob('*.jpg')
    df = []
    for i in cat_images:
        df.append((i, 0)) # appending cat images as 0
    for j in dog_images:
        df.append((i, 1)) # appending dog images as 0
    df = pd.DataFrame(df, columns=["image_path", "label"], index = None) # converting into
dataframe
    df = df.sample(frac = 1).reset_index(drop=True)
    return df
train_csv = make_csv_with_image_labels(cat_samples_dir_train,dog_samples_dir_train)
train_csv.head()
len_cat = len(train_csv["label"][train_csv.label == 0])
```

```
len_dog = len(train_csv["label"][train_csv.label == 1])
arr = np.array([len_cat , len_dog])
labels = ['CAT', 'DOG']
print("Total No. Of CAT Samples :- ", len_cat)
print("Total No. Of DOG Samples :- ", len_dog)
plt.pie(arr, labels=labels, explode = [0.2,0.0] , shadow=True)
plt.show()
def get_train_generator(train_dir, batch_size=64, target_size=(224, 224)):
    '''
    Function for preparing training data
    '''
    train_datagen = ImageDataGenerator(rescale = 1./255., # normalizing the image
                                       rotation_range = 40,
                                       width_shift_range = 0.2,
                                       height_shift_range = 0.2,
                                       shear_range = 0.2,
                                       zoom_range = 0.2,
                                       horizontal_flip = True)
    train_generator = train_datagen.flow_from_directory(train_dir,
    batch_size = batch_size, color_mode='rgb',class_mode = 'binary',
