

**Vth Semester B. Tech Data Science & Engineering DSE 3141 Deep Learning Lab [0 0 3 1]**

# LABORATORY MANUAL

***Instructors*:**

**Dr. Rohini Rao,**

**Dr. Abhilash K Pai,**

**Dr. Gangothri,**

**Dr. Rajesh Mahadeva,**

**Dr. Namratha Marium Chacko**

# School of Computer Engineering, Manipal Institute of Technology, Manipal, India

**JULY 2025**

**COURSE OUTCOMES (COS)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **At the end of this course, the student should be able to:** | |  | |  | |
|  | **No. of**  **Contact**  **Hours** | **Marks** |  |
|  |  |
| **CO1** | Apply the tools, on different dataset types, do performance evaluation methods, and fine-tuning strategies to build and optimize vanilla deep neural network models for performing classification and regression on structured data. | 7 | | 15 | |
| **CO2** | Design, develop, fine-tune, evaluate simple and advanced CNN models for Image classification. | 9 | | 35 | |
| **CO3** | Design, develop, fine-tune, evaluate simple and advanced RNN models for sequence modelling tasks like Time series prediction and NLP. | 12 | | 35 | |
| **CO4** | Design, develop, fine-tune, and evaluate Autoencoders and Generative models for representational learning. | 8 | | 15 | |
|  | **Total** | **36** | | **100** | |

**ASSESSMENT PLAN**

|  |  |  |  |
| --- | --- | --- | --- |
| **Components** |  | **Continuous Evaluation** | **End semester Examination** |
| **Duration** |  | 2.5 Hours per week | 180 Minutes |
| **Weightage** |  | 60% | 40% |
| **Pattern** | •  •  • | 1 evaluation of 20 marks:   1. Record : 6M, 2. Program execution : 7M, 3. Quiz : 7M   1 Mid-Sem Examination: 20 marks Mini Project : 20 marks  1. Phase1: Problem + Literature: 5M 2. Phase 2: End-to-End solution: 8M  3. Phase 3:Deployment & Demo: 7M | Model Performance Analysis:  15 marks,  Program execution : 25 marks. |

**LESSON PLAN**

|  |  |  |
| --- | --- | --- |
| **Week No** | **TOPICS** | **Course**  **Outcome**  **Addressed** |
| Week 1 | Tensorflow & Keras Tutotial,  Getting Started with Building Fully Connected Neural Networks In  Keras | CO1 |
| Week 2 | Experimenting with Deep Neural Networks | CO1 |
| Week 3 | Convolutional Neural Networks (CNN) Vs Fully Connected Neural  Networks for Image Classification | CO2 |
| Week 4 | Advanced CNN Architectures and Transfer Learning for Image  Classification | CO2 |
| Week 5 | Recurrent Neural Networks for Time Series Forecasting | CO3 |
| Week 6 | Mid-Semester Examination, Mini Project | CO1, CO2, CO3 |
| Week 7 | LSTM and GRU for Sentiment Analysis | CO3 |
| Week 8 | Neural Machine Translation using Encoder-Decoder Architecture,  Mini Project Phase 2 evaluation | CO3 |
| Week 9 | Image Reconstruction and Image Denoising Using Autoencoders,  Mini Project Implementation | CO4 |
| Week 10 | Image Generation Using Generative Adversarial Networks,  Mini Project Implementation | CO4 |
| Week 11 | Mini Project Final Evaluation | CO2, CO3, CO4 |
| Week 12 | End-term lab examination | CO2, CO3, CO4 |

**References:**

|  |  |
| --- | --- |
| **SL.No** | **References** |
| **1** | Aurelien Geron, “Hands-On Machine Learning with Scikit-Learn, Keras & Tensorflow, OReilly Publications |
| **2** | Francois Chollet, “Deep Learning with Python”, Manning Publications Co, 2nd edition |
| **3** | Introduction to Tensorflow, https://www.tensorflow.org/learn |
| **4** | Keras Documentation, https://keras.io/ |
| **5** | Ahmed Menshawy, Md. Rezaul Karim, Giancarlo Zaccone, “ Deep Learning with TensorFlow”, Packt Publishing |

## TENSORFLOW & KERAS TUTORIAL

**1.1 What is TensorFlow?**

TensorFlow is an open-source deep learning framework developed by the Google Brain team. It allows users to create, train, and deploy machine learning models, especially deep neural networks. TensorFlow provides a flexible architecture to work with numerical data using multi-dimensional arrays called **tensors**. It supports both CPU and GPU computations, making it suitable for running on a variety of hardware.

