LIVE TO LEARN & LEARN TO SHARE

MRCET CAMPUS

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(AUTONOMOUS INSTITUTION - UGC, GOVT. OF INDIA)

Affiliated to JNTUH; Approved by AICTE, NBA-Tier 1 & NAAC with A-GRADE | ISO 9001:2015

Maisammaguda, Dhulapally, Komaplly, Secunderabad - 500100, Telangana State, India

LABORATORY MANUAL & RECORD

Name:..

Roll No:................Branch:...

Year:..................Sem:...

MRCET CAMPUS

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(AUTONOMOUS INSTITUTION - UGC, GOVT. OF INDIA)

Affiliated to JNTUH; Approved by AICTE, NBA-Tier 1 & NAAC with A-GRADE | ISO 9001:2015

Maisammaguda, Dhulapally, Komaplly, Secunderabad - 500100, Telangana State, India

Certificate

Certified that this is the Bonafide Record of the Work Done by

Mr./Ms...Roll.No.................of

B.Tech............year.......................... Semester for Academic year........................

in..Laboratory.

Date: Faculty Incharge HOD

Internal Examiner External Examiner

INDEX

S.No	Date	Name of the Activity/Experiment	Grade/ Marks	Faculty Signature

DEEP LEARNING
LAB MANUAL

B. TECH

(IV YEAR – I SEM)
(2024-25)

DEPARTMENT OF COMPUTATIONAL INTELLIGENCE

(CSE – AIML, AIML, AIDS)

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY
(Autonomous Institution – UGC, Govt. of India)

Recognized under 2(f) and 12 (B) of UGC ACT 1956
(Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015 Certified)
Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad – 500100, Telangana State, India

Department of Computer Science & Engineering

(Artificial Intelligence & Machine Learning)

Vision

To be a premier center for academic excellence and research through innovative interdisciplinary collaborations and making significant contributions to the community, organizations, and societyas a whole.

Mission

- To impart cutting-edge Artificial Intelligence technology in accordance with industry norms.
- To instill in students a desire to conduct research in order to tackle challenging technical problems for industry.
- To develop effective graduates who are responsible for their professional growth, leadership qualities and are committed to lifelong learning.

Quality Policy

- To provide sophisticated technical infrastructure and to inspire students to reach their full potential.
- To provide students with a solid academic and research environment for a comprehensive learning experience.
- To provide research development, consulting, testing, and customized training to satisfy specific industrial demands, thereby encouraging self-employment and entrepreneurship among students.

Department of Computer Science & Engineering

(Artificial Intelligence & Machine Learning)

Programme Educational Objectives (PEO):

PEO1: To possess knowledge and analytical abilities in areas such as Maths, Science, and fundamentalengineering.
PEO2: To analyse, design, create products, and provide solutions to problems in Computer Scienceand Engineering.
PEO3: To leverage the professional expertise to enter the workforce, seek higher education, andconduct research on AI-based problem resolution.
PEO4: To be solution providers and business owners in the field of computer science and engineering with an emphasis on artificial intelligence and machine learning.

Programme Specific Outcomes (PSO):

After successful completion of the program a student is expected to have specificabilities to:

PSO1: To understand and examine the fundamental issues with AI and ML applications.

PSO2: To apply machine learning, deep learning, and artificial intelligence approaches to address issues in social computing, healthcare, vision, language processing, speech recognition, and other domains.
PSO3: Use cutting-edge AI and ML tools and technology to further your study and research.

Department of Computer Science & Engineering

(Artificial Intelligence & Machine Learning)

PROGRAM OUTCOMES (POs)

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **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.
3. **Design / development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations 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.
5. **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.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **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.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multi-disciplinary environments.
12. **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.

Department of Computer Science & Engineering

(Artificial Intelligence & Machine Learning)

Lab Objectives:

- To introduce the basic concepts and techniques of Deep Learning and the need of Deep Learning techniques in real-world problems.
- To provide understanding of various Deep Learning algorithms and the way to evaluate performance of the Deep Learning algorithms.
- To apply Deep Learning to learn, predict and classify the real-world problems.
- To understand, learn and design Artificial Neural Networks of Supervised Learning for the selected problems and very the different parameters.
- To understand the concept of CNN, RNN, GANs, Auto-encoders.
- To inculcate in students professional and ethical attitude, multidisciplinary approach and an ability to relate real-world issues and provide a cost-effective solution to it by developing DL applications.
- To provide student with an academic environment aware of excellence, written ethical codes and guidelines and lifelong learning needed for a successful professional career.

Lab Outcomes:

Upon successful completion of this course, the students will be able to:

- Understand the basic concepts and techniques of Deep Learning and the need of Deep Learning techniques in real-world problems.
- Understand CNN algorithms and the way to evaluate performance of the CNN architectures.
- Apply RNN and LSTM to learn, predict and classify the real-world problems in the paradigms of Deep Learning.
- Understand, learn and design GANs for the selected problems.
- Understand the concept of Auto-encoders and enhancing GANs using auto-encoders.

Introduction about lab

System configurations are as follows:

- **Hardware / Software's installed: I**ntel® CORE™ i3-3240 CPU@3.40GHZRAM: 4GB / Anaconda Navigator or Python and Jupyter Notebook or Google Colab.
- **Packages required to run the programs:** Math, Scipy, Numpy, Matplotlib, Pandas, Sklearn, Tensorflow, Keras etc.
- Systems are provided for students in the **1:1 ratio.**
- Systems are assigned numbers and same system is allotted for students when they do the lab.
- All Systems are configured in LINUX, it is open source and students can use any different programming environments through package installation.

Guidelines to students

A. Standard operating procedure

a) Explanation on today's experiment by the concerned faculty using PPT covering the following aspects:

1) Name of the experiment
2) Aim
3) Software/Hardware requirements
4) Writing the python programs by the students
5) Commands for executing programs

Writing of the experiment in the Observation Book

The students will write the today's experiment in the Observation book as per the following format:

a) Name of the experiment
b) Aim
c) Writing the program
d) Viva-Voce Questions and Answers
e) Errors observed (if any) during compilation/execution

Signature of the Faculty

B. Guide Lines to Students in Lab

Disciplinary to be maintained by the students in the Lab

- Students are required to carry their lab observation book and record book with completed experiments while entering the lab.
- Students must use the equipment with care. Any damage is caused student is punishable.
- Students are not allowed to use their cell phones/pen drives/ CDs in labs.
- Students need to maintain proper dress code along with ID Card.
- Students are supposed to occupy the computers allotted to them and are not supposed totalk or make noise in the lab.
- Students, after completion of each experiment they need to be updated in observation notes and same to be updated in the record.
- Lab records need to be submitted after completion of experiment and get it corrected withthe concerned lab faculty.
- If a student is absent for any lab, they need to be completed the same experiment in the free time before attending next lab.

Steps to perform experiments in the lab by the student

Step 1: Students have to write the date, aim and for that experiment in the observation book.

Step 2: Students have to listen and understand the experiment explained by the faculty and note down the important points in the observation book.

Step 3: Students need to write procedure/algorithm in the observation book.

Step 4: Analyze and Develop/Implement the logic of the program by the student inrespective platform

Step 5: After approval of logic of the experiment by the faculty then the experiment has to beexecuted on the system.

Step 6: After successful execution the results are to be shown to the faculty and noted thesame in the observation book.

Step 7: Students need to attend the Viva-Voce on that experiment and write the same in theobservation book.

Step 8: Update the completed experiment in the record and submit to the concernedfaculty in-charge.

Instructions to maintain the record

- Before start of the first lab, they have to buy the record and bring the record to the lab.
- Regularly (Weekly) update the record after completion of the experiment and get it corrected with concerned lab in-charge for continuous evaluation. In case the record is lost inform the same day to the faculty in charge and get the new record within 2 days the record has to be submitted and get it corrected by the faculty.
- If record is not submitted in time or record is not written properly, the evaluation marks (5M) will be deducted.