          target_size = target_size)
    return train_generator
train_generator = get_train_generator(train_dir)

def get_testgenerator(test_dir,batch_size=64, target_size=(224,224)):
```

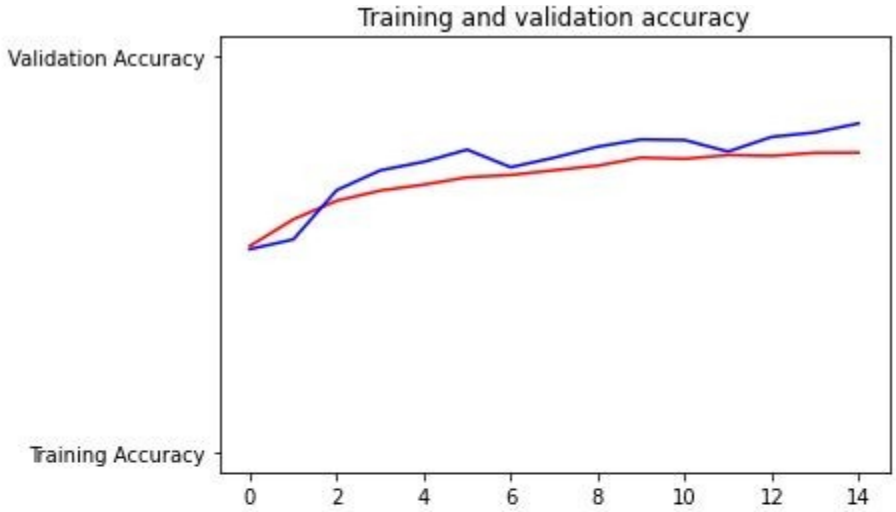
```
    '''
    Function for preparing testing data
    '''
    test_datagen = ImageDataGenerator( rescale = 1.0/255. )
    test_generator  =  test_datagen.flow_from_directory(test_dir,
        batch_size  = batch_size, color_mode='rgb',
      class_mode  = 'binary',   target_size = target_size)
    return test_generator
test_generator = get_testgenerator(test_dir)
model = tf.keras.Sequential([
    layers.Conv2D(64, (3,3), strides=(2,2),padding='same',input_shape= (224,224,3),activation =
'relu'),
    layers.MaxPool2D(2,2),
    layers.Conv2D(128, (3,3), strides=(2,2),padding='same',activation = 'relu'),
    layers.MaxPool2D(2,2),
    layers.Conv2D(256, (3,3), strides=(2,2),padding='same',activation = 'relu'),
    layers.MaxPool2D(2,2),
    layers.Flatten(),
    layers.Dense(158, activation ='relu'),
    layers.Dense(256, activation = 'relu'),
    layers.Dense(128, activation = 'relu'),
    layers.Dense(1, activation = 'sigmoid'),
])
model.summary()
model.compile(optimizer=RMSprop(lr=0.001), loss='binary_crossentropy', metrics=['acc'])
```

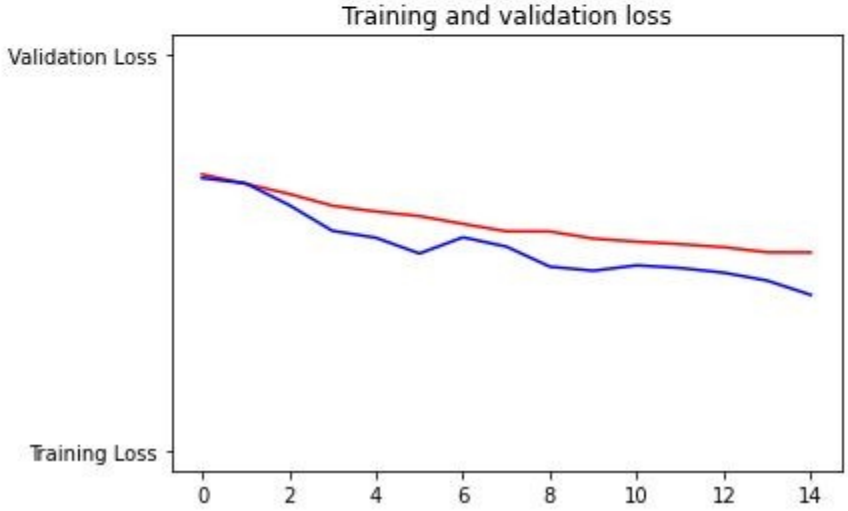
```
history = model.fit_generator(train_generator,
                              epochs=15,
                              verbose=1,
                              validation_data=test_generator)