**1.2 What are Tensors?**

In TensorFlow, tensors are the fundamental data structures used for representing data. They are similar to multi-dimensional arrays and can hold data of any number of dimensions. Tensors are the building blocks of neural networks, as they store the input data, weights, biases, and intermediate outputs during the computation.

Examples of Tensors:

1. Scalar (0-D tensor): A single value is a 0-D tensor.

Eg: scalar\_tensor = 5 #rank-0 tensor

1. Vector (1-D tensor): A 1-D tensor contains a sequence of values.

Eg: vector\_tensor = [1, 2, 3, 4, 5] #rank-1 tensor

1. Matrix (2-D tensor): A 2-D tensor is an array of arrays.

Eg: matrix\_tensor = [[1, 2, 3], [4, 5, 6], [7, 8, 9]] #rank-2 tensor

1. Higher-dimensional tensor (e.g., 3-D tensor):

Eg: tensor\_3d = [[[1, 2], [3, 4]], [[5, 6], [7, 8]]] #rank-3 tensor

Note: For a detailed explanation, visit the TensorFlow | Tensor documentation:

<https://www.tensorflow.org/guide/tensor>

**1. 3 Graph Computation:**

TensorFlow follows a symbolic approach for computation using graphs. A graph is a computational graph that represents the flow of data through a series of operations (nodes) to produce output (tensors). The nodes in the graph represent operations, and the edges represent tensors flowing between these operations.

Example of Graph Computation:

import tensorflow as tf

# Define input variables (placeholders) x = tf.placeholder(tf.float32) y = tf.placeholder(tf.float32)

# Define operations

x\_squared = tf.square(x) # Square operation

x\_squared\_times\_y = tf.multiply(x\_squared, y) # Multiply operation result = tf.add(x\_squared\_times\_y, tf.add(y, 2)) # Add operation

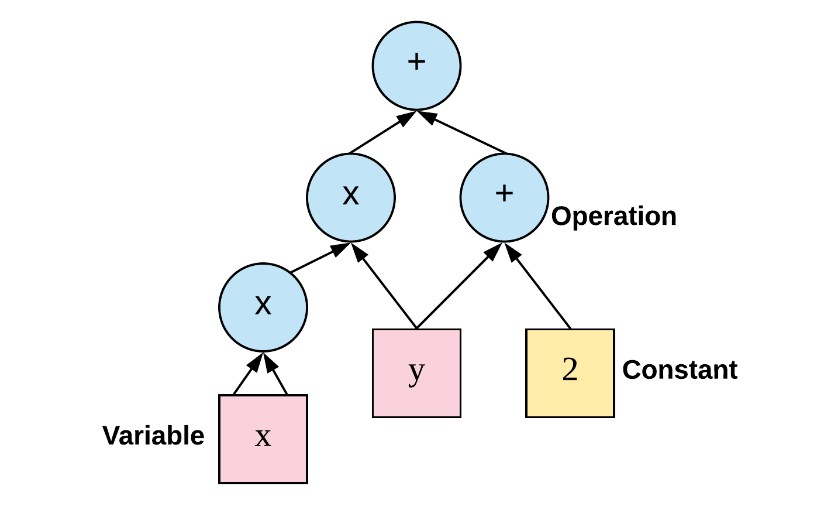
# Create a session to run the computation graph

with tf.Session() as sess:

# Provide input values and run the graph

output = sess.run(result, feed\_dict={x: 3.0, y: 4.0}) print("Output:", output)

Fig1: Computation graph in tensorflow for **f(x, y) = x y + y + 2** [Image Source: https://iq.opengenus.org ]



**2**

**1.4 What is Keras?**

Keras is an open-source high-level neural networks API written in Python and capable of running on top of TensorFlow, among other backends. It was designed with a focus on enabling fast experimentation and easy-to-use syntax for building deep learning models. Keras provides a userfriendly interface for constructing complex neural networks, making it an ideal choice for beginners in deep learning.