Awarding the marks for day-to-day evaluation

Total marks for day-to-day evaluation is 15 Marks as per Autonomous (JNTUH). These 15 Marks are distributed as:

Regularity	3 Marks
Program written	3 Marks
Execution & Result	3 Marks
Viva-Voce	3 Marks
Dress Code	3 Marks

Allocation of Marks for Lab Internal

Total marks for lab internal are 30 Marks as per Autonomous (JNTUH)

These 30 Marks are distributed as:

Average of day-to-day evaluation marks: 15

Marks Lab Mid exam: 10 Marks

VIVA & Observation: 5 Marks

Allocation of Marks for Lab External

Total marks for lab Internal and External are 70 Marks as per Autonomous / (JNTUH).

These 70 External Lab Marks are distributed as:

Program Written	30 Marks
Program Execution and Result	20 Marks
Viva-Voce	10 Marks
Record	10 Marks

C. General laboratory instructions

1. Students are advised to come to the laboratory at least 5 minutes before (to the starting time), those who come after 5 minutes will not be allowed into the lab.
2. Plan your task properly much before to the commencement, come prepared to the lab withthe synopsis / program / experiment details.
3. Student should enter into the laboratory with:

a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session.
b. Laboratory Record updated up to the last session experiments and other utensils (if any) needed in the lab.
c. Proper Dress code and Identity card.

4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer system allotted to you by the faculty.
5. Execute your task in the laboratory, and record the results / output in the lab observation note book, and get certified by the concerned faculty.
6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.
7. Computer labs are established with sophisticated and high-end branded systems, which should be utilized properly.
8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviors with the staff and systems etc., will attract severe punishment.
9. Students must take the permission of the faculty in case of any urgency to go out; if anybody found loitering outside the lab / class without permission during working hours will be treated seriously and punished appropriately.
10. Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

Head of the Department **Principal**

INDEX

Deep Learning Lab Manual (R20A6683)		
S. No.	**Name of the Program**	**Page No.**
1	a. Design a single unit perceptron for classification of a linearly separable binary dataset without using pre-defined models. Use the Perceptron() from sklearn. b. Identify the problem with single unit Perceptron. Classify using Or-, And- and Xor-ed data and analyze the result.	
2	Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets. Vary the activation functions used and compare the results.	
3	Build a Deep Feed Forward ANN by implementing the Backpropagation algorithm and test the same using appropriate data sets. Use the number of hidden layers >=4.	
4	Design and implement an Image classification model to classify a dataset of images using Deep Feed Forward NN. Record the accuracy corresponding to the number of epochs. Use the MNIST, CIFAR-10 datasets.	
5	Design and implement a CNN model (with 2 layers of convolutions) to classify multi category image datasets. Record the accuracy corresponding to the number of epochs. Use the MNIST, CIFAR-10 datasets.	
6	Design and implement a CNN model (with 4+ layers of convolutions) to classify multi category image datasets. Use the MNIST, Fashion MNIST, CIFAR-10 datasets. Set the No. of Epoch as 5, 10 and 20. Make the necessary changes whenever required. Record the accuracy corresponding to the number of epochs. Record the time required to run the program, using CPU as well as using GPU in Colab.	
7	Design and implement a CNN model (with 2+ layers of convolutions) to classify multi category image datasets. Use the concept of padding and Batch Normalization while designing the CNN model. Record the accuracy corresponding to the number of epochs. Use the Fashion MNIST/MNIST/CIFAR10 datasets.	
8	Design and implement a CNN model (with 4+ layers of convolutions) to classify multi category image datasets. Use the concept of regularization and dropout while designing the CNN model. Use the Fashion MNIST datasets. Record the Training accuracy and Test accuracy corresponding to the following architectures: a. Base Model b. Model with L1 Regularization c. Model with L2 Regularization d. Model with Dropout e. Model with both L2 (or L1) and Dropout	
9	Use the concept of Data Augmentation to increase the data size from a single image.	
10	Design and implement a CNN model to classify CIFAR10 image dataset. Use the concept of Data Augmentation while designing the CNN model. Record the accuracy corresponding to the number of epochs.	
11	Implement the standard LeNet-5 CNN architecture model to classify multi-category image dataset (MNIST, Fashion MNIST) and check the accuracy.	

12	Implement the standard VGG-16 & 19 CNN architecture model to classify multi category image dataset and check the accuracy.	
13	Implement RNN for sentiment analysis on movie reviews.	
14	Implement Bidirectional LSTM for sentiment analysis on movie reviews.	
15	Implement Generative Adversarial Networks to generate realistic Images. Use MNIST, Fashion MNIST or any human face datasets.	
16	Implement Auto encoders for image denoising on MNIST, Fashion MNIST or any suitable dataset.	

Week-1

a. Design a single unit perceptron for classification of a linearly separable binary dataset (placement.csv) without using pre-defined models. Use the Perceptron() from sklearn.

Program

```
# Single unit perceptron
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import Perceptron

df=pd.read_csv('/content/gdrive/My Drive/ML_lab/placement.csv')
X = df.iloc[:,0:2]
y  =  df.iloc[:,-1]
p = Perceptron()
p.fit(X,y)
print(p.coef_)
print(p.intercept_)
z=p.score(X,y)
print("accuracy score is",z)
from mlxtend.plotting import plot_decision_regions
plot_decision_regions(X.values, y.values, clf=p, legend=2)
```

OUTPUT:

Exercise:

Design a single unit perceptron for classification of a linearly separable binary dataset without using pre-defined models. Use the Perceptron() from sklearn.

[hint-use make_classification() to generate binary dataset from sklearn
Eg:
```
from sklearn.datasets import make_classification
X, y = make_classification(n_samples=100, n_features=2,
n_informative=1,n_redundant=0,n_classes=2, n_clusters_per_class=1,
random_state=41,hypercube=False,class_sep=10)
```
]

b. Identify the problem with single unit Perceptron. Classify using Or-, And- and Xor-ed data and analysis the result.

Program

```
# Perceptron on Or-, And- and Xor-ed data
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
or_data = pd.DataFrame()
and_data = pd.DataFrame()
xor_data = pd.DataFrame()

or_data['input1']=[1,1,0,0]
or_data['input2']=[1,0,1,0]
or_data['ouput']=[1,1,1,0]

and_data['input1']=[1,1,0,0]
and_data['input2']=[1,0,1,0]
and_data['ouput']=[1,0,0,0]

xor_data['input1']=[1,1,0,0]
xor_data['input2']=[1,0,1,0]
xor_data['ouput']=[0,1,1,0]

from sklearn.linear_model import Perceptron
clf1=Perceptron()
clf2=Perceptron()
clf3=Perceptron()
clf1.fit(and_data.iloc[:,0:2].values,and_data.iloc[:,-1].values)
print(clf1.coef_)
print(clf1.intercept_)
x=np.linspace(-1,1,5)
y=-x+1
plt.plot(x,y)
#sns.scatterplot(and_data['input1'],and_data['input2'],hue=and_data['ouput'],s=200)
clf2.fit(or_data.iloc[:,0:2].values,or_data.iloc[:,-1].values)
print(clf2.coef_)
print(clf2.intercept_)
x1=np.linspace(-1,1,5)
y1=-x+0.5
plt.plot(x1,y1)
#sns.scatterplot(or_data['input1'],or_data['input2'],hue=or_data['ouput'],s=200)
clf3.fit(xor_data.iloc[:,0:2].values,xor_data.iloc[:,-1].values)
print(clf3.coef_)
print(clf3.intercept_)
plot_decision_regions(xor_data.iloc[:,0:2].values,xor_data.iloc[:,-1].values,
clf=clf3, legend=2)
```

OUTPUT:

Exercise

Identify the problem with single unit Perceptron. Classify using Not- and XNOR-ed data and analyze the result.

Week-2

Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets. Vary the activation functions used and compare the results.