import matplotlib.image  as mpimg
import matplotlib.pyplot as plt
acc=history.history['acc']
val_acc=history.history['val_acc']
loss=history.history['loss']
val_loss=history.history['val_loss']
epochs=range(len(acc))
plt.plot(epochs, acc, 'r', "Training Accuracy")
plt.plot(epochs, val_acc, 'b', "Validation Accuracy")
plt.title('Training and validation accuracy')
plt.figure()
plt.plot(epochs, loss, 'r', "Training Loss")
plt.plot(epochs, val_loss, 'b', "Validation Loss")
plt.title('Training and validation loss')
model.save('my_model.h5') # saving the trained model
new_model = tf.keras.models.load_model('./my_model.h5') # loading the trained model
```

OUTPUT:

```
Epoch 1/15
126/126 [==============================] - 117s 927ms/step - loss: 0.6851 - acc: 0.5718 - val_loss: 0.6760 - val_acc: 0.5378
Epoch 3/15
126/126 [==============================] - 117s 929ms/step - loss: 0.6533 - acc: 0.6313 - val_loss: 0.6210 - val_acc: 0.6624
Epoch 4/15
126/126 [==============================] - 118s 940ms/step - loss: 0.6212 - acc: 0.6506 - val_loss: 0.5557 - val_acc: 0.7123
Epoch 5/15
126/126 [==============================] - 118s 940ms/step - loss: 0.6128 - acc: 0.6592 - val_loss: 0.5382 - val_acc: 0.7341
Epoch 6/15
126/126 [==============================] - 124s 983ms/step - loss: 0.5853 - acc: 0.6998 - val_loss: 0.4984 - val_acc: 0.7647
Epoch 7/15
126/126 [==============================] - 124s 984ms/step - loss: 0.5730 - acc: 0.7027 - val_loss: 0.5394 - val_acc: 0.7202
Epoch 8/15
126/126 [==============================] - 117s 931ms/step - loss: 0.5627 - acc: 0.7048 - val_loss: 0.5158 - val_acc: 0.7444
Epoch 9/15
126/126 [==============================] - 116s 922ms/step - loss: 0.5628 - acc: 0.7179 - val_loss: 0.4652 - val_acc: 0.7721
Epoch 10/15
126/126 [==============================] - 119s 947ms/step - loss: 0.5381 - acc: 0.7495 - val_loss: 0.4548 - val_acc: 0.7904
Epoch 11/15
126/126 [==============================] - 118s 933ms/step - loss: 0.5238 - acc: 0.7433 - val_loss: 0.4686 - val_acc: 0.7889
Epoch 12/15
126/126 [==============================] - 120s 949ms/step - loss: 0.5149 - acc: 0.7508 - val_loss: 0.4618 - val_acc: 0.7598
Epoch 13/15
126/126 [==============================] - 116s 919ms/step - loss: 0.5215 - acc: 0.7429 - val_loss: 0.4499 - val_acc: 0.7968
Epoch 14/15
126/126 [==============================] - 118s 936ms/step - loss: 0.4905 - acc: 0.7663 - val_loss: 0.4299 - val_acc: 0.8082
Epoch 15/15
126/126 [==============================] - 121s 959ms/step - loss: 0.5031 - acc: 0.7558 - val_loss: 0.3939 - val_acc: 0.8309
```


Training and validation accuracy
Validation Accuracy
Training Accuracy
0
2
4
6
8
10
12
14


Training and validation loss
Validation Loss
Training Loss
0
2
4
6
8
10
12
14

VIVA QUESTIONS

1) What is the role of Conv2D layers in the CNN model?

2) Why is MaxPooling2D used after the convolutional layers?

3) Explain the purpose of the Flatten layer in the CNN model.

4) What is the significance of the ReLU activation function in the convolutional layers?

5) Why is the softmax activation function used in the output layer?

6) What is the purpose of categorical_crossentropy as the loss function?

7) How does the Adam optimizer function, and why is it chosen for this model?

8) What role does the validation_split parameter play during training?

9) How would you modify the CNN to handle grayscale images instead of RGB?

10) What are some strategies to improve the performance of this CNN model?

EXPERIMENT 8

Image classification on MNIST dataset (CNN model with Fully connected layer)

```
import numpy as np

import tensorflow as tf

from sklearn.datasets import fetch_mldata


#Change USERNAME by the username of your machine

## Windows USER

mnist = fetch_mldata('C:\\Users\\USERNAME\\Downloads\\MNIST original')

## Mac User

mnist = fetch_mldata('/Users/USERNAME/Downloads/MNIST original')


print(mnist.data.shape)

print(mnist.target.shape)

from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split(mnist.data, mnist.target, test_size=0.2,
random_state=42)

y_train  = y_train.astype(int)

y_test  = y_test.astype(int)

batch_size =len(X_train)


print(X_train.shape, y_train.shape,y_test.shape )

## resclae

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
```

```
# Train
X_train_scaled = scaler.fit_transform(X_train.astype(np.float64))
# test
X_test_scaled = scaler.fit_transform(X_test.astype(np.float64))
feature_columns = [tf.feature_column.numeric_column('x', shape=X_train_scaled.shape[1:])]
X_train_scaled.shape[1:]


def cnn_model_fn(features, labels, mode):
    input_layer = tf.reshape(tensor = features["x"],shape =[-1, 28, 28, 1])
# first Convolutional Layer
  conv1 = tf.layers.conv2d(
      inputs=input_layer,
      filters=14,
      kernel_size=[5, 5],
      padding="same",
      activation=tf.nn.relu)
# first Pooling Layer
pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2], strides=2)


conv2 = tf.layers.conv2d(
      inputs=pool1,
      filters=36,
      kernel_size=[5, 5],
      padding="same",
      activation=tf.nn.relu)
```

