

**SIES (NERUL) COLLEGE OF ARTS, SCIENCE AND COMMERCE NERUL, NAVI MUMBAI-400706**

# DEPARTMENT OF COMPUTER SCIENCE

**MSc (CS) PART-II SEMESTER IV**

Practical Journal

in

**Advanced Deep Learning**

**Course Code:** PSCS402

Submitted by

**Karthik Jayan Nair**

**SEAT NUMBER -**

**ROLL NUMBER – 09**

Prof. in charge of Practical : **Shweta Khubchandani**

For the academic year

**2023-2024**

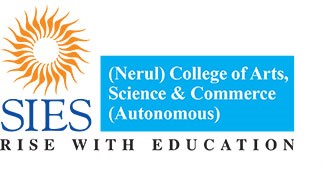
SIES(NERUL) College Of Arts, Science and Commerce

NAAC Re-Accredited ‘A’ Grade

Sri Chandrasekarendra saraswathy vidhyapuram

plot 1-c Sector v

, Nerul Navi Mumbai -400706

****

**CERTIFICATE**

This is to certify that the Practical Journal of subject entitled, **“Advanced Deep Learning”**, is bona fide work of **Nandagopal Kamaraj** of Masters Of Science in Computer Science (Part 2) submitted in partial fulfillment of the requirements for the award of degree of **MASTER OF SCIENCE** in **COMPUTER SCIENCE** from University of Mumbai. It is also to clarify that this is the original work of the candidate done during the academic year 2023-24

Seat Number: Date:

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(Teacher In-Charge) (External Examiner)

**Index**

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Practical** | **Date** |
| 1 | Implement Feed-forward Neural Network and train the network with different optimizers and compare the results. |  |
| 2 | Write a Program to implement regularization to prevent the model from overfitting |  |
| 3 | Implement deep learning for recognizing classes for datasets like CIFAR-10 images for previously unseen images and assign them to one of the 10 classes. |  |
| 4 | Implement deep learning for the Prediction of the autoencoder from the test data (e.g. MNIST data set) |  |
| 5 | Implement Convolutional Neural Network for Digit Recognition on the MNIST Dataset |  |
| 6 | Write a program to implement Transfer Learning on the suitable dataset (e.g. classify the cats versus dogs dataset from Kaggle). |  |
| 7 | Write a program for the Implementation of a Generative Adversarial Network for generating synthetic shapes (like digits). |  |
| 8 | Write a program to implement a simple form of a recurrent neural network.  (i) E.g. (4-to-1 RNN) to show that the quantity of rain on a certain day also depends on the values of the previous day  (ii) LSTM for sentiment analysis on datasets like UMICH SI650 for similar. |  |
| 9 | Write a program for object detection from the image/video. |  |

## Practical No.1

**Aim : Implement Feed-forward Neural Network and train the network with different optimizers and compare the results.**

**Theory :**

A Feedforward Neural Network (FNN) is a type of artificial neural network where connections between the nodes do not form cycles. This characteristic differentiates it from recurrent neural networks (RNNs). The network consists of an input layer, one or more hidden layers, and an output layer. Information flows in one direction—from input to output—hence the name “feedforward.”

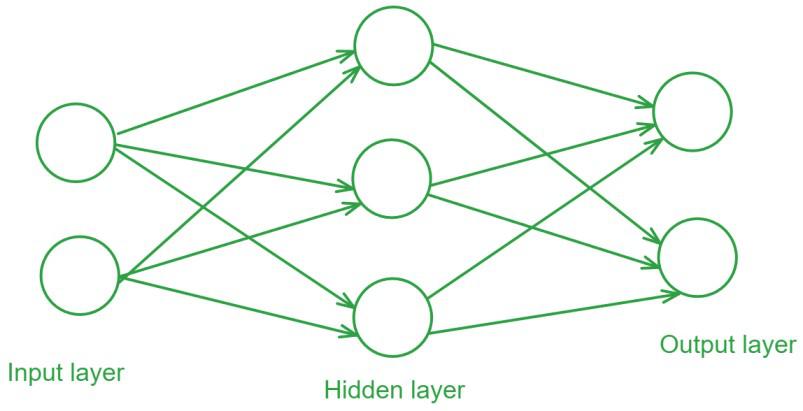
Structure of a Feedforward Neural Network

**Input Layer:** The input layer consists of neurons that receive the input data. Each neuron in the input layer represents a feature of the input data.

**Hidden Layers:** One or more hidden layers are placed between the input and output layers. These layers are responsible for learning the complex patterns in the data. Each neuron in a hidden layer applies a weighted sum of inputs followed by a non-linear activation function.

**Output Layer:** The output layer provides the final output of the network. The number of neurons in this layer corresponds to the number of classes in a classification problem or the number of outputs in a regression problem.

Each connection between neurons in these layers has an associated weight that is adjusted during the training process to minimize the error in predictions.

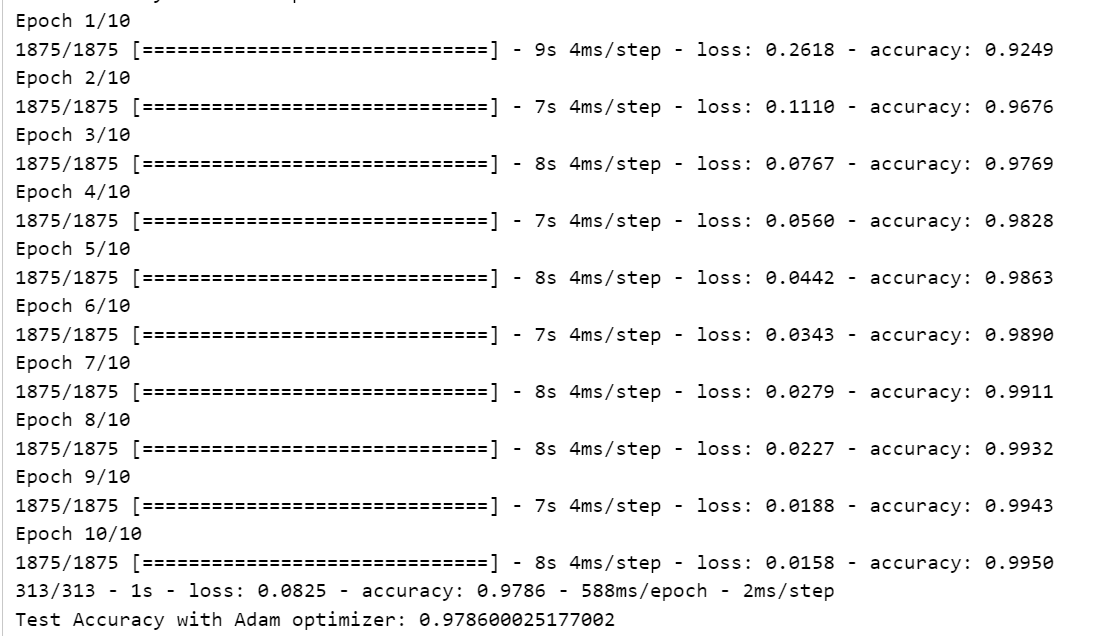


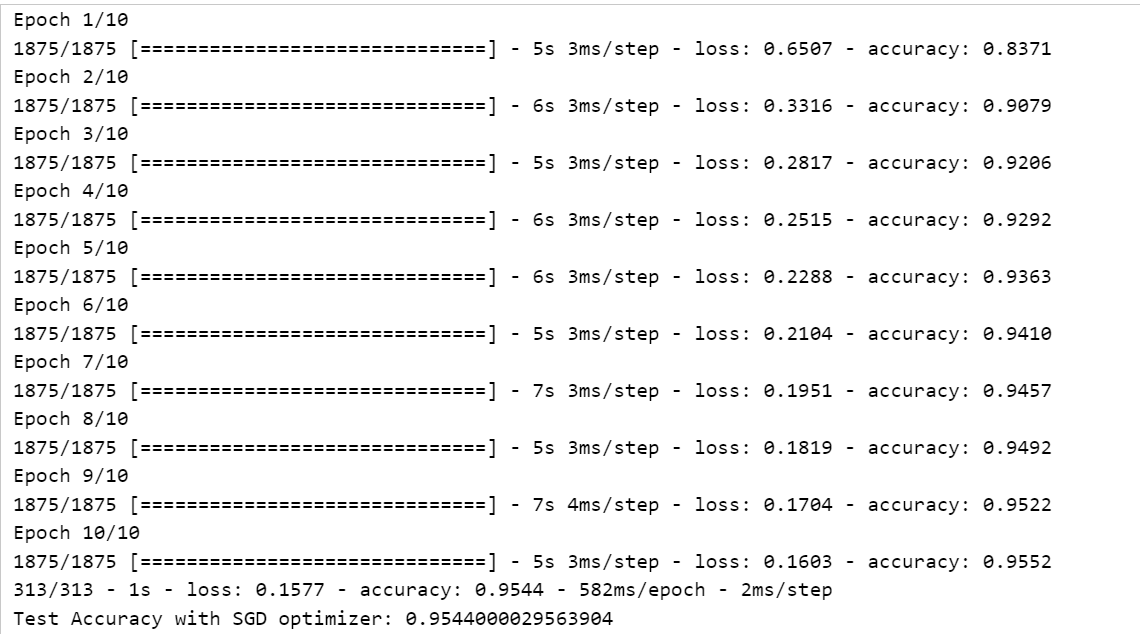
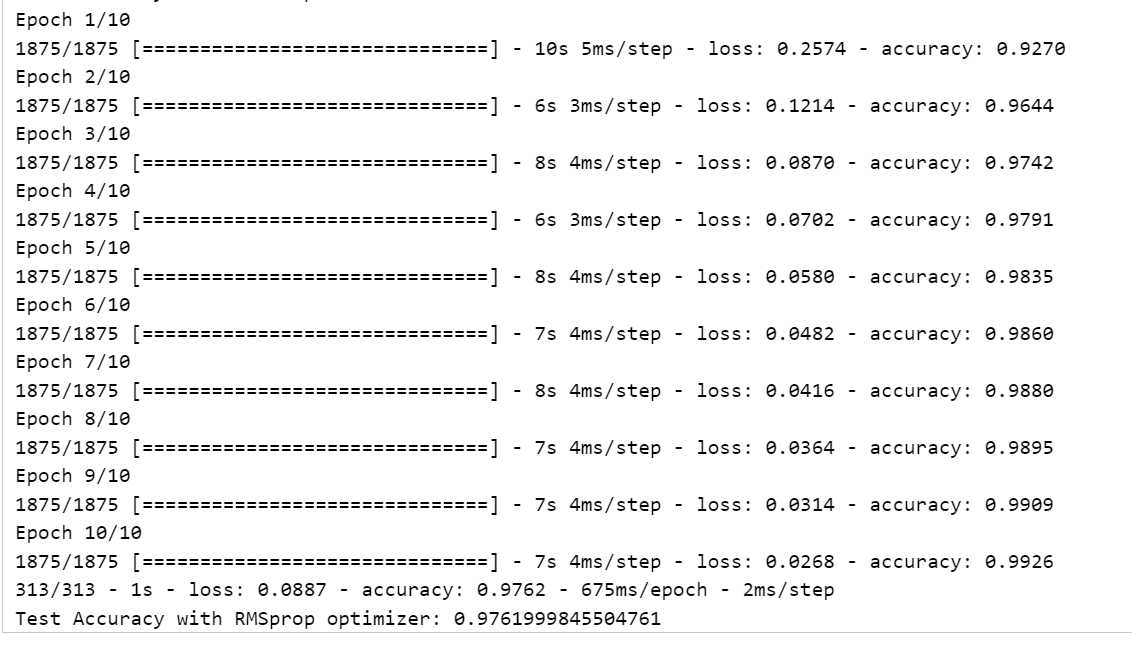
**Program:-**