Program:

```
from keras.models import Sequential
from keras.layers import Dense, Activation
import numpy as np
import pandas as pd
from sklearn import datasets
iris = datasets.load_iris()
X, y = datasets.load_iris( return_X_y = True)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.40)
# Define the network model and its arguments.
# Set the number of neurons/nodes for each layer:
model = Sequential()
model.add(Dense(2, input_shape=(4,)))
model.add(Activation('sigmoid'))
model.add(Dense(1))
model.add(Activation('sigmoid'))
#sgd = SGD(lr=0.0001, decay=1e-6, momentum=0.9, nesterov=True)
#model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
# Compile the model and calculate its accuracy:
model.compile(loss='mean_squared_error', optimizer='sgd', metrics=['accuracy'])
#model.fit(X_train, y_train, batch_size=32, epochs=3)
# Print a summary of the Keras model:
model.summary()
#model.fit(X_train, y_train)
#model.fit(X_train, y_train, batch_size=32, epochs=300)
model.fit(X_train, y_train, epochs=5)
score = model.evaluate(X_test, y_test)
print(score)
```

OUTPUT:

Exercise:

Note down the accuracies for the following set of experiments on the given NN and compare the results Do the required modifications needed. Take training data percentage 30%, test data percentage 70%.

b) NN model with 2 hidden layers

- **(1) Iris dataset**
  - **(a) No. of epochs=100,**
    - **(i) check accuracy using activation functions Sigmoid, ReLu, Tanh**
    - **(ii) check accuracy using optimizer sgd, ADAM**
    - **(iii) check accuracy by varying learning rate in sgd as 0.0001, 0.0005, 5.**
    - **(iv) check accuracy using loss mean squared error, categorical cross entropy.**
  - **(b) No. of epochs =300**
    - **(i) Repeat the same above variations**
- **(2) Ionosphere data**
  - **(a) Repeat the same settings as Iris**

Week-3

Build a Deep Feed Forward ANN by implementing the Backpropagation algorithm and test the same using appropriate data sets. Use the number of hidden layers >=4.

Program:

```python
from keras.models import Sequential
from keras.layers import Dense, Activation
import numpy as np
import pandas as pd
from sklearn import datasets
iris = datasets.load_iris()
X, y = datasets.load_iris(return_X_y = True)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.40)
# Define the network model and its arguments.
# Set the number of neurons/nodes for each layer:
model = Sequential()
model.add(Dense(2, input_shape=(4,)))
model.add(Activation('sigmoid'))
model.add(Dense(1))
model.add(Activation('sigmoid'))
model.add(Dense(2, input_shape=(4,)))
model.add(Activation('sigmoid'))
model.add(Dense(1))
model.add(Activation('sigmoid'))
model.add(Dense(2, input_shape=(4,)))
model.add(Activation('sigmoid'))
model.add(Dense(1))
model.add(Activation('sigmoid'))

#sgd = SGD(lr=0.0001, decay=1e-6, momentum=0.9, nesterov=True)
#model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
# Compile the model and calculate its accuracy:
model.compile(loss='mean_squared_error', optimizer='sgd', metrics=['accuracy'])
#model.fit(X_train, y_train, batch_size=32, epochs=3)
# Print a summary of the Keras model:
model.summary()
#model.fit(X_train, y_train)
#model.fit(X_train, y_train, batch_size=32, epochs=300)
model.fit(X_train, y_train, epochs=5)
score = model.evaluate(X_test, y_test)
print(score)
```

OUTPUT:

Exercise:

Modify the above NN model to run on Ionosphere dataset with number of hidden layers >=4. Take training data percentage 30%, test data percentage 70%. No. of epochs=100, activation function ReLu, optimizer ADAM.

Week-4

Design and implement an Image classification model to classify a dataset of images using Deep Feed Forward NN. Record the accuracy corresponding to the number of epochs. Use the MNIST datasets.

Program

```
#load required packages
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras import Input
from keras.layers import Dense
import pandas as pd
import numpy as np
import sklearn
from sklearn.metrics import classification_report
import matplotlib
import matplotlib.pyplot as plt

# Load digits data
(X_train, y_train), (X_test, y_test) = keras.datasets.mnist.load_data()

# Print shapes
print("Shape of X_train: ", X_train.shape)
print("Shape of y_train: ", y_train.shape)
print("Shape of X_test: ", X_test.shape)
print("Shape of y_test: ", y_test.shape)

# Display images of the first 10 digits in the training set and their true lables
fig, axs = plt.subplots(2, 5, sharey=False, tight_layout=True, figsize=(12,6),
facecolor='white')
n=0
for i in range(0,2):
    for j in range(0,5):
        axs[i,j].matshow(X_train[n])
        axs[i,j].set(title=y_train[n])
        n=n+1
plt.show()

# Reshape and normalize (divide by 255) input data
X_train = X_train.reshape(60000, 784).astype("float32") / 255
X_test = X_test.reshape(10000, 784).astype("float32") / 255

# Print shapes
print("New shape of X_train: ", X_train.shape)
print("New shape of X_test: ", X_test.shape)

#Design the Deep FF Neural Network architecture
model = Sequential(name="DFF-Model") # Model
model.add(Input(shape=(784,), name='Input-Layer')) # Input Layer - need to specify
the shape of inputs
```

```
model.add(Dense(128, activation='relu', name='Hidden-Layer-1',
kernel_initializer='HeNormal'))
model.add(Dense(64, activation='relu', name='Hidden-Layer-2',
kernel_initializer='HeNormal'))
model.add(Dense(32, activation='relu', name='Hidden-Layer-3',
kernel_initializer='HeNormal'))
model.add(Dense(10, activation='softmax', name='Output-Layer'))

#Compile keras model
model.compile(optimizer='adam', loss='SparseCategoricalCrossentropy',
metrics=['Accuracy'], loss_weights=None, weighted_metrics=None, run_eagerly=None,
steps_per_execution=None)

#Fit keras model on the dataset
model.fit(X_train, y_train, batch_size=10, epochs=5, verbose='auto', callbacks=None,
validation_split=0.2, shuffle=True, class_weight=None, sample_weight=None,
initial_epoch=0, # Integer, default=0, Epoch at which to start training (useful for
resuming a previous training run).
          steps_per_epoch=None, validation_steps=None, validation_batch_size=None,
validation_freq=5, max_queue_size=10, workers=1, use_multiprocessing=False,)

# apply the trained model to make predictions
# Predict class labels on training data
pred_labels_tr = np.array(tf.math.argmax(model.predict(X_train),axis=1))
# Predict class labels on a test data
pred_labels_te = np.array(tf.math.argmax(model.predict(X_test),axis=1))


#Model Performance Summary
print("")
print('--------------------------- Model Summary ---------------------------')
model.summary()
print("")

# Printing the parameters:Deep Feed Forward Neural Network contains more than 100K
#print('--------------------------- Weights and Biases ---------------------------')
#for layer in model_d1.layers:
    #print("Layer: ", layer.name) # print layer name
    #print("  --Kernels (Weights): ", layer.get_weights()[0]) # kernels (weights)
    #print("  --Biases: ", layer.get_weights()[1]) # biases

print("")
print('---------- Evaluation on Training Data-----------')
print(classification_report(y_train, pred_labels_tr))
print("")

print('---------- Evaluation on Test Data-----------')
print(classification_report(y_test, pred_labels_te))
print("")
```

OUTPUT:

Exercise:

Design and implement an Image classification model to classify a dataset of images using Deep Feed Forward NN. Record the accuracy corresponding to the number of epochs 5, 50. Use the CIFAR10/Fashion MNIST datasets. [You can use CIFAR10 available in keras package]. Make the necessary changes whenever required. Below note down only the changes made and the accuracies obtained.

Week-5

Design and implement a CNN model (with 2 layers of convolutions) to classify multi category image datasets. Record the accuracy corresponding to the number of epochs. Use the MNIST, CIFAR-10 datasets.