```
pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2], strides=2)

pool2_flat = tf.reshape(pool2, [-1, 7 * 7 * 36])

dense = tf.layers.dense(inputs=pool2_flat, units=7 * 7 * 36, activation=tf.nn.relu)
dropout = tf.layers.dropout(
      inputs=dense, rate=0.3, training=mode == tf.estimator.ModeKeys.TRAIN)

# Logits Layer
logits = tf.layers.dense(inputs=dropout, units=10)

predictions = {
          # Generate predictions
    "classes": tf.argmax(input=logits, axis=1),
    "probabilities": tf.nn.softmax(logits, name="softmax_tensor")  }
if mode == tf.estimator.ModeKeys.PREDICT:
    return tf.estimator.EstimatorSpec(mode=mode, predictions=predictions)

# Calculate Loss (for both TRAIN and EVAL modes)
loss = tf.losses.sparse_softmax_cross_entropy(labels=labels, logits=logits)

optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001)
train_op = optimizer.minimize(
        loss=loss,
        global_step=tf.train.get_global_step())
```

```
# Set up logging for predictions
tensors_to_log = {"probabilities": "softmax_tensor"}
logging_hook = tf.train.LoggingTensorHook(tensors=tensors_to_log, every_n_iter=50)


# Train the model
train_input_fn = tf.estimator.inputs.numpy_input_fn(
    x={"x": X_train_scaled},
    y=y_train,
    batch_size=100,
    num_epochs=None,
    shuffle=True)
mnist_classifier.train(
    input_fn=train_input_fn,
    steps=16000,
    hooks=[logging_hook])
# Evaluate the model and print results
eval_input_fn = tf.estimator.inputs.numpy_input_fn(
    x={"x": X_test_scaled},
    y=y_test,
    num_epochs=1,
    shuffle=False)
eval_results = mnist_classifier.evaluate(input_fn=eval_input_fn)
print(eval_results)
```

OUTPUT:

INFO:tensorflow:Calling model_fn.

INFO:tensorflow:Done calling model_fn.

INFO:tensorflow:Starting evaluation at 2018-08-05-12:52:41

INFO:tensorflow:Graph was finalized.

INFO:tensorflow:Restoring parameters from train/mnist_convnet_model/model.ckpt-15652

INFO:tensorflow:Running local_init_op.

INFO:tensorflow:Done running local_init_op.

INFO:tensorflow:Finished evaluation at 2018-08-05-12:52:56

INFO:tensorflow:Saving dict for global step 15652: accuracy = 0.9589286, global_step = 15652, loss = 0.13894269

{'accuracy': 0.9689286, 'loss': 0.13894269, 'global_step': 15652}

VIVA QUESTIONS

1) What is the role of Conv2D layers in the CNN model?

2) Why is the MaxPooling2D layer used after the convolutional layers?

3) Explain the purpose of the Flatten layer in this CNN model.

4) What is the significance of using the ReLU activation function in the hidden layers?

5) Why is the softmax activation function used in the output layer?

6) What is the role of the categorical_crossentropy loss function in this model?

7) Explain why Adam is used as the optimizer in this CNN model.

8) How does the validation_split parameter benefit the model training process?

9)What are the effects of normalizing image data by scaling pixel values to the range [0, 1]?

10) How would you adjust this model for a more complex dataset with higher resolution images?

EXPERIMENT 9

Applying the Deep Learning Models in the field of Natural Language Processing

```
import numpy as np
import nltk
import re

f = open("chatbot.txt", 'r', errors = 'ignore')
raw_doc = f.read()
raw_doc

raw_doc = raw_doc.lower() #converts text to lowercase

nltk.download('punkt') #using the punkt tokenizer
nltk.download('wordnet') #using the wordNet dictionary
from nltk.tokenize
import sent_tokenize, word_tokenize
sent_tokens = nltk.sent_tokenize(raw_data) #converts doc to list of sentences
word_tokens = nltk.word_tokenize(raw_data) #converts doc to list of words

clean_data = []
for words in word_tokens:
  item = []
result = re.sub(r "[^\w\s]", "", words)
if result != "":
  clean_data.append(result)
nltk.download('stopwords')
from nltk.corpus
```