|  |
| --- |
| import tensorflow as tf  from tensorflow.keras.datasets import mnist  from tensorflow.keras.layers import Dense, Flatten  from tensorflow.keras.models import Sequential |

|  |
| --- |
| import numpy as np  import tensorflow as tf  from tensorflow.keras.datasets import mnist  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense, Flatten  from tensorflow.keras.optimizers import SGD, Adam, RMSprop  (x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()  x\_train, x\_test = x\_train / 255.0, x\_test / 255.0  epochs = 10  batch\_size = 32  loss\_fn = tf.keras.losses.SparseCategoricalCrossentropy()  metrics = ['accuracy']  optimizers = [  SGD(),  Adam(),  RMSprop()  ]  for optimizer in optimizers:  model = Sequential([  Flatten(input\_shape=(28, 28)),  Dense(128, activation='relu'),  Dense(10, activation='softmax')  ])  model.compile(optimizer=optimizer, loss=loss\_fn, metrics=metrics)  model.fit(x\_train, y\_train, epochs=epochs, batch\_size=batch\_size, verbose=1)  test\_loss, test\_acc = model.evaluate(x\_test, y\_test, verbose=2)  print(f'Test Accuracy with {type(optimizer).\_\_name\_\_} optimizer: {test\_acc}') |

**Output :**

****

****

# Practical No.2

## Aim : Write a Program to implement regularization to prevent the model from overfitting

## Theory :

## Regularization is a technique used to prevent overfitting by adding a penalty term to the loss function, discouraging the model from assigning too much importance to individual features or coefficients.

## Complexity Control: Regularization helps control model complexity by preventing overfitting to training data, resulting in better generalization to new data.

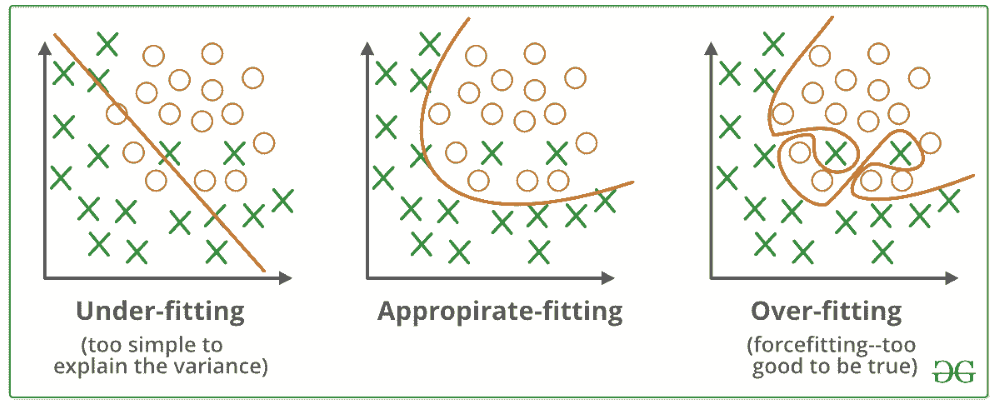
## Preventing Overfitting: One way to prevent overfitting is to use regularization, which penalizes large coefficients and constrains their magnitudes, thereby preventing a model from becoming overly complex and memorizing the training data instead of learning its underlying patterns.

## Balancing Bias and Variance: Regularization can help balance the trade-off between model bias (underfitting) and model variance (overfitting) in machine learning, which leads to improved performance.

## Feature Selection: Some regularization methods, such as L1 regularization (Lasso), promote sparse solutions that drive some feature coefficients to zero. This automatically selects important features while excluding less important ones.

## Handling Multicollinearity: When features are highly correlated (multicollinearity), regularization can stabilize the model by reducing coefficient sensitivity to small data changes.

## Generalization: Regularized models learn underlying patterns of data for better generalization to new data, instead of memorizing specific examples.

****

**Program :**

|  |
| --- |
| import numpy as np  import pandas as pd  from matplotlib import pyplot as plt  from keras.models import Sequential  from keras.layers import Dense, Dropout  from keras import regularizers |

|  |
| --- |
| data = pd.read\_csv('/content/hour.csv') |

|  |
| --- |
| ohe\_features = ['season', 'mnth', 'hr', 'weekday']  for feature in ohe\_features:  dummies = pd.get\_dummies(data[feature], prefix=feature, drop\_first=False)  data = pd.concat([data, dummies], axis=1)  drop\_features = ['instant', 'dteday', 'season', 'weathersit', 'weekday', 'atemp', 'mnth', 'workingday', 'hr', 'casual', 'registered']  data = data.drop(drop\_features, axis=1) |

|  |
| --- |
| norm\_features = ['cnt', 'temp', 'hum', 'windspeed']  scaled\_features = {}  for feature in norm\_features:  mean, std = data[feature].mean(), data[feature].std()  scaled\_features[feature] = [mean, std]  data.loc[:, feature] = (data[feature] - mean) / std  test\_data = data[-31\*24:]  data = data[:-31\*24] |

|  |
| --- |
| target\_fields = ['cnt']  features, targets = data.drop(target\_fields, axis=1), data[target\_fields]  test\_features, test\_targets = test\_data.drop(target\_fields, axis=1), test\_data[target\_fields] |

|  |
| --- |
| x\_train, y\_train = features[:-30\*24], targets[:-30\*24]  x\_val, y\_val = features[-30\*24:], targets[-30\*24:]  x\_train, y\_train = x\_train.values.astype(np.float32), y\_train['cnt'].values.astype(np.float32)  x\_val, y\_val = x\_val.values.astype(np.float32), y\_val['cnt'].values.astype(np.float32) |

|  |
| --- |
| model = Sequential()  model.add(Dense(250, input\_dim=x\_train.shape[1], activation='relu'))  model.add(Dense(150, activation='relu'))  model.add(Dense(50, activation='relu'))  model.add(Dense(25, activation='relu'))  model.add(Dense(1, activation='relu'))  model.compile(loss='mse', optimizer='sgd', metrics=['mse']) |

|  |
| --- |
| n\_epochs = 500  batch\_size = 50  history = model.fit(x\_train, y\_train,  validation\_data=(x\_val, y\_val),  batch\_size=batch\_size, epochs=n\_epochs, verbose=0) |