Program

```
import keras
from keras.datasets import mnist
from keras.layers import Dense, Activation, Flatten, Conv2D, MaxPooling2D
from keras.models import Sequential
from keras.utils import to_categorical
import numpy as np
import matplotlib.pyplot as plt
(train_X,train_Y), (test_X,test_Y) = mnist.load_data()
train_X = train_X.reshape(-1, 28,28, 1)
test_X = test_X.reshape(-1, 28,28, 1)
train_X.shape
train_X = train_X.astype('float32')
test_X = test_X.astype('float32')
train_X = train_X / 255
test_X = test_X / 255
train_Y_one_hot = to_categorical(train_Y)
test_Y_one_hot = to_categorical(test_Y)
model = Sequential()
model.add(Conv2D(64, (3,3), input_shape=(28, 28, 1)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(64, (3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(64))
model.add(Dense(10))
model.add(Activation('softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
optimizer=keras.optimizers.Adam(),metrics=['accuracy'])
model.fit(train_X, train_Y_one_hot, batch_size=64, epochs=10)
test_loss, test_acc = model.evaluate(test_X, test_Y_one_hot)
print('Test loss', test_loss)
print('Test accuracy', test_acc)
predictions = model.predict(test_X)
print(np.argmax(np.round(predictions[0])))
plt.imshow(test_X[0].reshape(28, 28), cmap = plt.cm.binary)
plt.show()
```

OUTPUT:

Exercise:

Design and implement a CNN model (with 2 layers of convolutions) to classify multi category image datasets. Record the accuracy corresponding to the number of epochs 10, 100. Use the CIFAR10/Fashion MNIST datasets. Make the necessary changes whenever required. Below note down only the changes made and the accuracies obtained.

Week-6

Design and implement a CNN model (with 4+ layers of convolutions) to classify multi category image datasets. Record the accuracy corresponding to the number of epochs. Use the Fashion MNIST datasets. Record the time required to run the program, using CPU as well as using GPU in Colab.

Program-

```
import keras
from keras.datasets import fashion_mnist
from keras.layers import Dense, Activation, Flatten, Conv2D, MaxPooling2D
from keras.models import Sequential
from keras.utils import to_categorical
import numpy as np
import matplotlib.pyplot as plt

(train_X,train_Y), (test_X,test_Y) = fashion_mnist.load_data()

train_X = train_X.reshape(-1, 28,28, 1)
test_X = test_X.reshape(-1, 28,28, 1)

train_X = train_X.astype('float32')
test_X = test_X.astype('float32')
train_X = train_X / 255
test_X = test_X / 255

train_Y_one_hot = to_categorical(train_Y)
test_Y_one_hot = to_categorical(test_Y)

model = Sequential()

model.add(Conv2D(256, (3,3), input_shape=(28, 28, 1)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(128, (3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64, (3,3), input_shape=(28, 28, 1)))
model.add(Activation('relu'))
#model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(28, (3,3)))
model.add(Activation('relu'))
#model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())
model.add(Dense(64))

model.add(Dense(10))
model.add(Activation('softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
optimizer=keras.optimizers.Adam(),metrics=['accuracy'])

model.fit(train_X, train_Y_one_hot, batch_size=64, epochs=5)

test_loss, test_acc = model.evaluate(test_X, test_Y_one_hot)
print('Test loss', test_loss)
print('Test accuracy', test_acc)

predictions = model.predict(test_X)
print(np.argmax(np.round(predictions[0])))

plt.imshow(test_X[0].reshape(28, 28), cmap = plt.cm.binary)
plt.show()
```

OUTPUT:

Exercise:

Design and implement a CNN model (with 4+ layers of convolutions) to classify multi category image datasets. Use the MNIST/ CIFAR-10 datasets. Set the No. of Epoch as 5, 10 and 20. Make the necessary changes whenever required. Record the accuracy corresponding to the number of epochs. Record the time required to run the program, using CPU as well as using GPU in Colab. Below note down only the changes made and the accuracies obtained.

Week-7

Design and implement a CNN model (with 2+ layers of convolutions) to classify multi category image datasets. Use the concept of padding and Batch Normalization while designing the CNN model. Record the accuracy corresponding to the number of epochs. Use the Fashion MNIST datasets.

Program

```
# Batch-Normalization and padding

import keras
from keras.datasets import fashion_mnist
from keras.layers import Dense, Activation, Flatten, Conv2D, MaxPooling2D,
BatchNormalization
from keras.models import Sequential
from keras.utils import to_categorical
import numpy as np
import matplotlib.pyplot as plt

(train_X,train_Y), (test_X,test_Y) = fashion_mnist.load_data()

train_X = train_X.reshape(-1, 28,28, 1)
test_X = test_X.reshape(-1, 28,28, 1)

train_X = train_X.astype('float32')
test_X = test_X.astype('float32')
train_X = train_X / 255
test_X = test_X / 255

train_Y_one_hot = to_categorical(train_Y)
test_Y_one_hot = to_categorical(test_Y)

model = Sequential()

model.add(Conv2D(256, (3,3), input_shape=(28, 28, 1),padding='same'))
model.add(Activation('relu'))
BatchNormalization()
model.add(MaxPooling2D(pool_size=(2,2) ,padding='same'))

model.add(Conv2D(128, (3,3),padding='same'))
model.add(Activation('relu'))
#BatchNormalization()
model.add(MaxPooling2D(pool_size=(2,2) ,padding='same'))

model.add(Conv2D(64, (3,3), input_shape=(28, 28, 1,padding='same'))
model.add(Activation('relu'))
#BatchNormalization()
model.add(MaxPooling2D(pool_size=(2,2),padding='same'))

model.add(Conv2D(28, (3,3)))
model.add(Activation('relu'))
```

```
#BatchNormalization()
model.add(MaxPooling2D(pool_size=(2,2),padding='same'))

model.add(Flatten())
model.add(Dense(64))

model.add(Dense(10))
model.add(Activation('softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
optimizer=keras.optimizers.Adam(),metrics=['accuracy'])

model.fit(train_X, train_Y_one_hot, batch_size=64, epochs=5)

test_loss, test_acc = model.evaluate(test_X, test_Y_one_hot)
print('Test loss', test_loss)
print('Test accuracy', test_acc)

predictions = model.predict(test_X)
print(np.argmax(np.round(predictions[0])))

plt.imshow(test_X[0].reshape(28, 28), cmap = plt.cm.binary)
plt.show()
```

OUTPUT:

Exercise:

Design and implement a CNN model (with 2+ layers of convolutions) to classify multi category image datasets. Use the concept of Batch-Normalization and padding while designing the CNN model. Record the accuracy corresponding to the number of epochs 5, 25, 225. Make the necessary changes whenever required. Use the MNIST/CIFAR-10 datasets. Below note down only the changes made and the accuracies obtained.

Week-8

Design and implement a CNN model (with 4+ layers of convolutions) to classify multi category image datasets. Use the concept of regularization and dropout while designing the CNN model. Use the Fashion MNIST datasets. Record the Training accuracy and Test accuracy corresponding to the following architectures:

- **a. Base Model**
- **b. Model with L1 Regularization**
- **c. Model with L2 Regularization**
- **d. Model with Dropout**
- **e. Model with both L2 (or L1) and Dropout**

Program

a. Base Model: Modify the b. experiment program commenting on kernel_regularizer=l1(0.01) function. See the below program for reference.