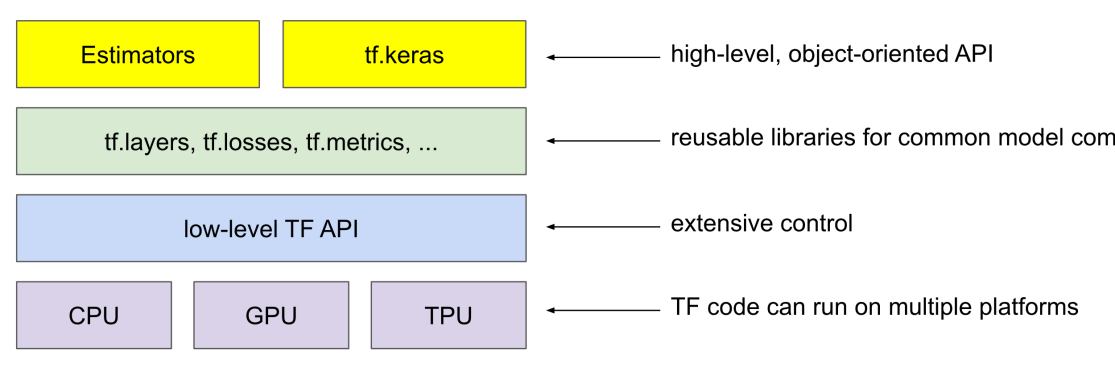


Fig 2. Tensorflow and Keras as API

Image Source: https://developers.google.com/

Note: For a detailed explanation, visit the TensorFlow | Keras documentation: <https://www.tensorflow.org/guide/keras>

In Keras, there are two primary ways to create deep learning models: the **Sequential API** and the **Functional API**. Each approach serves a different purpose and offers distinct advantages.

**1.5 Sequential API:**

The Sequential API is the simplest and most straightforward way to build deep learning models in

Keras. It allows you to create a linear stack of layers, where each layer has exactly one input tensor and one output tensor. This means that the data flows sequentially through each layer in the order they are added to the model. The Sequential API is well-suited for simple feedforward neural networks and other models that have a clear linear flow of data.

**Example of Sequential API:**

from keras.models import **Sequential** from keras.layers import Dense, Input

# Create a sequential model model = **Sequential()**

# Add layers to the model **model.add**(Input(shape=(input\_dim,))) **model.add**(Dense(64, activation='relu')) **model.add**(Dense(32, activation='relu')) **model.add**(Dense(10, activation='softmax'))

# Compile the model

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Print the model summary model.summary()

**1.6 Functional API:**

The Functional API in Keras allows you to create more complex models with multiple input and output tensors, as well as models with shared layers. It provides greater flexibility and is particularly useful when building models with branching or merging architectures.

Example of Functional API:

from keras.models import Model from keras.layers import Input, Dense

# Define input tensor

input\_tensor = Input(shape=(input\_dim,))

# Create layers and connect them

hidden\_layer1 = Dense(64, activation='relu')**(input\_tensor)** hidden\_layer2 = Dense(32, activation='relu')**(hidden\_layer1)** output\_tensor = Dense(10, activation='softmax')**(hidden\_layer2)**

# Create the model

model = Model(inputs=input\_tensor, outputs=output\_tensor)

# Compile the model

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Print the model summary model.summary()

**1.7 Deep Learning Model Life-Cycle**

The deep learning model life cycle typically involves the following steps: Define the model, Compile the model, Fit the model, Evaluate the model, and Make predictions. **I. Define the Model:**

In this step, you specify the architecture of your deep learning model. You define the layers, their configurations, activation functions, and any other required settings. The architecture depends on the problem you are trying to solve, and it may include fully connected layers, convolutional layers, recurrent layers, etc.

from keras.models import Sequential from keras.layers import Dense

# Define the model model = Sequential()

model.add(Dense(64, activation='relu', input\_shape=(input\_dim,))) model.add(Dense(32, activation='relu')) model.add(Dense(10, activation='softmax'))

1. **Compile the Model:**

After defining the model, you need to compile it. During this step, you specify the loss function, optimizer, and evaluation metrics. The loss function is used to measure how well the model is performing on the training data. The optimizer determines how the model's weights are updated during training, and the evaluation metrics provide additional performance metrics during training.

# Compile the model

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

1. **Fit the Model:**

In this step, you train the model on your training data. You provide the input features (X) and their corresponding target labels (y) to the model. The model then adjusts its internal parameters (weights) through an optimization process (usually gradient descent) to minimize the defined loss function.

# Fit the model

model.fit(X\_train, y\_train, epochs=10, batch\_size=32, validation\_data=(X\_val, y\_val))

1. **Evaluate the Model:**

After the model is trained, you need to evaluate its performance on a separate set of data that it has never seen before (e.g., a validation set or a test set). This step gives you an indication of how well the model generalizes to unseen data.

# Evaluate the model

loss, accuracy = model.evaluate(X\_test, y\_test)

print(f"Test loss: {loss}, Test accuracy: {accuracy}")

1. **Make Predictions:**

Once the model is trained and evaluated, you can use it to make predictions on new, unseen data. You pass the new data to the model, and it will provide predictions based on what it has learned during training.