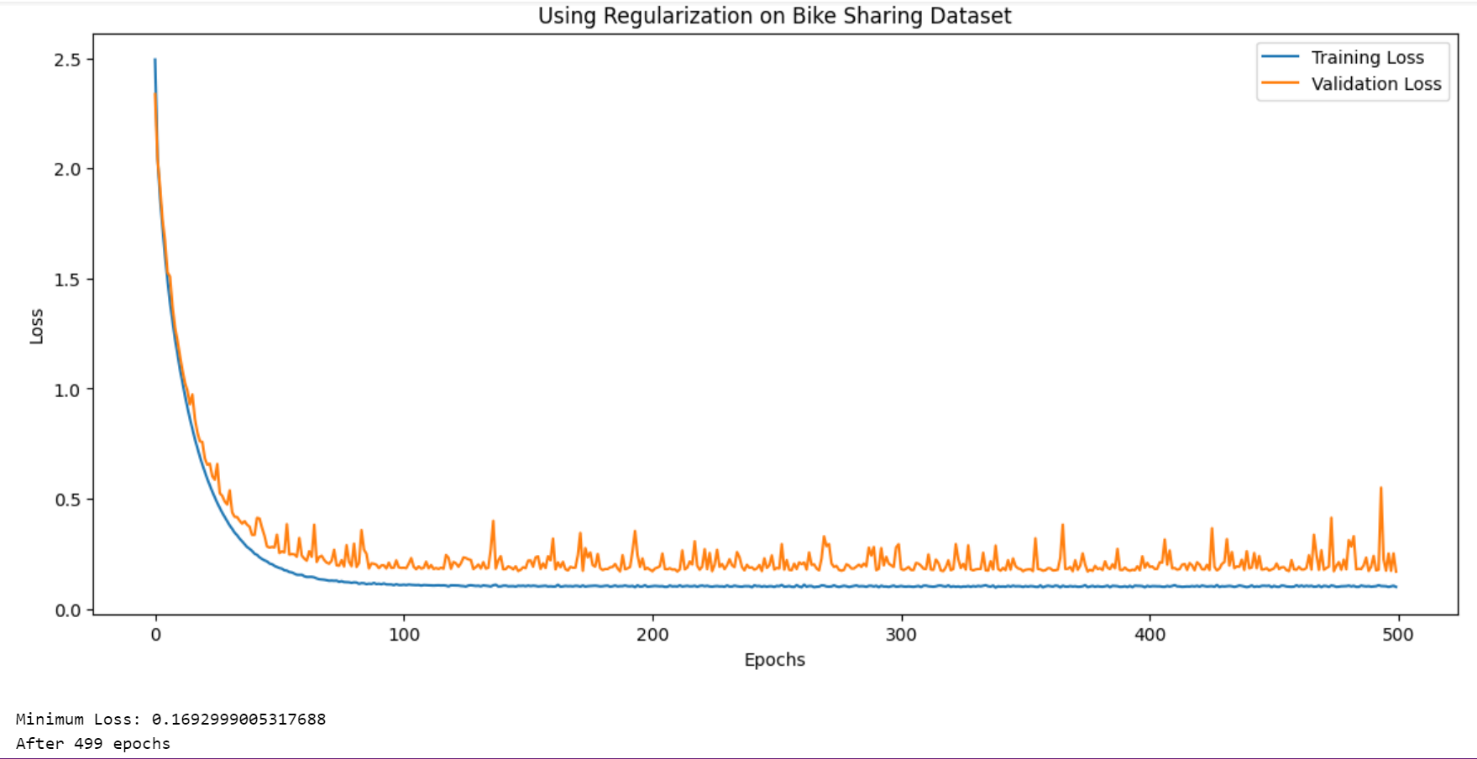
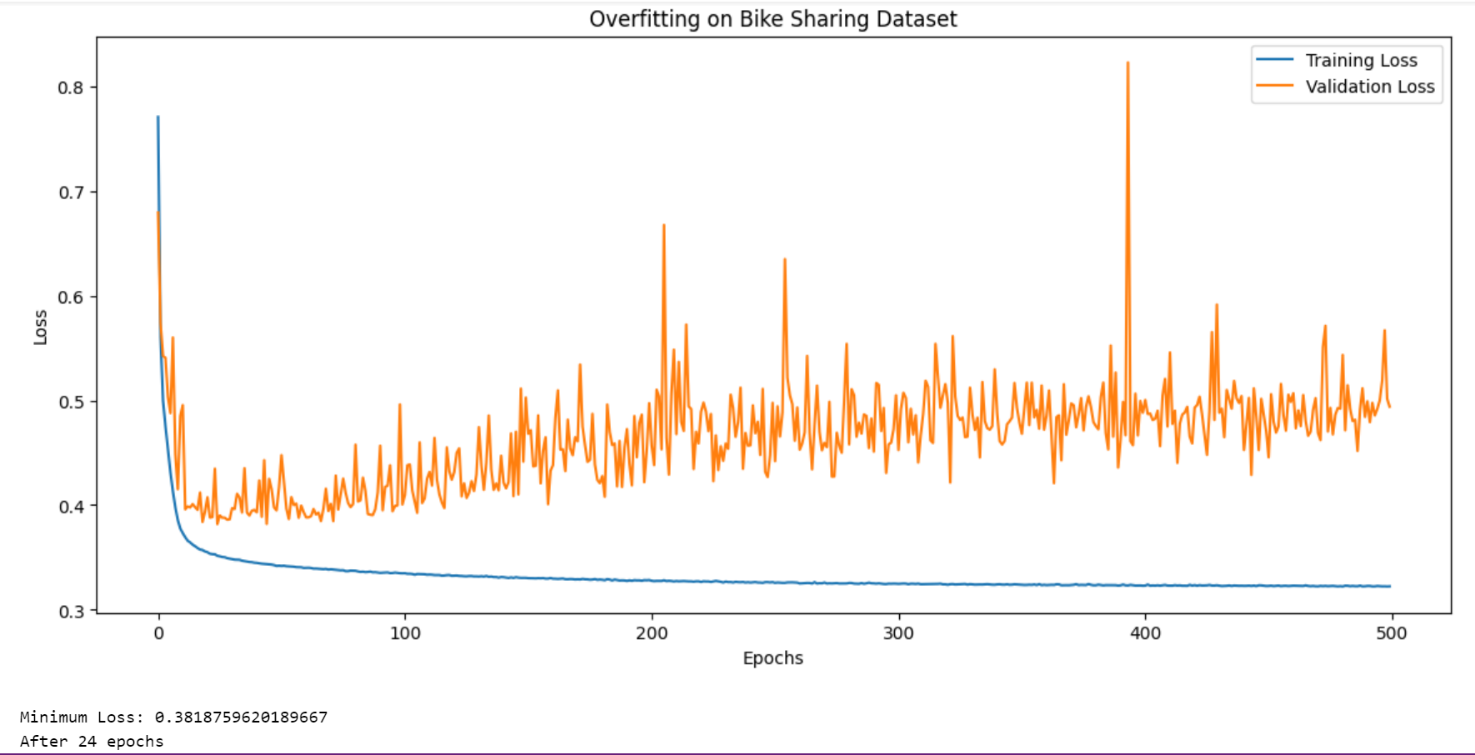
|  |
| --- |
| plt.figure(figsize=(14, 6))  plt.plot(np.arange(len(history.history['loss'])), history.history['loss'], label='Training Loss')  plt.plot(np.arange(len(history.history['val\_loss'])), history.history['val\_loss'], label='Validation Loss')  plt.title('Overfitting on Bike Sharing Dataset')  plt.xlabel('Epochs')  plt.ylabel('Loss')  plt.legend()  plt.show()  print('Minimum Loss:', min(history.history['val\_loss']), '\nAfter', np.argmin(history.history['val\_loss']), 'epochs') |

|  |
| --- |
| model\_reg = Sequential()  model\_reg.add(Dense(250, input\_dim=x\_train.shape[1], activation='relu', kernel\_regularizer=regularizers.l2(0.005)))  model\_reg.add(Dense(150, activation='relu', kernel\_regularizer=regularizers.l2(0.005)))  model\_reg.add(Dense(50, activation='relu', kernel\_regularizer=regularizers.l2(0.005)))  model\_reg.add(Dense(25, activation='relu', kernel\_regularizer=regularizers.l2(0.005)))  model\_reg.add(Dense(1, activation='linear'))  model\_reg.compile(loss='mse', optimizer='sgd', metrics=['mse']) |

|  |
| --- |
| history\_reg = model\_reg.fit(x\_train, y\_train,  validation\_data=(x\_val, y\_val),  batch\_size=batch\_size, epochs=n\_epochs, verbose=0) |

|  |
| --- |
| plt.figure(figsize=(14, 6))  plt.plot(np.arange(len(history\_reg.history['loss'])), history\_reg.history['loss'], label='Training Loss')  plt.plot(np.arange(len(history\_reg.history['val\_loss'])), history\_reg.history['val\_loss'], label='Validation Loss')  plt.title('Using Regularization on Bike Sharing Dataset')  plt.xlabel('Epochs')  plt.ylabel('Loss')  plt.legend()  plt.show()  print('Minimum Loss:', min(history\_reg.history['val\_loss']), '\nAfter', np.argmin(history\_reg.history['val\_loss']), 'epochs') |

**Output :**

****

# Practical No.3

## Aim : Implement deep learning for recognizing classes for datasets like CIFAR-10 images for previously unseen images and assign them to one of the 10 classes

**Theory :**

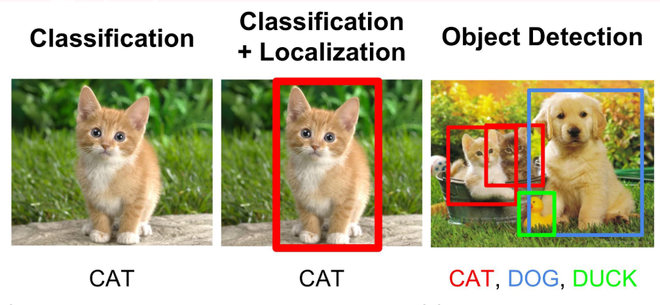
Image classification is a fundamental task in computer vision that deals with automatically understanding the content of an image. It involves assigning a category or label to an entire image based on its visual content.

**Assigning Labels:** The goal is to analyze an image and categorize it according to predefined classes. Imagine sorting photos into folders like “cats,” “dogs,” and “mountains.” Image classification automates this process using computer algorithms.

**Understanding Visual Content:** The algorithm goes beyond just recognizing shapes and colors. It extracts features from the image, like edges, textures, and patterns, to identify the objects or scene depicted.

**Training on Examples:** To achieve this, image classification models are trained on massive datasets of labeled images. These datasets help the model learn the characteristics of different categories.

Image classification is a fundamental task in computer vision that involves assigning a label or category to an image based on its visual content. Various types of image classification methods and techniques are used depending on the complexity of the task and the nature of the images.



**Program :**

|  |
| --- |
| import tensorflow as tf  from tensorflow.keras import datasets, layers, models  import matplotlib.pyplot as plt  import numpy as np |

|  |
| --- |
| (x\_train, y\_train), (x\_test, y\_test) = datasets.cifar10.load\_data()  x\_train.shape  **Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz**  **170498071/170498071 [==============================] - 3s 0us/step**  **(50000, 32, 32, 3)** |

|  |
| --- |
| x\_test.shape  **(10000, 32, 32, 3)** |

|  |
| --- |
| y\_train.shape  **(50000, 1)** |

|  |
| --- |
| y\_train[:5]  **array([[6],**  **[9],**  **[9],**  **[4],**  **[1]], dtype=uint8)** |

|  |
| --- |
| y\_train = y\_train.reshape(-1,)  classes = ["airplane", "automobile", "bird", "cat", "deer", "dog", "frog", "horse", "ship", "truck"]  fig, ax = plt.subplots(5, 5)  k = 0  for i in range(5):  for j in range(5):  ax[i][j].imshow(x\_train[k], aspect = 'auto')  k += 1  plt.show() |

|  |
| --- |
| def plot\_sample(x, y, index):  plt.figure(figsize=(15, 2))  plt.imshow(x[index])  plt.xlabel(classes[int(y[index])])  plot\_sample(x\_train, y\_train, 0) |

|  |
| --- |
| plot\_sample(x\_train, y\_train, 1) |

|  |
| --- |
| x\_train = x\_train / 255.0  x\_test = x\_test / 255.0  ann = models.Sequential([  layers.Flatten(input\_shape=(32,32,3)),  layers.Dense(3000, activation = 'relu'),  layers.Dense(1000, activation = 'relu'),  layers.Dense(10, activation = 'softmax')  ])  ann.compile(optimizer = 'SGD',  loss = 'sparse\_categorical\_crossentropy',  metrics = ['accuracy'])  ann.fit(x\_train, y\_train, epochs = 5)  **Epoch 1/5**  **1563/1563 [==============================] - 9s 5ms/step - loss: 1.8168 - accuracy: 0.3533**  **Epoch 2/5**  **1563/1563 [==============================] - 8s 5ms/step - loss: 1.6254 - accuracy: 0.4252**  **Epoch 3/5**  **1563/1563 [==============================] - 7s 5ms/step - loss: 1.5418 - accuracy: 0.4561**  **Epoch 4/5**  **1563/1563 [==============================] - 8s 5ms/step - loss: 1.4829 - accuracy: 0.4780**  **Epoch 5/5**  **1563/1563 [==============================] - 7s 5ms/step - loss: 1.4312 - accuracy: 0.4944** |