b.

```
# L1 Regularizer
import keras
from keras.datasets import fashion_mnist
from keras.layers import Dense, Activation, Flatten, Conv2D, MaxPooling2D
from keras.models import Sequential
from keras.regularizers import l1
from keras.utils import to_categorical
import numpy as np
import matplotlib.pyplot as plt

(train_X,train_Y), (test_X,test_Y) = fashion_mnist.load_data()

train_X = train_X.reshape(-1, 28,28, 1)
test_X = test_X.reshape(-1, 28,28, 1)

train_X = train_X.astype('float32')
test_X = test_X.astype('float32')
train_X = train_X / 255
test_X = test_X / 255

train_Y_one_hot = to_categorical(train_Y)
test_Y_one_hot = to_categorical(test_Y)

model = Sequential()

model.add(Conv2D(256,(3,3),input_shape=(28, 28, 1),kernel_regularizer=l1(0.01)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(128, (3,3),kernel_regularizer=l1(0.01)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(64, (3,3), input_shape=(28, 28, 1),
                 #kernel_regularizer=l1(0.01)
))
model.add(Activation('relu'))
#model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(28, (3,3),
                 #kernel_regularizer=l1(0.01)
))
model.add(Activation('relu'))
#model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())
model.add(Dense(64))

model.add(Dense(10))
model.add(Activation('softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
optimizer=keras.optimizers.Adam(),metrics=['accuracy'])

model.fit(train_X, train_Y_one_hot, epochs=5)

test_loss, test_acc = model.evaluate(test_X, test_Y_one_hot)
print('Test loss', test_loss)
print('Test accuracy', test_acc)

predictions = model.predict(test_X)
print(np.argmax(np.round(predictions[0])))

plt.imshow(test_X[0].reshape(28, 28), cmap = plt.cm.binary)
plt.show()
```

c.

```
# L2 regularizer
import keras
from keras.datasets import fashion_mnist
from keras.layers import Dense, Activation, Flatten, Conv2D, MaxPooling2D
from keras.models import Sequential
from keras.regularizers import l2
from keras.utils import to_categorical
import numpy as np
import matplotlib.pyplot as plt

(train_X,train_Y), (test_X,test_Y) = fashion_mnist.load_data()

train_X = train_X.reshape(-1, 28,28, 1)
```

```
test_X = test_X.reshape(-1, 28,28, 1)

train_X = train_X.astype('float32')
test_X = test_X.astype('float32')
train_X = train_X / 255
test_X = test_X / 255

train_Y_one_hot = to_categorical(train_Y)
test_Y_one_hot = to_categorical(test_Y)

model = Sequential()

model.add(Conv2D(256,(3,3),input_shape=(28,28,1), kernel_regularizer=l2(0.01)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(128, (3,3),
                 #kernel_regularizer=l2(0.01)
))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64, (3,3), input_shape=(28, 28, 1),
                 #kernel_regularizer=l2(0.01)
))
model.add(Activation('relu'))
#model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(28, (3,3),
                 #kernel_regularizer=l2(0.01)
))
model.add(Activation('relu'))
#model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())
model.add(Dense(64))

model.add(Dense(10))
model.add(Activation('softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,optimizer=keras.optimizers.Ada
m(),metrics=['accuracy'])

model.fit(train_X, train_Y_one_hot, epochs=5)

test_loss, test_acc = model.evaluate(test_X, test_Y_one_hot)
print('Test loss', test_loss)
print('Test accuracy', test_acc)
```

```
predictions = model.predict(test_X)
print(np.argmax(np.round(predictions[0])))

plt.imshow(test_X[0].reshape(28, 28), cmap = plt.cm.binary)
plt.show()
```

d.

```
 #Dropout

import keras
from keras.datasets import fashion_mnist
from keras.layers import Dense, Activation, Flatten, Conv2D, MaxPooling2D, Dropout
from keras.models import Sequential
from keras.utils import to_categorical
import numpy as np
import matplotlib.pyplot as plt

(train_X,train_Y), (test_X,test_Y) = fashion_mnist.load_data()

train_X = train_X.reshape(-1, 28,28, 1)
test_X = test_X.reshape(-1, 28,28, 1)

train_X = train_X.astype('float32')
test_X = test_X.astype('float32')
train_X = train_X / 255
test_X = test_X / 255

train_Y_one_hot = to_categorical(train_Y)
test_Y_one_hot = to_categorical(test_Y)

model = Sequential()

model.add(Conv2D(256, (3,3), input_shape=(28, 28, 1)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
Dropout(0.20)

model.add(Conv2D(128, (3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
#Dropout(0.20)

model.add(Conv2D(64, (3,3), input_shape=(28, 28, 1)))
model.add(Activation('relu'))
#model.add(MaxPooling2D(pool_size=(2,2)))
#Dropout(0.20)

model.add(Conv2D(28, (3,3)))
model.add(Activation('relu'))
```

```
#model.add(MaxPooling2D(pool_size=(2,2)))
#Dropout(0.20)
model.add(Flatten())
model.add(Dense(64))

model.add(Dense(10))
model.add(Activation('softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
optimizer=keras.optimizers.Adam(),metrics=['accuracy'])

model.fit(train_X, train_Y_one_hot, batch_size=64, epochs=5)

test_loss, test_acc = model.evaluate(test_X, test_Y_one_hot)
print('Test loss', test_loss)
print('Test accuracy', test_acc)

predictions = model.predict(test_X)
print(np.argmax(np.round(predictions[0])))

plt.imshow(test_X[0].reshape(28, 28), cmap = plt.cm.binary)
plt.show()
```

e.

```
# L2 regularizer and Dropout
import keras
from keras.datasets import fashion_mnist
from keras.layers import Dense, Activation, Flatten, Conv2D, MaxPooling2D
from keras.models import Sequential
from keras.regularizers import l2
from keras.utils import to_categorical
import numpy as np
import matplotlib.pyplot as plt

(train_X,train_Y), (test_X,test_Y) = fashion_mnist.load_data()

train_X = train_X.reshape(-1, 28,28, 1)
test_X = test_X.reshape(-1, 28,28, 1)

train_X = train_X.astype('float32')
test_X = test_X.astype('float32')
train_X = train_X / 255
test_X = test_X / 255

train_Y_one_hot = to_categorical(train_Y)
test_Y_one_hot = to_categorical(test_Y)
```

```
model = Sequential()

model.add(Conv2D(256, (3,3), input_shape=(28, 28, 1), kernel_regularizer=l2(0.01)
))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
Dropout(0.20)


model.add(Conv2D(128, (3,3),
                 #kernel_regularizer=l2(0.01)
))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64, (3,3), input_shape=(28, 28, 1),
                 #kernel_regularizer=l2(0.01)
))
model.add(Activation('relu'))
#model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(28, (3,3),
                 #kernel_regularizer=l2(0.01)
))
model.add(Activation('relu'))
#model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())
model.add(Dense(64))

model.add(Dense(10))
model.add(Activation('softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
optimizer=keras.optimizers.Adam(),metrics=['accuracy'])

model.fit(train_X, train_Y_one_hot, epochs=5)

test_loss, test_acc = model.evaluate(test_X, test_Y_one_hot)
print('Test loss', test_loss)
print('Test accuracy', test_acc)

predictions = model.predict(test_X)
print(np.argmax(np.round(predictions[0])))

plt.imshow(test_X[0].reshape(28, 28), cmap = plt.cm.binary)
plt.show()
```

OUTPUT:

a. **Base Model:**

b. **Model with L1 Regularization:**

c. **Model with L2 Regularization:**

d. **Model with Dropout:**

e. **Model with both L2 (or L1) and Dropout:**

Exercise:

Design and implement a CNN model (with 4+ layers of convolutions) to classify multi category image datasets. Use the concept of regularization and dropout while designing the CNN model. Use the MNIST dataset. Modify the program as and when needed. Record the Training accuracy and Test accuracy corresponding to the following architectures:

a. **Base Model**
b. **Model with both L2 (or L1) and Dropout**

Week-9

Use the concept of data augmentation to increase the data size from a single image.