```
import stopwords

clean_data_1 = []

for words in clean_data:
  if not words in stopwords.words('english'):
  clean_data_1.append(words)

clean_data_1

from nltk.stem
import PorterStemmer
stemmer = PorterStemmer()

stemmed_data = [stemmer.stem(word) for word in clean_data_1]

stemmed_data

from nltk.stem.wordnet
import WordNetLemmatizer
import nltk
lemmer = nltk.stem.WordNetLemmatizer()
nltk.download('wordnet')

lemm_data = []
```

```
for word in clean_data_1:
  lemm_data.append(lemmer.lemmatize(word))


lemm_data


chunkGram = "NP: {<DT>?<JJ>*<NN>}"
chunkParser = nltk.RegexpParser(chunkGram)
chunked = chunkParser.parse(pos_data)


print(chunked)
```

VIVA QUESTIONS

1) What is the purpose of the Embedding layer in the LSTM model?

2) Why use LSTM instead of a simple RNN for this task?

3) Explain the role of dropout in the LSTM layer.

4) What does the recurrent_dropout parameter do in the LSTM layer?

5) Why is binary_crossentropy used as the loss function?

6) How does the Adam optimizer work, and why is it chosen here?

7) What is the significance of padding sequences to a fixed length?

8) How would you adapt this model for a multi-class sentiment analysis problem?

9) What are some common pre-processing steps for text data in NLP?

10) How would you evaluate and improve the performance of this LSTM model?

EXPERIMENT 10

Train a sentiment analysis model on IMDB dataset, use RNN layers with LSTM/GRUnotes

```
import matplotlib. pyplot as plt
import os
import re
import shutil
import string
import tensorflow as tf

from tensorflow.keras import layers
from tensorflow.keras import losses

url = "https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz"
dataset = tf.keras.utils.get_file("aclImdb_v1", url,

                                  untar=True, cache_dir='.',

                                  cache_subdir='')

dataset_dir = os.path.join(os.path.dirname(dataset), 'aclImdb')

Downloading data from https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz

84125825/84125825 [==============================] - 3s 0us/step

os.listdir(dataset_dir)

['train', 'imdb.vocab', 'test', 'README', 'imdbEr.txt']

train_dir = os.path.join(dataset_dir, 'train')

os.listdir(train_dir)

['neg',

 'urls_neg.txt',

 'urls_unsup.txt',

 'unsupBow.feat',

 'unsup',
```

'urls_pos.txt',

'pos',

'labeledBow.feat']

The aclImdb/train/pos and aclImdb/train/neg directories contain many text files, each of which is a single movie review. Let's take a look at one of them.

sample_file = os.path.join(train_dir, 'pos/1181_9.txt')

with open(sample_file) as f:

print(f.read())

main_directory/

...class_a/

......a_text_1.txt

......a_text_2.txt

...class_b/

......b_text_1.txt

......b_text_2.txt

emove_dir = os.path.join(train_dir, 'unsup')

shutil.rmtree(remove_dir)

batch_size = 32

seed = 42

```
raw_train_ds = tf.keras.utils.text_dataset_from_directory(
    'aclImdb/train',
    batch_size=batch_size,
    validation_split=0.2,
    subset='training',
    seed=seed)
for text_batch, label_batch in raw_train_ds.take(1):
  for i in range(3):
    print("Review", text_batch.numpy()[i])
    print("Label", label_batch.numpy()[i])
raw_val_ds = tf.keras.utils.text_dataset_from_directory(
    'aclImdb/train',
    batch_size=batch_size,
    validation_split=0.2,
    subset='validation',
    seed=seed)
```