|  |
| --- |
| from sklearn.metrics import confusion\_matrix, classification\_report  import numpy as np  y\_pred = ann.predict(x\_test)  y\_pred\_classes = [np.argmax(element) for element in y\_pred]  print("Classification Report : \n", classification\_report(y\_test, y\_pred\_classes))  **313/313 [==============================] - 1s 3ms/step**  **Classification Report :**  **precision recall f1-score support**  **0 0.58 0.54 0.56 1000**  **1 0.74 0.40 0.52 1000**  **2 0.37 0.41 0.39 1000**  **3 0.35 0.38 0.36 1000**  **4 0.65 0.14 0.22 1000**  **5 0.44 0.32 0.37 1000**  **6 0.56 0.53 0.54 1000**  **7 0.32 0.82 0.46 1000**  **8 0.71 0.52 0.60 1000**  **9 0.51 0.61 0.56 1000**  **accuracy 0.47 10000**  **macro avg 0.52 0.47 0.46 10000**  **weighted avg 0.52 0.47 0.46 10000** |

|  |
| --- |
| cnn = models.Sequential([  layers.Conv2D(filters = 32, kernel\_size = (3,3), activation = 'relu', input\_shape = (32,32,3)),  layers.MaxPool2D((2,2)),  layers.Conv2D(filters = 64, kernel\_size = (3,3), activation = 'relu'),  layers.MaxPooling2D((2,2)),  layers.Flatten(),  layers.Dense(64, activation = 'relu'),  layers.Dense(10, activation = 'softmax')  ]) |

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

|  |
| --- |
| cnn.fit(x\_train, y\_train, epochs = 10)  **Epoch 1/10**  **1563/1563 [==============================] - 10s 4ms/step - loss: 1.4818 - accuracy: 0.4659**  **Epoch 2/10**  **1563/1563 [==============================] - 7s 4ms/step - loss: 1.1216 - accuracy: 0.6070**  **Epoch 3/10**  **1563/1563 [==============================] - 11s 7ms/step - loss: 0.9914 - accuracy: 0.6533**  **Epoch 4/10**  **1563/1563 [==============================] - 11s 7ms/step - loss: 0.9146 - accuracy: 0.6818**  **Epoch 5/10**  **1563/1563 [==============================] - 11s 7ms/step - loss: 0.8503 - accuracy: 0.7079**  **Epoch 6/10**  **1563/1563 [==============================] - 11s 7ms/step - loss: 0.8008 - accuracy: 0.7220**  **Epoch 7/10**  **1563/1563 [==============================] - 8s 5ms/step - loss: 0.7549 - accuracy: 0.7390**  **Epoch 8/10**  **1563/1563 [==============================] - 9s 6ms/step - loss: 0.7125 - accuracy: 0.7530**  **Epoch 9/10**  **1563/1563 [==============================] - 10s 6ms/step - loss: 0.6770 - accuracy: 0.7645**  **Epoch 10/10**  **1563/1563 [==============================] - 12s 7ms/step - loss: 0.6422 - accuracy: 0.7764** |

|  |
| --- |
| cnn.evaluate(x\_test, y\_test)  **313/313 [==============================] - 2s 4ms/step - loss: 0.8989 - accuracy: 0.6977**  **[0.8988543152809143, 0.697700023651123]** |

|  |
| --- |
| y\_pred = cnn.predict(x\_test)  y\_pred[:5]  **313/313 [==============================] - 1s 4ms/step**  **array([[3.0173849e-02, 9.8805898e-04, 1.9098191e-03, 8.1463552e-01,**  **1.3280105e-03, 1.3709904e-01, 1.2399042e-02, 4.5039957e-05,**  **8.3514058e-04, 5.8646756e-04],**  **[1.3397818e-03, 2.0462195e-03, 2.7506516e-07, 8.8259065e-07,**  **2.8701550e-07, 2.8735604e-08, 4.7879163e-09, 4.5725841e-09,**  **9.9654859e-01, 6.3897816e-05],**  **[2.2821698e-01, 1.7012353e-01, 4.0136422e-03, 1.2758560e-02,**  **7.2992020e-03, 1.2773124e-03, 5.9229264e-04, 3.0331153e-03,**  **5.4171056e-01, 3.0974824e-02],**  **[8.3504176e-01, 1.7723478e-02, 3.7949149e-02, 7.4816965e-03,**  **3.5337296e-03, 4.6612869e-04, 3.1399771e-03, 2.0054047e-04,**  **9.3784243e-02, 6.7938864e-04],**  **[8.4606953e-08, 7.4015998e-07, 5.7980260e-03, 9.2952661e-03,**  **3.4317607e-01, 4.8733331e-04, 6.4123428e-01, 2.0105613e-08,**  **8.1475282e-06, 7.9513246e-10]], dtype=float32)** |

|  |
| --- |
| y\_classes = [np.argmax(element) for element in y\_pred]  y\_classes[:5]  **[3, 8, 8, 0, 6]** |

|  |
| --- |
| y\_test[:5]  **array([[3],**  **[8],**  **[8],**  **[0],**  **[6]], dtype=uint8)** |

|  |
| --- |
| plot\_sample(x\_test, y\_test, 8)  **<ipython-input-32-17ae9f091e9d>:4: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extract a single element from your array before performing this operation. (Deprecated NumPy 1.25.)**  **plt.xlabel(classes[int(y[index])])** |

# Practical No.4

## Aim : Implement deep learning for the Prediction of the autoencoder from the test data (e.g. MNIST data set)

## Theory :

Autoencoders are a specialized class of algorithms that can learn efficient representations of input data with no need for labels. It is a class of artificial neural networks designed for unsupervised learning.

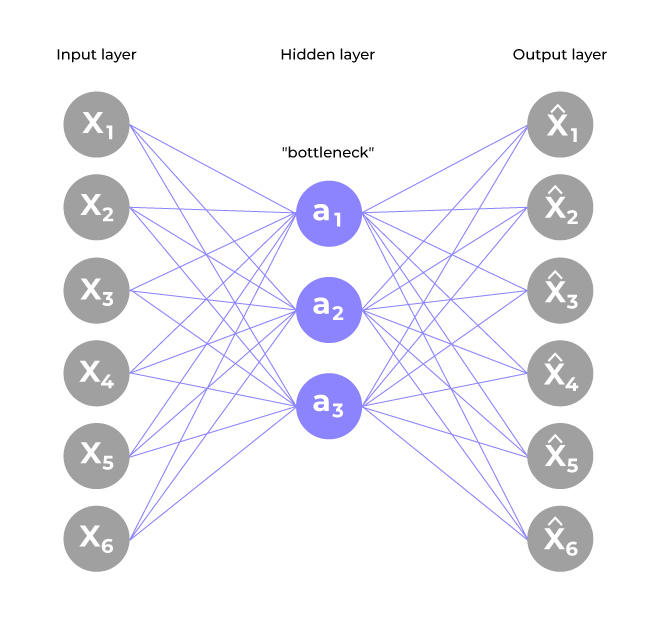
Learning to compress and effectively represent input data without specific labels is the essential principle of an automatic decoder. This is accomplished using a two-fold structure that consists of an encoder and a decoder.

**Encoder**

Input layer take raw input data.The hidden layers progressively reduce the dimensionality of the input, capturing important features and patterns. These layer compose the encoder.The bottleneck layer (latent space) is the final hidden layer, where the dimensionality is significantly reduced. This layer represents the compressed encoding of the input data.

**Decoder**

The bottleneck layer takes the encoded representation and expands it back to the dimensionality of the original input.The hidden layers progressively increase the dimensionality and aim to reconstruct the original input.The output layer produces the reconstructed output, which ideally should be as close as possible to the input data.



**Program :**

|  |
| --- |
| import keras  from keras import layers  from keras.datasets import mnist  import numpy as np  import matplotlib.pyplot as plt |

|  |
| --- |
| encoding\_dim = 32  input\_img = keras.Input(shape=(784,))  encoded = layers.Dense(encoding\_dim, activation = 'relu')(input\_img)  decoded = layers.Dense(784, activation = 'sigmoid')(encoded)  autoencoder = keras.Model(input\_img, decoded)  encoder = keras.Model(input\_img, encoded)  encoded\_input = keras.Input(shape=(encoding\_dim,))  decoder\_layer = autoencoder.layers[-1]  decoder = keras.Model(encoded\_input, decoder\_layer(encoded\_input))  autoencoder.compile(optimizer = 'adam', loss = 'binary\_crossentropy') |

|  |
| --- |
| (x\_train, \_), (x\_test, \_) =mnist.load\_data()  x\_train= x\_train.astype('float32') / 255.  x\_test = x\_test.astype('float32') / 255.  x\_train = x\_train.reshape((len(x\_train), np.prod(x\_train.shape[1:])))  x\_test = x\_test.reshape((len(x\_test), np.prod(x\_test.shape[1:])))  print(x\_train.shape)  print(x\_test.shape)  **(60000, 784)**  **(10000, 784)** |