Program-

```
#data augmentation on a single image

from numpy import expand_dims
from tensorflow.keras.preprocessing import image
#from keras.preprocessing.image import img_to_array
from keras.preprocessing.image import ImageDataGenerator
from matplotlib import pyplot
# load the image
img = image.load_img('/content/gdrive/My Drive/data/train/cat.png')
# convert to numpy array
data  =  image.img_to_array(img)
# expand dimension to one sample
samples = expand_dims(data, 0)
# create image data augmentation generator
datagen = ImageDataGenerator(width_shift_range=[-100,100])
# prepare iterator
it = datagen.flow(samples, batch_size=1)
# generate samples and plot
for i in range(9):
 # define subplot
 pyplot.subplot(330 + 1 + i)
 # generate batch of images
 batch = it.next()
 # convert to unsigned integers for viewing
 image = batch[0].astype('uint8')
 # plot raw pixel data
 pyplot.imshow(image)
# show the figure
pyplot.show()
```

OUTPUT:

Exercise:

Use the concept of data augmentation to increase the data size from a single image. Use any random image of your choice. Apply variations of `ImageDataGenerator` () function on arguments height_shift_range=0.5, horizontal_flip=True, rotation_range=90, brightness_range=[0.2,1.0], zoom_range=[0.5,1.0] etc. and analyze the output images.

Week-10

Design and implement a CNN model to classify CIFAR10 image dataset. Use the concept of Data Augmentation while designing the CNN model. Record the accuracy corresponding to the number of epochs.

```
# data augmentation with flow function

from __future__ import print_function

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D

import matplotlib.pyplot as plt
%matplotlib inline

# The data, shuffled and split between train and test sets:
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

num_classes = 10

y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255

# Let's build a CNN using Keras' Sequential capabilities

model_1 = Sequential()


## 5x5 convolution with 2x2 stride and 32 filters
model_1.add(Conv2D(32, (5, 5), strides = (2,2), padding='same',
                 input_shape=x_train.shape[1:]))
model_1.add(Activation('relu'))

## Another 5x5 convolution with 2x2 stride and 32 filters
model_1.add(Conv2D(32, (5, 5), strides = (2,2)))
model_1.add(Activation('relu'))
```

```
## 2x2 max pooling reduces to 3 x 3 x 32
model_1.add(MaxPooling2D(pool_size=(2, 2)))
model_1.add(Dropout(0.25))

## Flatten turns 3x3x32 into 288x1
model_1.add(Flatten())
model_1.add(Dense(512))
model_1.add(Activation('relu'))
model_1.add(Dropout(0.5))
model_1.add(Dense(num_classes))
model_1.add(Activation('softmax'))

model_1.summary()

batch_size = 32

# initiate RMSprop optimizer
opt = tf.keras.optimizers.legacy.RMSprop(lr=0.0005, decay=1e-6)

# Let's train the model using RMSprop
model_1.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

datagen = ImageDataGenerator(
    featurewise_center=False, # set input mean to 0 over the dataset
    samplewise_center=False, # set each sample mean to 0
    featurewise_std_normalization=False,  # divide inputs by std of the dataset
    samplewise_std_normalization=False, # divide each input by its std
    zca_whitening=False,  # apply ZCA whitening
    rotation_range=0, # randomly rotate images in the range (degrees, 0 to 180)
    width_shift_range=0.1,  # randomly shift images horizontally (fraction of total
width)
    height_shift_range=0.1,  # randomly shift images vertically (fraction of total
height)
    horizontal_flip=True, # randomly flip images
    vertical_flip=False)  # randomly flip images

datagen.fit(x_train)      # This computes any statistics that may be needed (e.g.
for centering) from the training set.


# Fit the model on the batches generated by datagen.flow().
model_1.fit(datagen.flow(x_train, y_train,
                                 batch_size=batch_size),
                    steps_per_epoch=x_train.shape[0] // batch_size,
                    epochs=5,
                    validation_data=(x_test, y_test))

test_loss, test_acc = model_1.evaluate(x_test, y_test)
```

```
print('Test loss', test_loss)
print('Test accuracy', test_acc)
```

OUTPUT:

Exercise:

Can you make the above model do better on the same dataset? Can you make it do worse? Experiment with different settings of the data augmentation while designing the CNN model. Record the accuracy mentioning the modified settings of data augmentation.

Week-11

Implement the standard LeNet CNN architecture model to classify multi category image dataset (MNIST) and check the accuracy.

Program-

```
# LeNet

import tensorflow as tf
from tensorflow import keras
import numpy as np
(train_x, train_y), (test_x, test_y) = keras.datasets.mnist.load_data()
train_x = train_x / 255.0
test_x = test_x / 255.0
train_x = tf.expand_dims(train_x, 3)
test_x = tf.expand_dims(test_x, 3)

val_x = train_x[:5000]
val_y = train_y[:5000]

lenet_5_model = keras.models.Sequential([
    keras.layers.Conv2D(6, kernel_size=5, strides=1, activation='tanh',
input_shape=train_x[0].shape, padding='same'), #C1
    keras.layers.AveragePooling2D(), #S2
    keras.layers.Conv2D(16, kernel_size=5, strides=1, activation='tanh',
padding='valid'), #C3
    keras.layers.AveragePooling2D(), #S4
    keras.layers.Conv2D(120, kernel_size=5, strides=1, activation='tanh',
padding='valid'), #C5
    keras.layers.Flatten(), #Flatten
    keras.layers.Dense(84, activation='tanh'), #F6
    keras.layers.Dense(10, activation='softmax') #Output layer
])

lenet_5_model.compile(optimizer='adam',
loss=keras.losses.sparse_categorical_crossentropy, metrics=['accuracy'])
lenet_5_model.fit(train_x, train_y, epochs=5, validation_data=(val_x, val_y))
lenet_5_model.evaluate(test_x, test_y)
```

OUTPUT:

Exercise:

Implement the standard LeNet CNN architecture model to classify multi category image dataset (Fashion MNIST) and check the accuracy. Below note down only the changes made and the accuracies obtained for epochs 5, 50, 250.

Week-12

Implement the standard VGG 16 CNN architecture model to classify cat and dog image dataset and check the accuracy.

Program-

```
# VGG16

import keras,os
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPool2D, Flatten
from keras.preprocessing.image import ImageDataGenerator
import numpy as np
trdata = ImageDataGenerator()
traindata = trdata.flow_from_directory(directory="/content/gdrive/My
Drive/training_set",target_size=(224,224))
tsdata = ImageDataGenerator()
testdata = tsdata.flow_from_directory(directory="/content/gdrive/My Drive/test_set",
target_size=(224,224))

model = Sequential()
model.add(Conv2D(input_shape=(224,224,3),filters=64,kernel_size=(3,3),padding="same"
,activation="relu"))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))

model.add(Flatten())
model.add(Dense(units=4096,activation="relu"))
model.add(Dense(units=4096,activation="relu"))
model.add(Dense(units=2, activation="softmax"))

from keras.optimizers import Adam
opt = Adam(lr=0.001)
model.compile(optimizer=opt, loss=keras.losses.categorical_crossentropy,
metrics=['accuracy'])
```

```
model.summary()

from keras.callbacks import ModelCheckpoint, EarlyStopping checkpoint =
ModelCheckpoint("vgg16_1.h5", monitor='val_acc', verbose=1,save_best_only=True,
save_weights_only=False, mode='auto', period=1)
early = EarlyStopping(monitor='val_acc', min_delta=0, patience=20, verbose=1,
mode='auto')
hist = model.fit_generator(steps_per_epoch=100,generator=traindata, validation_data=
testdata, validation_steps=10,epochs=5,callbacks=[checkpoint,early])

import matplotlib.pyplot as plt
plt.plot(hist.history["accuracy"])
plt.plot(hist.history['val_accuracy'])
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title("model accuracy")
plt.ylabel("Accuracy")
plt.xlabel("Epoch")
plt.legend(["Accuracy","Validation Accuracy","loss","Validation Loss"])
plt.show()
```

OUTPUT:

Exercise:

Implement the standard VGG 19 CNN architecture model to classify cat and dog image dataset and check the accuracy. Make the necessary changes whenever required.

Week-13

Implement RNN for sentiment analysis on movie reviews.