# Make predictions

predictions = model.predict(X\_new\_data)

Example: Building a Simple Neural Network with Keras

#1) Import the necessray libraries

import numpy as np import tensorflow as tf

from tensorflow.keras.models import Sequential from tensorflow.keras.layers import Dense, Input

#2) For the tutorial, lets experiment with random data

# Generate random input data (features)

X = np.random.rand(num\_samples, num\_features)

# Generate random output labels (classes)

y = np.random.randint(0, num\_classes, size=num\_samples)

# Split the data into training and testing sets split\_ratio = 0.8

split\_index = int(num\_samples \* split\_ratio)

X\_train, X\_test = X[:split\_index], X[split\_index:] y\_train, y\_test = y[:split\_index], y[split\_index:]

#3) Define the model

# Build the neural network model using Sequential API model = Sequential([

Input(shape=(num\_features,)),

Dense(6, activation='relu'), # Hidden layer with 6 neurons Dense(num\_classes, activation='softmax') # Output layer with num\_classes neurons and softmax activation for classification

])

# Display a summary of the model architecture model.summary()

#4) Compile the model

# Compile the model

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

#5) Fit/train the model

# Train the model using the training data epochs = 50 batch\_size = 32

model.fit(X\_train, y\_train, epochs=epochs, batch\_size=batch\_size, validation\_split=0.1)

#6) Evaluate/test the model

# Evaluate the model on the testing data

loss, accuracy = model.evaluate(X\_test, y\_test, batch\_size=batch\_size) print("Test Loss:", loss) print("Test Accuracy:", accuracy)

## WEEK-1: GETTING STARTED WITH BUILDING FULLY CONNECTED NEURAL NETWORKS IN KERAS

Q1. Implement the example code provided in the tutorial by considering num\_samples=3, num\_features=10. Plot the training and validation performance curves and analyze the performance of the proposed neural network.

Q2. Using the **Iris Flowers Dataset**, build and Neural Network with the following specifications to perform multi-class classification.

* Split the Data into Training: Validation: Testing = 80:10:10
* Number of Hidden Layers = 2, containing 8 Neurons and 4 Neurons
* Use RELU activation function in the hidden layers, choose the optimizer as ADAM and set learning rate to be equal to 0.1.

Q3. Accurate measurement of body fat is inconvenient/costly, and it is desirable to have easy methods of predicting Body Fat. Using the given Body Fat dataset, build a Neural Network to predict body fat. Plot the training and validation performance curves and analyze the performance of the proposed neural network.

The attributes of the dataset are as follows:

* 1. Density determined from underwater weighing
  2. Percent body fat from Siri's (1956) equation
  3. Age (years)
  4. Weight (lbs)
  5. Height (inches)
  6. Neck circumference (cm)
  7. Chest circumference (cm)
  8. Abdomen 2 circumference (cm)
  9. Hip circumference (cm)
  10. Thigh circumference (cm)
  11. Knee circumference (cm)
  12. Ankle circumference (cm)
  13. Biceps (extended) circumference (cm)
  14. Forearm circumference (cm)
  15. Wrist circumference (cm)

Use the following hyperparameters/design choices for your neural network:

* Split the data in the ratio Training: Validation: Testing = 80:10:10.
* Perform Normalization using Standard Scalar.
* Number of Hidden layers = 3 and number of units for each hidden layers are 128,64,32, respectively.
* Use RELU activation function in the hidden layers, choose the optimizer as ADAM and set learning rate to be equal to 0.1.

Q4. For Q2 and Q3, Interpret the results of “model.summary()”.

## WEEK-2: EXPERIMENTING WITH DEEP NEURAL NETWORKS

Q1. Consider the following dataset ‘Churn\_Modelling.csv’:

<https://www.kaggle.com/datasets/aakash50897/churn-modellingcsv>

The dataset has 14 features which are as follows :

* RowNumber:- Represents the number of rows
* CustomerId:- Represents customerId
* Surname:- Represents surname of the customer
* CreditScore:- Represents credit score of the customer
* Geography:- Represents the city to which customers belongs to
* Gender:- Represents Gender of the customer
* Age:- Represents age of the customer
* Tenure:- Represents tenure of the customer with a bank
* Balance:- Represents balance hold by the customer
* NumOfProducts:- Represents the number of bank services used by the customer
* HasCrCard:- Represents if a customer has a credit card or not
* IsActiveMember:- Represents if a customer is an active member or not
* EstimatedSalary:- Represents estimated salary of the customer
* Exited:- Represents if a customer is going to exit the bank or not.