|  |
| --- |
| autoencoder.fit(x\_train, x\_train,                  epochs = 13,                  batch\_size = 256,                  shuffle = True, validation\_data = (x\_test, x\_test))  **Epoch 1/13**  **235/235 [==============================] - 3s 5ms/step - loss: 0.2771 - val\_loss: 0.1892**  **Epoch 2/13**  **235/235 [==============================] - 1s 4ms/step - loss: 0.1704 - val\_loss: 0.1538**  **Epoch 3/13**  **235/235 [==============================] - 1s 4ms/step - loss: 0.1446 - val\_loss: 0.1342**  **Epoch 4/13**  **235/235 [==============================] - 1s 4ms/step - loss: 0.1293 - val\_loss: 0.1222**  **Epoch 5/13**  **235/235 [==============================] - 1s 5ms/step - loss: 0.1191 - val\_loss: 0.1134**  **Epoch 6/13**  **235/235 [==============================] - 1s 6ms/step - loss: 0.1116 - val\_loss: 0.1072**  **Epoch 7/13**  **235/235 [==============================] - 2s 7ms/step - loss: 0.1062 - val\_loss: 0.1026**  **Epoch 8/13**  **235/235 [==============================] - 2s 6ms/step - loss: 0.1024 - val\_loss: 0.0995**  **Epoch 9/13**  **235/235 [==============================] - 1s 4ms/step - loss: 0.0996 - val\_loss: 0.0971**  **Epoch 10/13**  **235/235 [==============================] - 1s 4ms/step - loss: 0.0975 - val\_loss: 0.0955**  **Epoch 11/13**  **235/235 [==============================] - 1s 4ms/step - loss: 0.0961 - val\_loss: 0.0945**  **Epoch 12/13**  **235/235 [==============================] - 1s 4ms/step - loss: 0.0953 - val\_loss: 0.0937**  **Epoch 13/13**  **235/235 [==============================] - 1s 4ms/step - loss: 0.0947 - val\_loss: 0.0932** |

|  |
| --- |
| encoded\_imgs = encoder.predict(x\_test)  decoded\_imgs = decoder.predict(encoded\_imgs)  **313/313 [==============================] - 0s 1ms/step**  **313/313 [==============================] - 0s 1ms/step** |

|  |
| --- |
| n = 10  plt.figure(figsize = (20,4))  for i in range(n):    ax = plt.subplot(2, n, i+1)    plt.imshow(x\_test[i].reshape(28,28))    plt.gray()    ax.get\_xaxis().set\_visible(False)    ax.get\_yaxis().set\_visible(False)    ax = plt.subplot(2, n, i+1+n)    plt.imshow(decoded\_imgs[i].reshape(28, 28))    plt.gray()    ax.get\_xaxis().set\_visible(False)    ax.get\_yaxis().set\_visible(False)  plt.show() |

# Practical No.5

## Aim : Implement Convolutional Neural Network for Digit Recognition on the MNIST Dataset

## Theory :

## Handwritten digit recognition using MNIST dataset is a major project made with the help of Neural Network. It basically detects the scanned images of handwritten digits.

## We have taken this a step further where our handwritten digit recognition system not only detects scanned images of handwritten digits but also allows writing digits on the screen with the help of an integrated GUI for recognition.

## The input layer: It distributes the features of our examples to the next layer for calculation of activations of the next layer.

## The hidden layer: They are made of hidden units called activations providing nonlinear ties for the network. A number of hidden layers can vary according to our requirements.

## The output layer: The nodes here are called output units. It provides us with the final prediction of the Neural Network on the basis of which final predictions can be made.

## A neural network is a model inspired by how the brain works. It consists of multiple layers having many activations, this activation resembles neurons of our brain. A neural network tries to learn a set of parameters in a set of data which could help to recognize the underlying relationships. Neural networks can adapt to changing input; so the network generates the best possible result without needing to redesign the output criteria.

## 

## Convolutional Neural Network(CNN)

## Program :

|  |
| --- |
| import numpy as npfrom matplotlib import pyplot as pltfrom keras.utils import to\_categoricalfrom keras.models import Sequentialfrom keras.layers import Dense, Dropout, Flattenfrom keras.layers import Conv2Dfrom keras.callbacks import EarlyStoppingfrom keras.datasets import mnist |

|  |
| --- |
| (x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz11490434/11490434 [==============================] - 2s 0us/step |

|  |
| --- |
| img\_rows, img\_cols = x\_train[0].shape[0], x\_train[0].shape[1]x\_train = x\_train.reshape(x\_train.shape[0], img\_rows, img\_cols, 1)x\_test = x\_test.reshape(x\_test.shape[0], img\_rows, img\_cols, 1)input\_shape = (img\_rows, img\_cols, 1) |

|  |
| --- |
| x\_train = x\_train.astype('float32') / 255.x\_test = x\_test.astype('float32') / 255. |

|  |
| --- |
| n\_classes = len(set(y\_train))y\_train = to\_categorical(y\_train, n\_classes)y\_test = to\_categorical(y\_test, n\_classes) |

|  |
| --- |
| model = Sequential()model.add(Conv2D(64, kernel\_size = (3,3), activation = 'relu', input\_shape = input\_shape))model.add(Conv2D(128, kernel\_size = (3,3), activation = 'relu'))model.add(Conv2D(256, kernel\_size = (3,3), activation = 'relu'))model.add(Dropout(0.5))model.add(Flatten())model.add(Dense(128, activation = 'relu'))model.add(Dropout(0.5))model.add(Dense(n\_classes, activation = 'softmax'))model.compile(loss = 'categorical\_crossentropy', optimizer = 'adam', metrics = ['accuracy']) |

|  |
| --- |
| callbacks = [EarlyStopping(monitor = 'val\_acc', patience = 5)] |

|  |
| --- |
| batch\_size = 128n\_epochs = 100 |

|  |
| --- |
| model.fit(x\_train, y\_train, batch\_size = batch\_size, epochs = n\_epochs, verbose = 1, validation\_split = 0.2, callbacks = callbacks)Epoch 1/100375/375 [==============================] - ETA: 0s - loss: 0.1952 - accuracy: 0.9404WARNING:tensorflow:Early stopping conditioned on metric `val\_acc` which is not available. Available metrics are: loss,accuracy,val\_loss,val\_accuracy375/375 [==============================] - 25s 44ms/step - loss: 0.1952 - accuracy: 0.9404 - val\_loss: 0.0565 - val\_accuracy: 0.9833Epoch 2/100375/375 [==============================] - ETA: 0s - loss: 0.0716 - accuracy: 0.9794WARNING:tensorflow:Early stopping conditioned on metric `val\_acc` which is not available. Available metrics are: loss,accuracy,val\_loss,val\_accuracy375/375 [==============================] - 15s 40ms/step - loss: 0.0716 - accuracy: 0.9794 - val\_loss: 0.0385 - val\_accuracy: 0.9895Epoch 3/100374/375 [============================>.] - ETA: 0s - loss: 0.0538 - accuracy: 0.9841WARNING:tensorflow:Early stopping conditioned on metric `val\_acc` which is not available. Available metrics are: loss,accuracy,val\_loss,val\_accuracy375/375 [==============================] - 15s 39ms/step - loss: 0.0537 - accuracy: 0.9842 - val\_loss: 0.0396 - val\_accuracy: 0.9890Epoch 4/100375/375 [==============================] - ETA: 0s - loss: 0.0428 - accuracy: 0.9866WARNING:tensorflow:Early stopping conditioned on metric `val\_acc` which is not available. Available metrics are: loss,accuracy,val\_loss,val\_accuracy375/375 [==============================] - 15s 39ms/step - loss: 0.0428 - accuracy: 0.9866 - val\_loss: 0.0410 - val\_accuracy: 0.9898Epoch 5/100375/375 [==============================] - ETA: 0s - loss: 0.0339 - accuracy: 0.9897WARNING:tensorflow:Early stopping conditioned on metric `val\_acc` which is not available. Available metrics are: loss,accuracy,val\_loss,val\_accuracy375/375 [==============================] - 15s 41ms/step - loss: 0.0339 - accuracy: 0.9897 - val\_loss: 0.0456 - val\_accuracy: 0.9882375/375 [==============================] - 15s 41ms/step - loss: 0.0039 - accuracy: 0.9989 - val\_loss: 0.1029 - val\_accuracy: 0.9914Epoch 96/100375/375 [==============================] - ETA: 0s - loss: 0.0042 - accuracy: 0.9989WARNING:tensorflow:Early stopping conditioned on metric `val\_acc` which is not available. Available metrics are: loss,accuracy,val\_loss,val\_accuracy375/375 [==============================] - 15s 40ms/step - loss: 0.0042 - accuracy: 0.9989 - val\_loss: 0.0921 - val\_accuracy: 0.9918Epoch 97/100375/375 [==============================] - ETA: 0s - loss: 0.0053 - accuracy: 0.9987WARNING:tensorflow:Early stopping conditioned on metric `val\_acc` which is not available. Available metrics are: loss,accuracy,val\_loss,val\_accuracy375/375 [==============================] - 15s 41ms/step - loss: 0.0053 - accuracy: 0.9987 - val\_loss: 0.1137 - val\_accuracy: 0.9923Epoch 98/100375/375 [==============================] - ETA: 0s - loss: 0.0054 - accuracy: 0.9986WARNING:tensorflow:Early stopping conditioned on metric `val\_acc` which is not available. Available metrics are: loss,accuracy,val\_loss,val\_accuracy375/375 [==============================] - 15s 40ms/step - loss: 0.0054 - accuracy: 0.9986 - val\_loss: 0.1157 - val\_accuracy: 0.9917Epoch 99/100375/375 [==============================] - ETA: 0s - loss: 0.0047 - accuracy: 0.9989WARNING:tensorflow:Early stopping conditioned on metric `val\_acc` which is not available. Available metrics are: loss,accuracy,val\_loss,val\_accuracy375/375 [==============================] - 15s 40ms/step - loss: 0.0047 - accuracy: 0.9989 - val\_loss: 0.1100 - val\_accuracy: 0.9922Epoch 100/100375/375 [==============================] - ETA: 0s - loss: 0.0045 - accuracy: 0.9990WARNING:tensorflow:Early stopping conditioned on metric `val\_acc` which is not available. Available metrics are: loss,accuracy,val\_loss,val\_accuracy375/375 [==============================] - 15s 40ms/step - loss: 0.0045 - accuracy: 0.9990 - val\_loss: 0.0865 - val\_accuracy: 0.9928 |

|  |
| --- |
| score = model.evaluate(x\_test, y\_test, verbose = 0)print('Test Loss : ', score[0])print('Test Accuracy : ', score[1])preds = model.predict(x\_test)n\_examples = 10plt.figure(figsize=(15,15))for i in range(n\_examples):ax = plt.subplot(2, n\_examples, i+1)plt.imshow(x\_test[i, :, :, 0], cmap = 'gray')plt.title("Label : {}\nPredicted : {}".format(np.argmax(y\_test[i]), np.argmax(preds[i]))plt.axis('off')Test Loss : 0.06791581958532333Test Accuracy : 0.9923999905586243313/313 [==============================] - 1s 3ms/step |

|  |
| --- |
| plt.figure(figsize=(15,15))j=1for i in range(len(y\_test)):if(j>10):breaklabel = np.argmax(y\_test[i])pred = np.argmax(preds[i])if label != pred:ax = plt.subplot(2, n\_examples, j)plt.imshow(x\_test[i, :, :, 0], cmap = 'gray')plt.title("Label : {}\nPredicted : {}".format(label, pred))j += 1plt.show() |

# Practical No.6

## Aim : Write a program to implement Transfer Learning on the suitable dataset (e.g. classify the cats versus dogs dataset from Kaggle).