Program-

```
# RNN sentiment analysis on movie reviews

from keras.datasets import imdb
from keras.preprocessing.text import Tokenizer
from keras.utils import pad_sequences
from keras import Sequential
from keras.layers import
Dense,SimpleRNN,Embedding,Flatten(X_train,y_train),(X_test
,y_test) = imdb.load_data() X_train =
pad_sequences(X_train,padding='post',maxlen=50) X_test =
pad_sequences(X_test,padding='post',maxlen=50)
X_train.shape
model = Sequential()
#model.add(Embedding(10000, 2))
model.add(SimpleRNN(32,input_shape=(50,1), return_sequences=False))
model.add(Dense(1, activation='sigmoid'))

model.summary()
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
model.fit(X_train, y_train,epochs=5,validation_data=(X_test,y_test))
test_loss, test_acc = model.evaluate(X_test, y_test)
print('Test loss', test_loss)
print('Test accuracy', test_acc)
```

OUTPUT:

Exercise:
Implement RNN for sentiment analysis on movie reviews. Use the concept of Embedding layer.

Week-14

Implement Bi-directional LSTM for sentiment analysis on movie reviews.

Program-

```
# Bi directional LSTM

import numpy as np
from keras.preprocessing import sequence
from keras.utils import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Dropout, Embedding, LSTM, Bidirectional
from keras.datasets import imdb

n_unique_words = 10000 # cut texts after this number of words
maxlen = 200
batch_size = 128
(x_train, y_train),(x_test, y_test) = imdb.load_data(num_words=n_unique_words)
x_train = pad_sequences(x_train, maxlen=maxlen)
x_test = pad_sequences(x_test, maxlen=maxlen)
y_train = np.array(y_train)
y_test = np.array(y_test)

model = Sequential()
model.add(Embedding(n_unique_words, 128, input_length=maxlen))
model.add(Bidirectional(LSTM(64)))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
history=model.fit(x_train, y_train, batch_size=batch_size, epochs=10,
validation_data=[x_test, y_test])
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test loss', test_loss)
print('Test accuracy', test_acc)
print(history.history['loss'])
print(history.history['accuracy'])
from matplotlib import pyplot
pyplot.plot(history.history['loss'])
pyplot.plot(history.history['accuracy'])
pyplot.title('model loss vs accuracy')
pyplot.xlabel('epoch')
pyplot.legend(['loss', 'accuracy'], loc='upper right')
pyplot.show()
```

OUTPUT:

Exercise:
Implement Bi-directional LSTM on a suitable dataset of your choice. Modify the program as needed.

Week-15

Implement Generative Adversarial Networks to generate realistic Images. Use MNIST dataset.

Program-

```
# loading the mnist dataset
from tensorflow.keras.datasets.mnist import load_data

# load the images into memory
(trainX, trainy), (testX, testy) = load_data()

# summarize the shape of the dataset
print('Train', trainX.shape, trainy.shape)
print('Test', testX.shape, testy.shape)

#plot of 25 images from the MNIST training dataset, arranged in a 5×5 square.


from tensorflow.keras.datasets.mnist import load_data
from matplotlib import pyplot
# load the images into memory
(trainX, trainy), (testX, testy) = load_data()
# plot images from the training dataset
for i in range(25):
  # define subplot
  pyplot.subplot(5, 5, 1 + i)
  # turn off axis
  pyplot.axis('off')
  # plot raw pixel data
  pyplot.imshow(trainX[i], cmap='gray_r')
pyplot.show()

import glob
import imageio
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
from tensorflow.keras import layers
import time
import tensorflow as tf

from IPython import display
(train_images, train_labels), (_, _) = tf.keras.datasets.mnist.load_data()
train_images = train_images.reshape(train_images.shape[0], 28, 28,
1).astype('float32')
train_images = (train_images - 127.5) / 127.5 # Normalize the images to [-1, 1]
BUFFER_SIZE = 60000
BATCH_SIZE = 256
# Batch and shuffle the data
```

```
train_dataset =
tf.data.Dataset.from_tensor_slices(train_images).shuffle(BUFFER_SIZE).batch(BATCH_SIZ
E)
# input 7*7*256 (low resolution version of the output image)
# outputs a single 28×28 grayscale image
# this generator takes a vector of size 100 and first reshape that into (7, 7, 128)
vector then applied transpose
# convolution in combination with batch normalization.


def  make_generator_model():
    model = tf.keras.Sequential()
    model.add(layers.Dense(7*7*256, use_bias=False, input_shape=(100,)))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Reshape((7, 7, 256)))
    assert model.output_shape == (None, 7, 7, 256)  # Note: None is the batch size
    model.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same',
use_bias=False))
    assert model.output_shape == (None, 7, 7, 128)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

   # upsample to 14x14
    model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same',
use_bias=False))
    assert model.output_shape == (None, 14, 14, 64)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    # upsample to 28x28
    model.add(layers.Conv2DTranspose(1, (5, 5), strides=(2, 2), padding='same',
use_bias=False, activation='tanh'))
    assert model.output_shape == (None, 28, 28, 1)

    return model
# sample image generated by the the generator
generator = make_generator_model()

noise = tf.random.normal([1, 100]) #latent space
generated_image = generator(noise, training=False)

plt.imshow(generated_image[0, :, :, 0], cmap='gray')

# Input to discriminator = 28*28*1 grayscale image
# Output  binary prediction (image is real (class=1) or fake (class=0))
# no pooling layers
# single node in the output layer with the sigmoid activation function to predict
whether the input sample is real or fake.
```

```
# Downsampling from 28×28 to 14×14, then to 7×7, before the model makes an output
prediction
def make_discriminator_model():
    model = tf.keras.Sequential()

    model.add(layers.Conv2D(64, (5, 5), strides=(2, 2),
padding='same',input_shape=[28, 28, 1])) #2×2 stride to downsample
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
#downsampling  2×2 stride to downsample
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Flatten())  # classifier real (class=1) or fake (class=0))
    model.add(layers.Dense(1, activation='sigmoid'))

    return model

discriminator = make_discriminator_model()
decision = discriminator(generated_image)
print (decision)
# This method returns a helper function to compute cross entropy loss
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
def discriminator_loss(real_output, fake_output):
    real_loss = cross_entropy(tf.ones_like(real_output), real_output)
    fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
    total_loss = real_loss + fake_loss
    return total_loss
def generator_loss(fake_output):
    return cross_entropy(tf.ones_like(fake_output), fake_output)
generator_optimizer = tf.keras.optimizers.Adam(1e-4)
discriminator_optimizer = tf.keras.optimizers.Adam(1e-4)
checkpoint_dir = './training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt")
checkpoint = tf.train.Checkpoint(generator_optimizer=generator_optimizer,
                                 discriminator_optimizer=discriminator_optimizer,
                                 generator=generator,
                                 discriminator=discriminator)
EPOCHS = 5
noise_dim = 100
num_examples_to_generate = 16

# You will reuse this seed overtime (so it's easier)
# to visualize progress in the animated GIF)
seed = tf.random.normal([num_examples_to_generate, noise_dim])
# Notice the use of `tf.function`
# This annotation causes the function to be "compiled".
@tf.function
```

```
def train_step(images):
    noise = tf.random.normal([BATCH_SIZE, noise_dim])

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
      generated_images = generator(noise, training=True)

      real_output = discriminator(images, training=True)
      fake_output = discriminator(generated_images, training=True)

      gen_loss = generator_loss(fake_output)
      disc_loss = discriminator_loss(real_output, fake_output)

    gradients_of_generator = gen_tape.gradient(gen_loss,
generator.trainable_variables)
    gradients_of_discriminator = disc_tape.gradient(disc_loss,
discriminator.trainable_variables)

    generator_optimizer.apply_gradients(zip(gradients_of_generator,
generator.trainable_variables))
    discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator,
discriminator.trainable_variables))

def train(dataset, epochs):
  for epoch in range(epochs):
    start = time.time()

    for image_batch in dataset:
      train_step(image_batch)

    # Produce images for the GIF as you go
    display.clear_output(wait=True)
    generate_and_save_images(generator,
                             epoch + 1,
                             seed)

    # Save the model every 15 epochs
    if (epoch + 1) % 15 == 0:
      checkpoint.save(file_prefix = checkpoint_prefix)

    print ('Time for epoch {} is {} sec'.format(epoch + 1, time.time()-start))

  # Generate after the final epoch
  display.clear_output(wait=True)
  generate_and_save_images(generator,
                           epochs,
                           seed)
def generate_and_save_images(model, epoch, test_input):
  # Notice `training` is set to False.
  # This is so all layers run in inference mode (batchnorm).
  predictions = model(test_input, training=False)
  fig = plt.figure(figsize=(4, 4))

  for i in range(predictions.shape[0]):
      plt.subplot(4, 4, i+1)
      plt.imshow(predictions[i, :, :, 0] * 127.5 + 127.5, cmap='gray')
      plt.axis('off')

  plt.savefig('image_at_epoch_{:04d}.png'.format(epoch))
  plt.show()
train(train_dataset, EPOCHS)
checkpoint.restore(tf.train.latest_checkpoint(checkpoint_dir))
# Display a single image using the epoch number
def display_image(epoch_no):
  return PIL.Image.open('image_at_epoch_{:04d}.png'.format(epoch_no))
display_image(EPOCHS)
import tensorflow_docs.vis.embed as embed
embed.embed_file(anim_file)
```