```
def custom_standardization(input_data):
  lowercase = tf.strings.lower(input_data)
  stripped_html = tf.strings.regex_replace(lowercase, '<br />', ' ')


  return tf.strings.regex_replace(stripped_html,


                                  '[%s]' % re.escape(string.punctuation),


                                  '')
max_features = 10000
sequence_length = 250
vectorize_layer = layers.TextVectorization(
    standardize=custom_standardization,
    max_tokens=max_features,
    output_mode='int',
    output_sequence_length=sequence_length)
```

OUTPUT :

Found 25000 files belonging to 2 classes.

Using 20000 files for training.

Label 0 corresponds to neg

Label 1 corresponds to pos

1287 ---> silent

 313 ---> night

Vocabulary size: 10000

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, None, 16)	160016
dropout (Dropout)	(None, None, 16)	0

global_average_pooling1d (GlobalAveragePooling1D)	(None, 16)	0
dropout_1 (Dropout)	(None, 16)	0
dense (Dense)	(None, 1)	17

Total params: 160,033

Trainable params: 160,033

Non-trainable params: 0

VIVA QUESTIONS

1) What is the purpose of using LSTM/GRU layers in the model?

2) Why do we pad sequences in preprocessing?

3) What is the function of the Embedding layer?

4) Why do we use dropout layers in the model?

5) What does the binary_crossentropy loss function do?

6) How does the model's architecture affect its performance?

7) What is the IMDB dataset?

8) How does the optimizer affect the model training?

9) Why is it important to evaluate the model on test data?

10) Why do we use dropout layers in the model?

EXPERIMENT 11

Applying the Autoencoder algorithms for encoding the real-world data

Step 1: Encoding the input data The Auto-encoder first tries to encode the data using the initialized weights and biases.

Step 2: Decoding the input data The Auto-encoder tries to reconstruct the original input from the encoded data to test the reliability of the encoding.

```
autoencoder.fit(X_train, X_train, epochs=200)

#build the simple encoder-decoder model.

#Notice the number of neurons in each Dense layer.

#The model will contract in the encoder then expand in the decoder.

encoder = keras.models.Sequential([keras.layers.Dense(2, input_shape=[3])])

decoder = keras.models.Sequential([keras.layers.Dense(3, input_shape=[2])])

 autoencoder = keras.models.Sequential([encoder, decoder])

#compile the model

autoencoder.compile(loss="mse", optimizer=keras.optimizers.SGD(lr=0.1))

 #train the model

history = autoencoder.fit(X_train, X_train, epochs=200)

 # encode the data

codings = encoder.predict(X_train)

 # decode the encoder output

decodings = decoder.predict(codings)

faces = fetch_olivetti_faces(shuffle=True, random_state=1000)

X_train = faces['images']

import tensorflow as tf

  nb_epochs = 600

batch_size = 50

code_length = 256

width = 32
```

```
height = 32
graph = tf.Graph()
import numpy as np
  for e in range(nb_epochs):
    np.random.shuffle(X_train)
      total_loss = 0.0
    code_means = []
      for i in range(0, X_train.shape[0] - batch_size, batch_size):
        X = np.expand_dims(X_train[i:i + batch_size, :, :],
                           axis=3).astype(np.float32)

        _, n_loss, c_mean = session.run([training_step, loss, code_mean],
                                 feed_dict={input_images_xl: X})
          total_loss += n_loss
        code_means.append(c_mean)
```

OUTPUT :

Epoch 1) Average loss per sample: 11.933397521972656 (Code mean: 0.5420681238174438)

Epoch 2) Average loss per sample: 10.294102325439454 (Code mean: 0.4132006764411926)

Epoch 3) Average loss per sample: 9.917563934326171 (Code mean: 0.38105469942092896)

...

Epoch 600) Average loss per sample: 0.4635812330245972 (Code mean:0.42368677258491516)

VIVA QUESTIONS

1) What is the IMDB dataset?

2) What is the purpose of using LSTM/GRU layers in the model?

3) Why do we pad sequences in preprocessing?

4) What is the function of the Embedding layer?

5) Why do we use dropout layers in the model?

6) What does the binary_crossentropy loss function do?

7) What is the significance of the validation_data parameter in the fit method?

8) What is the role of the return_sequences parameter in LSTM/GRU layers?

9) How does the optimizer affect the model training?

10) Why is it important to evaluate the model on test data?