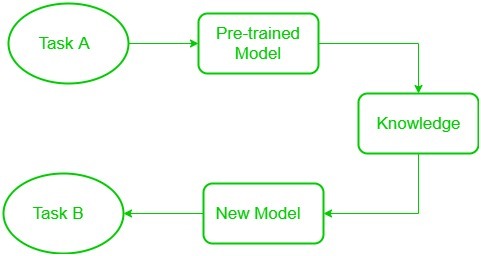
## Theory :

## Transfer learning is a smart method in machine learning where a model uses knowledge from one task to help with a different, but related, task. Instead of learning from zero, the model uses what it already knows to solve new problems faster and better.

## Transfer learning is making a big impact in areas like understanding language and recognizing images

## Transfer learning is a technique in machine learning where a model trained on one task is used as the starting point for a model on a second task. This can be useful when the second task is similar to the first task, or when there is limited data available for the second task.

## By using the learned features from the first task as a starting point, the model can learn more quickly and effectively on the second task. This can also help to prevent overfitting, as the model will have already learned general features that are likely to be useful in the second task.



**Transfer Learning**

**Programs :**

|  |
| --- |
| !pip install Kaggle  **Requirement already satisfied: kaggle in /usr/local/lib/python3.10/dist-packages (1.6.14)**  **Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.10/dist-packages (from kaggle) (1.16.0)**  **Requirement already satisfied: certifi>=2023.7.22 in /usr/local/lib/python3.10/dist-packages (from kaggle) (2024.2.2)**  **Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.8.2)**  **Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.31.0)**  **Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from kaggle) (4.66.4)**  **Requirement already satisfied: python-slugify in /usr/local/lib/python3.10/dist-packages (from kaggle) (8.0.4)**  **Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.0.7)**  **Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from kaggle) (6.1.0)**  **Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->kaggle) (0.5.1)**  **Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.10/dist-packages (from python-slugify->kaggle) (1.3)**  **Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.3.2)**  **Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.7)** |

|  |
| --- |
| !mkdir -p ~/.kaggle  !cp kaggle.json ~/.kaggle/  !chmod 600 ~/.kaggle/kaggle.json |

|  |
| --- |
| !kaggle competitions download -c dogs-vs-cats  **Downloading dogs-vs-cats.zip to /content**  **100% 810M/812M [00:36<00:00, 24.3MB/s]**  **100% 812M/812M [00:36<00:00, 23.4MB/s]** |

|  |
| --- |
| from zipfile import ZipFile  dataset = "/content/dogs-vs-cats.zip"  with ZipFile(dataset, 'r') as zip:  zip.extractall()  print("The Dataset is Extracted")  **The Dataset is Extracted** |

|  |
| --- |
| from zipfile import ZipFile  dataset = "/content/train.zip"  with ZipFile(dataset, 'r') as zip:  zip.extractall()  print("The Dataset is Extracted")  **The Dataset is Extracted** |

|  |
| --- |
| import os  path, dirs, files = next(os.walk("/content/train"))  file\_count = len(files)  print("Number of Images : ", file\_count)  **Number of Images : 25000** |

|  |
| --- |
| file\_names = os.listdir("/content/train")  print(file\_names) |

|  |
| --- |
| import numpy as np  from PIL import Image  import matplotlib.pyplot as plt  import matplotlib.image as mpimg  from sklearn.model\_selection import train\_test\_split  from google.colab.patches import cv2\_imshow |

|  |
| --- |
| img = mpimg.imread("/content/train/dog.8949.jpg")  imgplt = plt.imshow(img)  plt.show() |

|  |
| --- |
| img = mpimg.imread("/content/train/cat.10002.jpg")  imgplt = plt.imshow(img)  plt.show() |

|  |
| --- |
| file\_names = os.listdir("/content/train")  for i in range(5):  name = file\_names[i]  print(name[0:3])  **dog**  **dog**  **cat**  **cat**  **cat** |

|  |
| --- |
| file\_names = os.listdir("/content/train")  dog\_count = 0  cat\_count = 0 |

|  |
| --- |
| for img\_file in file\_names:  name = img\_file[0:3]  if name == "dog":  dog\_count += 1  else:  cat\_count += 1  print("Number of Dog Images : ", dog\_count)  print("Number of Cat Images : ", cat\_count)  **Number of Dog Images : 12500**  **Number of Cat Images : 12500** |

|  |
| --- |
| os.mkdir("/content/images\_resized") |

|  |
| --- |
| original\_folder = "/content/train/"  resized\_folder = "/content/images\_resized/"  for i in range(2000):  filename = os.listdir(original\_folder)[i]  img\_path = original\_folder + filename  img = Image.open(img\_path)  img = img.resize((224, 224))  img = img.convert('RGB')  newImgPath = resized\_folder + filename  img.save(newImgPath) |

|  |
| --- |
| img = mpimg.imread("/content/images\_resized/cat.10031.jpg")  imgplt = plt.imshow(img)  plt.show() |

|  |
| --- |
| filenames = os.listdir("/content/images\_resized")  labels = []  for i in range(2000):  file\_name = filenames[i]  label = file\_name[0:3]  if label == "dog":  labels.append(1)  else:  labels.append(0) |

|  |
| --- |
| print(filenames[0:5])  print(len(filenames))  **['dog.10160.jpg', 'dog.6063.jpg', 'cat.5313.jpg', 'cat.9171.jpg', 'cat.9108.jpg']**  **2000** |

|  |
| --- |
| print(labels[0:5])  print(len(labels))  **[1, 1, 0, 0, 0]**  **2000** |

|  |
| --- |
| values, counts = np.unique(labels, return\_counts=True)  print(values)  print(counts)  **[0 1]**  **[1012 988]** |

|  |
| --- |
| import cv2  import glob  image\_directory = "/content/images\_resized/"  image\_extension = ["png", "jpg"]  files = []  [files.extend(glob.glob(image\_directory + '\*.' + e)) for e in image\_extension]  dog\_cat\_images = np.asarray([cv2.imread(file) for file in files])  print(dog\_cat\_images) |

|  |
| --- |
| type(dog\_cat\_images)  **numpy.ndarray** |

|  |
| --- |
| print(dog\_cat\_images.shape)  **(2000, 224, 224, 3)** |

|  |
| --- |
| x = dog\_cat\_images  y = np.asarray(labels)  x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, random\_state= 2)  print(x.shape, x\_train.shape, x\_test.shape)  **(2000, 224, 224, 3) (1600, 224, 224, 3) (400, 224, 224, 3)** |

|  |
| --- |
| x\_train\_scaled = x\_train/255  x\_test\_scaled = x\_test/255 |

|  |
| --- |
| import tensorflow as tf  import tensorflow\_hub as hub  mobilenet\_model = "https://tfhub.dev/google/tf2-preview/mobilenet\_v2/feature\_vector/4"  pretrained\_model = hub.KerasLayer(mobilenet\_model, input\_shape = (224, 224, 3), trainable = False)  num\_of\_classes = 2  model = tf.keras.Sequential([  pretrained\_model,  tf.keras.layers.Dense(num\_of\_classes)  ])  model.summary()  **Model: "sequential"**  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Layer (type) Output Shape Param #**  **=================================================================**  **keras\_layer (KerasLayer) (None, 1280) 2257984**    **dense (Dense) (None, 2) 2562**    **=================================================================**  **Total params: 2260546 (8.62 MB)**  **Trainable params: 2562 (10.01 KB)**  **Non-trainable params: 2257984 (8.61 MB)**  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** |

|  |
| --- |
| model.compile(  optimizer = "adam",  loss = tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True),  metrics = ["acc"]) |

|  |
| --- |
| model.fit(x\_train\_scaled, y\_train, epochs = 5)  **Epoch 1/5**  **50/50** **[==============================] - 10s 48ms/step - loss: 0.1864 - acc: 0.9244**  **Epoch 2/5**  **50/50 [==============================] - 2s 41ms/step - loss: 0.0669 - acc: 0.9806**  **Epoch 3/5**  **50/50 [==============================] - 2s 41ms/step - loss: 0.0464 - acc: 0.9875**  **Epoch 4/5**  **50/50 [==============================] - 3s 54ms/step - loss: 0.0359 - acc: 0.9919**  **Epoch 5/5**  **50/50 [==============================] - 2s 47ms/step - loss: 0.0284 - acc: 0.9925** |

|  |
| --- |
| score, acc = model.evaluate(x\_test\_scaled, y\_test)  print("Test Loss : ", score)  print("Test Accuracy : ", acc)  **13/13 [==============================] - 2s 105ms/step - loss: 0.0782 - acc: 0.9700**  **Test Loss : 0.0782330334186554**  **Test Accuracy : 0.9700000286102295** |

|  |
| --- |
| input\_image\_path = input("Path of the Image to be Predicted : ")  input\_image = cv2.imread(input\_image\_path)  cv2\_imshow(input\_image)  input\_image\_resize = cv2.resize(input\_image, (224, 224))  input\_image\_scaled = input\_image\_resize/255  image\_reshape = np.reshape(input\_image\_scaled, [1, 224, 224, 3])  input\_prediction = model.predict(image\_reshape)  print(input\_prediction)  input\_pred\_label = np.argmax(input\_prediction)  print(input\_pred\_label)  if input\_pred\_label == 0:  print("The Image represents a Cat")  else:  print("The Image represents a Dog")  **Path of the Image to be Predicted : /content/Test image.jpg**    **1/1 [==============================] - 0s 25ms/step**  **[[ 4.781723 -3.400223]]**  **0**  **The Image represents a Cat** |

# Practical No.7

## Aim : Write a program for the Implementation of a Generative Adversarial Network for generating synthetic shapes (like digits)

**Theory :**

Generative Adversarial Networks (GANs) are a powerful class of neural networks that are used for an unsupervised learning. GANs are made up of two neural networks, a discriminator and a generator. They use adversarial training to produce artificial data that is identical to actual data.