1. Perform the required pre-processing and write comment lines to explain the pre-processing steps.

1. Perform experiments using (80,10,10) split and tabulate performance in terms of Accuracy, Precision & Recall for the following experimental setup:
   1. Number of Hidden Layers and Number of Units per Layer

|  |  |
| --- | --- |
| **Number of Hidden Layers** | **Number of Units** |
| **1** | 128, 0 ,0 |
| **2** | 128, 64, 0 |
| **3** | 128, 64, 32 |

* 1. Epochs (10,20,30)
  2. Activation function (Sigmoid, ReLU )
  3. Without Regularization, with Regularization (L1/L2)
  4. Learning rate (0.1, 0.01,0.001)

Visualize the training and validation loss against the epochs and comment on optimal hyperparameters.

Q2. Consider the following Boston Housing dataset — **a well‑known dataset used in housing value-t**o understand the effect of various hyperparameters and architectural changes on the performance of a neural network using the House Boat dataset.

**Features in the dataset:**

* **CRIM**: per capita crime rate by town
* **ZN**: proportion of residential land zoned for lots over 25,000 sq.ft.
* **INDUS**: proportion of non-retail business acres per town
* **CHAS**: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
* **NOX**: nitric oxides concentration (air pollution)
* **RM**: average number of rooms per dwelling
* **AGE**: proportion of owner-occupied units built before 1940
* **DIS**: distances to five Boston employment centers
* **RAD**: accessibility to radial highways
* **TAX**: property tax rate per $10,000
* **PTRATIO**: pupil–teacher ratio by town
* **B**: 1000 × (Bk – 0.63)² where Bk is the proportion of Black residents
* **LSTAT**: % of lower status population
* **MEDV**: **Target** – Median value of owner-occupied homes (in $1000s)

Build a neural network **to predict the median house value (MEDV)** using the above features.

Use the following hyperparameters/design choices for your neural network:

* Split the data in the ratio Training: Validation: Testing = 80:10:10.
* Perform Normalization using Standard Scalar.
* Number of Hidden layers = 3 and number of units for each hidden layers are 128,64,32, respectively.
* Epochs (10,20,30)
* Activation function (Sigmoid, ReLU )
* Without Regularization, with Regularization (L1/L2)
* Learning rate (0.1, 0.01,0.001)
* Visualize the training and validation loss against the epochs and comment on optimal hyperparameters.

Q3. For Q1 and Q2, Interpret the results of “model.summary()”.

**WEEK-3: CONVOLUTIONAL NEURAL NETWORKS (CNN) VS FULLY CONNECTED NEURAL NETWORKS FOR IMAGE CLASSIFICATION**

Consider the following datasets:

1. **Fashion MNIST dataset** [[Fashion MNIST dataset, an alternative to MNIST (keras.io)](https://keras.io/api/datasets/fashion_mnist/)],
2. **CIFAR-10 dataset** [[CIFAR10 small images classification dataset (keras.io)]](https://keras.io/api/datasets/cifar10/)

**For each of the Datasets A and B, do the following:**

Q1. Understanding the Dataset and Pre-processing: Implement the following:

1. Compute and display the number of classes.
2. Compute and display the dimensions of each image.
3. Display one image from each class.
4. Perform normalization.

Q2. Performing experiments on Fully Connected Neural Networks (FCNN):

1. Design a FCNN which is most suitable for the given dataset:   
   Experimentally choose the best network (the intuitions and learnings from the experiments you have performed in Week-1 and Week-2 will help you choose the hyperparameters for the network).
2. Train and test the network (choose the best epoch size so that there is no overfitting).
3. Plot the performance curves.

Q3. Performing experiments on a Convolutional Neural Networks (CNNs):

1. Design **CNN-1** which contains:

* One Convolution layer which uses 32 kernels each of size 5x5, stride = 1 and, padding =0.
* One Pooling layer which uses MAXPOOLING with stride =2.
* One hidden layer having number of neurons = 100

1. Design **CNN-2** which contains:

* Two back-to-back Convolution layers which uses 32 kernels each of size 3x3, stride = 1, and padding =0.
* One Pooling layer which uses MAXPOOLING with stride =2.
* One hidden layer having number of neurons = 100

Note: use ReLU activation function after each convolution layer.

1. Train and test the networks (choose the best epoch size so that there is no overfitting).
2. Plot the performance curves for CNN-1 and CNN-2.
3. Compare the performances of CNN-1 and CNN-2.

Q4. Compare the performances of FCNN and CNN.

Q5. Compare the number of parameters in the FCNN and the CNN. Which layer/s in CNN contribute most to the total number of parameters.