OUTPUT:

Exercise:

Implement Generative Adversarial Networks to generate realistic Images. Use Fashion MNIST or any human face datasets.

Week-16
Implement Auto encoders for image denoising on MNIST dataset.

Program-

```
#Implement Auto encoders for image denoising on MNIST dataset.

import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import Conv2D,MaxPool2D, UpSampling2D,Dropout
from keras.datasets import mnist
(x_train,y_train),(x_test,y_test) = mnist.load_data()
# to get the shape of the data
print("x_train shape:",x_train.shape)
print("x_test shape", x_test.shape)
plt.figure(figsize = (8,8))
for i in range(25):
  plt.subplot(5,5,i+1)
  plt.title(str(y_train[i]),fontsize = 16, color = 'black', pad = 2)
  plt.imshow(x_train[i], cmap = plt.cm.binary )
  plt.xticks([])
  plt.yticks([])

plt.show()

val_images = x_test[:9000]
test_images = x_test[9000:]
val_images = val_images.astype('float32') / 255.0
val_images = np.reshape(val_images,(val_images.shape[0],28,28,1))

test_images = test_images.astype('float32') / 255.0
test_images = np.reshape(test_images,(test_images.shape[0],28,28,1))

train_images = x_train.astype("float32") / 255.0
train_images = np.reshape(train_images, (train_images.shape[0],28,28,1))
factor = 0.39
train_noisy_images = train_images + factor * np.random.normal(loc = 0.0,scale =
1.0,size = train_images.shape)
val_noisy_images = val_images + factor * np.random.normal(loc = 0.0,scale = 1.0,size
= val_images.shape)
test_noisy_images = test_images + factor * np.random.normal(loc = 0.0,scale =
1.0,size = test_images.shape)

# here maximum pixel value for our images may exceed 1 so we have to clip the images
train_noisy_images = np.clip(train_noisy_images,0.,1.)
```

```
val_noisy_images = np.clip(val_noisy_images,0.,1.)
test_noisy_images = np.clip(test_noisy_images,0.,1.)
plt.figure(figsize = (8,8))

for i in range(25):
      plt.subplot(5,5,i+1)
      plt.title(str(y_train[i]),fontsize = 16, color = 'black', pad = 2)
      plt.imshow(train_noisy_images[i].reshape(1,28,28)[0], cmap = plt.cm.binary )
      plt.xticks([])
      plt.yticks([])

plt.show()
model = Sequential()
# encoder network
model.add(Conv2D(filters = 128, kernel_size = (2,2), activation = 'relu', padding =
'same', input_shape = (28,28,1)))
model.add(tf.keras.layers.BatchNormalization())
model.add(Conv2D(filters = 128, kernel_size = (2,2), activation = 'relu', padding =
'same'))
model.add(tf.keras.layers.BatchNormalization())
model.add(Conv2D(filters = 256, kernel_size = (2,2),strides = (2,2), activation =
'relu', padding = 'same'))
model.add(tf.keras.layers.BatchNormalization())
model.add(Conv2D(filters = 256, kernel_size = (2,2), activation = 'relu', padding =
'same'))
model.add(tf.keras.layers.BatchNormalization())
model.add(Conv2D(filters = 512, kernel_size = (3,3), activation = 'relu', padding =
'same'))
model.add(tf.keras.layers.BatchNormalization())
model.add(Conv2D(filters = 512, kernel_size = (2,2),strides = (2,2), activation =
'relu', padding = 'same'))



# decoder network
model.add(Conv2D(filters = 512, kernel_size = (2,2), activation = 'relu', padding =
'same'))

model.add(tf.keras.layers.Conv2DTranspose(filters = 512, kernel_size = (2,2), strides
= (2,2),activation = 'relu', padding = 'same'))
model.add(tf.keras.layers.BatchNormalization())
model.add(Conv2D(filters = 256, kernel_size = (2,2), activation = 'relu', padding =
'same'))
model.add(tf.keras.layers.BatchNormalization())
model.add(Conv2D(filters = 256, kernel_size = (2,2), activation = 'relu', padding =
'same'))
model.add(tf.keras.layers.BatchNormalization())
model.add(Conv2D(filters = 128, kernel_size = (2,2), activation = 'relu', padding =
'same'))
```

```
model.add(tf.keras.layers.Conv2DTranspose(filters = 128, kernel_size = (2,2),strides
= (2,2), activation = 'relu', padding = 'same'))
model.add(Conv2D(filters = 64, kernel_size = (2,2), activation = 'relu', padding =
'same'))
model.add(tf.keras.layers.BatchNormalization())

model.add(Conv2D(filters = 1, kernel_size = (2,2), activation = 'relu', padding =
'same'))


# to get the summary of the model
model.summary()

OPTIMIZER = tf.keras.optimizers.Adam(learning_rate = 0.001)
LOSS = 'mean_squared_error'
model.compile(optimizer =OPTIMIZER, loss = LOSS, metrics = ['accuracy'])

EPOCHS = 5
BATCH_SIZE = 256
VALIDATION = (val_noisy_images, val_images)
history = model.fit(train_noisy_images, train_images,batch_size = BATCH_SIZE,epochs =
EPOCHS, validation_data = VALIDATION)

plt.subplot(2,1,1)
plt.plot(history.history['loss'], label = 'loss')
plt.plot(history.history['val_loss'], label = 'val_loss')
plt.legend(loc = 'best')
plt.subplot(2,1,2)
plt.plot(history.history['accuracy'], label = 'accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.legend(loc = 'best')
plt.show()

plt.figure(figsize = (18,18))
for i in range(10,19):
    plt.subplot(9,9,i)
    if(i == 14):
        plt.title('Real Images', fontsize = 25, color = 'Green')
    plt.imshow(test_images[i].reshape(1,28,28)[0], cmap = plt.cm.binary)
plt.show()


plt.figure(figsize = (18,18))
for i in range(10,19):
    if(i == 15):
        plt.title('Noised Images', fontsize = 25, color = 'red')
    plt.subplot(9,9,i)
    plt.imshow(test_noisy_images[i].reshape(1,28,28)[0], cmap = plt.cm.binary)
plt.show()
plt.figure(figsize = (18,18))
for i in range(10,19):
    if(i == 15):
        plt.title('Denoised Images', fontsize = 25, color = 'Blue')

    plt.subplot(9,9,i)

plt.imshow(model.predict(test_noisy_images[i].reshape(1,28,28,1)).reshape(1,28,28)[0]
, cmap = plt.cm.binary)
plt.show()
```

OUTPUT:

Exercise:

Implement Auto encoders for image denoising on Fashion MNIST dataset or on any suitable dataset of your choice.