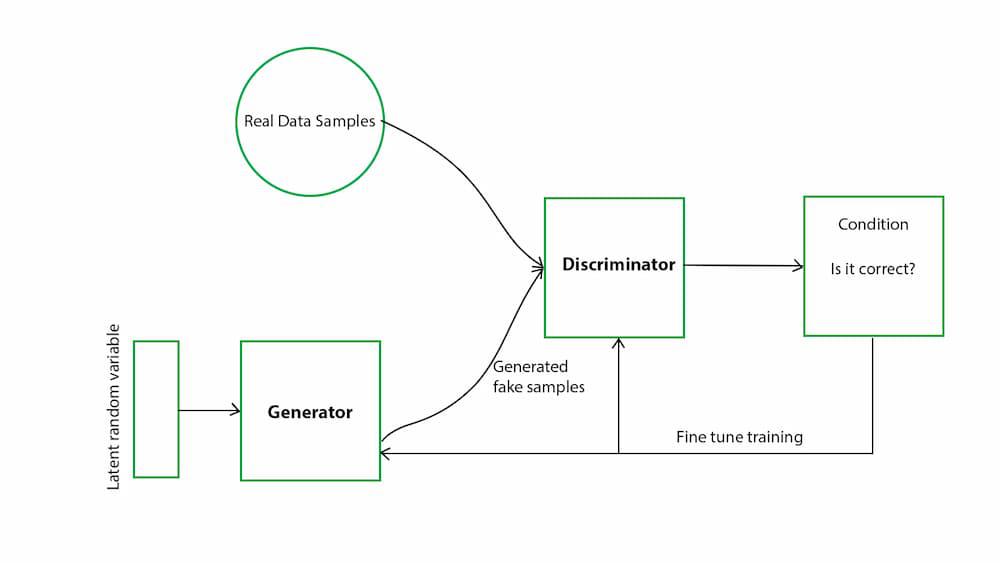
The Generator attempts to fool the Discriminator, which is tasked with accurately distinguishing between produced and genuine data, by producing random noise samples.

Realistic, high-quality samples are produced as a result of this competitive interaction, which drives both networks toward advancement.

GANs are proving to be highly versatile artificial intelligence tools, as evidenced by their extensive use in image synthesis, style transfer, and text-to-image synthesis.

They have also revolutionized generative modeling.

Through adversarial training, these models engage in a competitive interplay until the generator becomes adept at creating realistic samples, fooling the discriminator approximately half the time.

****

**Generative Adversarial Network**

**Program :**

|  |
| --- |
| import numpy as np  from keras.models import Sequential, Model  from keras.layers import Input, Dense, Activation, Flatten, Reshape  from keras.layers import Conv2D, Conv2DTranspose, UpSampling2D  from keras.layers import LeakyReLU, Dropout  from keras.layers import BatchNormalization  from keras.optimizers import Adam  from keras import initializers  from keras.datasets import mnist  import matplotlib.pyplot as plt |

|  |
| --- |
| (x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()  img\_rows, img\_cols = x\_train.shape[1:]  x\_train = (x\_train.reshape(-1, img\_rows\*img\_cols, 1).astype(np.float32) - 127.5) / 127.5 |

|  |
| --- |
| def discriminator\_model(dropout=0.5):    model = Sequential()    model.add(Dense(1024, input\_dim = 784, kernel\_initializer = initializers.RandomNormal(stddev=0.02)))    model.add(LeakyReLU(0.2))    model.add(Dropout(dropout))    model.add(Dense(512))    model.add(LeakyReLU(0.2))    model.add(Dropout(dropout))    model.add(Dense(256))    model.add(LeakyReLU(0.2))    model.add(Dropout(dropout))    model.add(Dense(1, activation = 'sigmoid'))    opt = Adam(learning\_rate = 0.0001)    model.compile(loss = 'binary\_crossentropy', optimizer = opt)    return model |

|  |
| --- |
| def generator\_model():    model = Sequential()    model.add(Dense(256, input\_dim = 100, kernel\_initializer = initializers.RandomNormal(stddev = 0.02)))    model.add(LeakyReLU(0.2))    model.add(Dense(512))    model.add(LeakyReLU(0.2))    model.add(Dense(1024))    model.add(LeakyReLU(0.2))    model.add(Dense(784, activation = 'tanh'))    opt = Adam(learning\_rate = 0.00005)    model.compile(loss = 'binary\_crossentropy', optimizer = opt)    return model |

|  |
| --- |
| discriminator = discriminator\_model()  generator = generator\_model()  discriminator.trainable = False  gan\_input = Input(shape=(100,))  x = generator(gan\_input)  gan\_output = discriminator(x)  gan = Model(inputs = gan\_input, outputs = gan\_output)  opt = Adam(learning\_rate = 0.0001)  gan.compile(loss = 'binary\_crossentropy', optimizer = opt) |

|  |
| --- |
| def plot\_images(samples = 16, step = 0):    plt.figure(figsize = (5,5))    for i in range(samples):      noise = np.random.uniform(-1, 1, size = [batch\_size, 100])      images = generator.predict(noise)      plt.subplot(4, 4, i+1)      image = images[i, :,]      image = np.reshape(image, [img\_rows, img\_cols])      plt.imshow(image, cmap = 'gray')      plt.axis('off')    plt.show() |

|  |
| --- |
| batch\_size = 500  n\_steps = 100000  plot\_every = 100 |

|  |
| --- |
| noise\_input = np.random.uniform(-1, 1, size = [16, 100])  for step in range(n\_steps):    noise = np.random.uniform(-1, 1, size = [batch\_size, 100])    batch = x\_train[np.random.randint(0, x\_train.shape[0], size = batch\_size)].reshape(batch\_size, 784)    gen\_output = generator.predict(noise)    x = np.concatenate([batch, gen\_output])    y\_D = np.zeros(2\*batch\_size)    y\_D[:batch\_size] = 1    discriminator.trainable = True    loss\_D = discriminator.train\_on\_batch(x, y\_D)    noise = np.random.uniform(-1, 1, size = [batch\_size, 100])    y\_G = np.ones(batch\_size)    discriminator.trainable = False    loss\_G = gan.train\_on\_batch(noise, y\_G)    if step % plot\_every == 0:      print('Step {} '.format(step))      plot\_images(samples = noise\_input.shape[0], step = (step+1)) |

# Practical No.8

## Aim : Write a program to implement a simple form of a recurrent neural network.

## E.g. (4-to-1 RNN) to show that the quantity of rain on a certain day also depends on the values of the previous day

## Theory :

## Recurrent Neural Network(RNN) is a type of [Neural Network](https://www.geeksforgeeks.org/tag/neural-network) where the output from the previous step is fed as input to the current step. In traditional neural networks, all the inputs and outputs are independent of each other.

## Still, in cases when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words. Thus RNN came into existence, which solved this issue with the help of a Hidden Layer.

## The main and most important feature of RNN is its ****Hidden state****, which remembers some information about a sequence. The state is also referred to as Memory State since it remembers the previous input to the network.

## It uses the same parameters for each input as it performs the same task on all the inputs or hidden layers to produce the output. This reduces the complexity of parameters, unlike other neural networks.

## 

## Recurrent Neural Network

## Program :

|  |
| --- |
| import numpy as npx = [[1, 0, 0, 0],[0, 1, 0, 0],[0, 0, 1, 0],[0, 0, 0, 1],[0, 0, 0, 1],[1, 0, 0, 0],[0, 1, 0, 0],[0, 0, 1, 0],[0, 0, 0, 1]]y = [0.20, 0.30, 0.40, 0.50, 0.05, 0.10, 0.20, 0.30, 0.40] |

|  |
| --- |
| def sigmoid(x):return 1 / (1 + np.exp(-x))def sigmoid\_der(x):return x \* (1 - x) |

|  |
| --- |
| n\_units = [4, 16, 1]n\_layers = len(n\_units)layers = [np.ones(n\_units[i]) for i in range(n\_layers)]weights = [np.random.randn(n\_units[i], n\_units[i + 1]) for i in range(n\_layers - 1)]weights\_delta = [np.zeros\_like(weights[i]) for i in range(len(weights))] |

|  |
| --- |
| def forward(data):layers[0] = datafor i in range(1, n\_layers):layers[i] = sigmoid(np.dot(layers[i - 1], weights[i - 1]))return layers[-1] |

|  |
| --- |
| def backwards(target, learning\_rate=0.1, momentum=0.1):error = target - layers[-1]deltas = [error \* sigmoid\_der(layers[-1])] |

|  |
| --- |
| for i in range(n\_layers - 2, 0, -1):delta = np.dot(deltas[0], weights[i].T) \* sigmoid\_der(layers[i])deltas.insert(0, delta)for i in range(len(weights)):layer = np.atleast\_2d(layers[i])delta = np.atleast\_2d(deltas[i])weights\_delta\_temp = np.dot(layer.T, delta)weights[i] += learning\_rate \* weights\_delta\_temp + momentum \* weights\_delta[i]weights\_delta[i] = weights\_delta\_tempreturn (error \*\* 2).sum() |

|  |
| --- |
| n\_epochs = 1000for epoch in range(n\_epochs):loss = 0for j in range(len(x)):forward(x[j])loss += backwards(y[j])if epoch % 100 == 0:print('epoch {} - loss: {:04.4f}'.format(epoch, loss))epoch 0 - loss: 4.3115epoch 100 - loss: 0.1351epoch 200 - loss: 0.1321epoch 300 - loss: 0.1318epoch 400 - loss: 0.1317epoch 500 - loss: 0.1317epoch 600 - loss: 0.1317epoch 700 - loss: 0.1316epoch 800 - loss: 0.1316epoch 900 - loss: 0.1316 |

|  |
| --- |
| for i in range(len(x)):pred = forward(x[i])loss = (y[i] - pred)\*\*2print("x : {}; y : {:04.2f}; pred : {:04.2f}".format(x[i], y[i], pred[0]))x : [1, 0, 0, 0]; y : 0.20; pred : 0.15x : [0, 1, 0, 0]; y : 0.30; pred : 0.25x : [0, 0, 1, 0]; y : 0.40; pred : 0.35x : [0, 0, 0, 1]; y : 0.50; pred : 0.31x : [0, 0, 0, 1]; y : 0.05; pred : 0.31x : [1, 0, 0, 0]; y : 0.10; pred : 0.15x : [0, 1, 0, 0]; y : 0.20; pred : 0.25x : [0, 0, 1, 0]; y : 0.30; pred : 0.35x : [0, 0, 0, 1]; y : 0.40; pred : 0.31 |

## (ii) LSTM for sentiment analysis on datasets like UMICH SI650 for similar.

## Theory :

## ****Introduction to LSTM****

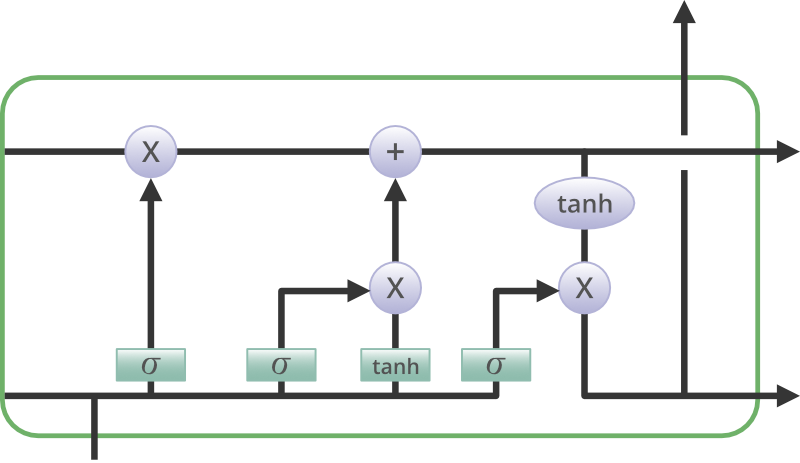
LSTM networks are an extension of recurrent neural networks ([RNNs](https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/)) mainly introduced to handle situations where RNNs fail.

* It fails to store information for a longer period of time. At times, a reference to certain information stored quite a long time ago is required to predict the current output. But RNNs are absolutely incapable of handling such “long-term dependencies”.
* There is no finer control over which part of the context needs to be carried forward and how much of the past needs to be ‘forgotten’.
* Other issues with RNNs are exploding and vanishing gradients (explained later) which occur during the training process of a network through backtracking.

Thus, Long Short-Term Memory ([LSTM](https://www.geeksforgeeks.org/long-short-term-memory-networks-explanation/)) was brought into the picture. It has been so designed that the vanishing gradient problem is almost completely removed, while the training model is left unaltered.

Long-time lags in certain problems are bridged using LSTMs which also handle noise, distributed representations, and continuous values. With LSTMs, there is no need to keep a finite number of states from beforehand as required in the hidden [Markov model](https://www.geeksforgeeks.org/hidden-markov-model-in-machine-learning/) (HMM).

LSTMs provide us with a large range of parameters such as learning rates, and input and output biases.



## Structure of LSTM Network

**Program :**

|  |
| --- |
| !pip install keras |

|  |
| --- |
| import numpy as np  from keras.layers import Dense, Dropout, Activation  from keras.layers import Embedding  from keras.layers import LSTM  from keras.callbacks import EarlyStopping  import keras  import keras.utils  from keras.datasets import imdb |

|  |
| --- |
| n\_words = 1000  (x\_train, y\_train), (x\_test, y\_test) = imdb.load\_data(num\_words = n\_words)  print("Train seq: {}".format(len(x\_train)))  print("Test seq: {}".format(len(x\_train)))  **Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz**  **17464789/17464789 [==============================] - 1s 0us/step**  **Train seq: 25000**  **Test seq: 25000** |

|  |
| --- |
| print("Train example : \n{}".format(x\_train[0]))  print("\nTest example : \n{}".format(x\_test[0]))  **Train example :**  **[1, 14, 22, 16, 43, 530, 973, 2, 2, 65, 458, 2, 66, 2, 4, 173, 36, 256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 172, 2, 2, 17, 546, 38, 13, 447, 4, 192, 50, 16, 6, 147, 2, 19, 14, 22, 4, 2, 2, 469, 4, 22, 71, 87, 12, 16, 43, 530, 38, 76, 15, 13, 2, 4, 22, 17, 515, 17, 12, 16, 626, 18, 2, 5, 62, 386, 12, 8, 316, 8, 106, 5, 4, 2, 2, 16, 480, 66, 2, 33, 4, 130, 12, 16, 38, 619, 5, 25, 124, 51, 36, 135, 48, 25, 2, 33, 6, 22, 12, 215, 28, 77, 52, 5, 14, 407, 16, 82, 2, 8, 4, 107, 117, 2, 15, 256, 4, 2, 7, 2, 5, 723, 36, 71, 43, 530, 476, 26, 400, 317, 46, 7, 4, 2, 2, 13, 104, 88, 4, 381, 15, 297, 98, 32, 2, 56, 26, 141, 6, 194, 2, 18, 4, 226, 22, 21, 134, 476, 26, 480, 5, 144, 30, 2, 18, 51, 36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38, 2, 88, 12, 16, 283, 5, 16, 2, 113, 103, 32, 15, 16, 2, 19, 178, 32]**  **Test example :**  **[1, 591, 202, 14, 31, 6, 717, 10, 10, 2, 2, 5, 4, 360, 7, 4, 177, 2, 394, 354, 4, 123, 9, 2, 2, 2, 10, 10, 13, 92, 124, 89, 488, 2, 100, 28, 2, 14, 31, 23, 27, 2, 29, 220, 468, 8, 124, 14, 286, 170, 8, 157, 46, 5, 27, 239, 16, 179, 2, 38, 32, 25, 2, 451, 202, 14, 6, 717]** |

|  |
| --- |
| from keras.preprocessing import sequence  max\_len = 200  x\_train = sequence.pad\_sequences(x\_train, maxlen = max\_len)  x\_test = sequence.pad\_sequences(x\_test, maxlen = max\_len) |

|  |
| --- |
| from keras.models import Sequential  model = Sequential()  model.add(Embedding(n\_words, 50, input\_length = max\_len))  model.add(Dropout(0.2))  model.add(LSTM(100, dropout = 0.2, recurrent\_dropout = 0.2))  model.add(Dense(250, activation = 'relu'))  model.add(Dropout(0.2))  model.add(Dense(1, activation = 'sigmoid'))  model.compile(loss = 'binary\_crossentropy', optimizer = 'adam', metrics = ['accuracy'])  model.summary()  **WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.**  **Model: "sequential"**  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Layer (type) Output Shape Param #**  **=================================================================**  **embedding (Embedding) (None, 200, 50) 50000**    **dropout (Dropout) (None, 200, 50) 0**    **lstm (LSTM) (None, 100) 60400**    **dense (Dense) (None, 250) 25250**    **dropout\_1 (Dropout) (None, 250) 0**    **dense\_1 (Dense) (None, 1) 251**    **=================================================================**  **Total params: 135901 (530.86 KB)**  **Trainable params: 135901 (530.86 KB)**  **Non-trainable params: 0 (0.00 Byte)**  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** |

|  |
| --- |
| callbacks = [EarlyStopping(monitor = 'val\_acc', patience = 3)] |

|  |
| --- |
| batch\_size = 128  n\_epochs = 10  history = model.fit(x\_train, y\_train, batch\_size, epochs = n\_epochs, validation\_split = 0.2, callbacks = callbacks)  **Epoch 1/10**  **157/157 [==============================] - ETA: 0s - loss: 0.5383 - accuracy: 0.7120WARNING:tensorflow:Early stopping conditioned on metric `val\_acc` which is not available. Available metrics are: loss,accuracy,val\_loss,val\_accuracy**  **157/157 [==============================] - 125s 707ms/step - loss: 0.5383 - accuracy: 0.7120 - val\_loss: 0.3997 - val\_accuracy: 0.8260**  **Epoch 2/10**  **157/157 [==============================] - ETA: 0s - loss: 0.3611 - accuracy: 0.8440WARNING:tensorflow:Early stopping conditioned on metric `val\_acc` which is not available. Available metrics are: loss,accuracy,val\_loss,val\_accuracy**  **157/157 [==============================] - 102s 646ms/step - loss: 0.3611 - accuracy: 0.8440 - val\_loss: 0.3604 - val\_accuracy: 0.8456**  **Epoch 3/10**  **157/157 [==============================] - ETA: 0s - loss: 0.3364 - accuracy: 0.8554WARNING:tensorflow:Early stopping conditioned on metric `val\_acc` which is not available. Available metrics are: loss,accuracy,val\_loss,val\_accuracy**  **157/157 [==============================] - 101s 641ms/step - loss: 0.3364 - accuracy: 0.8554 - val\_loss: 0.3454 - val\_accuracy: 0.8528**  **Epoch 4/10**  **157/157 [==============================] - ETA: 0s - loss: 0.3256 - accuracy: 0.8633WARNING:tensorflow:Early stopping conditioned on metric `val\_acc` which is not available. Available metrics are: loss,accuracy,val\_loss,val\_accuracy**  **157/157 [==============================] - 99s 631ms/step - loss: 0.3256 - accuracy: 0.8633 - val\_loss: 0.3715 - val\_accuracy: 0.8406**  **Epoch 5/10**  **157/157 [==============================] - ETA: 0s - loss: 0.3152 - accuracy: 0.8677WARNING:tensorflow:Early stopping conditioned on metric `val\_acc` which is not available. Available metrics are: loss,accuracy,val\_loss,val\_accuracy**  **157/157 [==============================] - 100s 637ms/step - loss: 0.3152 - accuracy: 0.8677 - val\_loss: 0.3533 - val\_accuracy: 0.8528**  **Epoch 6/10**  **157/157 [==============================] - ETA: 0s - loss: 0.3032 - accuracy: 0.8726WARNING:tensorflow:Early stopping conditioned on metric `val\_acc` which is not available. Available metrics are: loss,accuracy,val\_loss,val\_accuracy**  **157/157 [==============================] - 97s 615ms/step - loss: 0.3032 - accuracy: 0.8726 - val\_loss: 0.3449 - val\_accuracy: 0.8572**  **Epoch 7/10**  **157/157 [==============================] - ETA: 0s - loss: 0.2967 - accuracy: 0.8775WARNING:tensorflow:Early stopping conditioned on metric `val\_acc` which is not available. Available metrics are: loss,accuracy,val\_loss,val\_accuracy**  **157/157 [==============================] - 100s 640ms/step - loss: 0.2967 - accuracy: 0.8775 - val\_loss: 0.3903 - val\_accuracy: 0.8400**  **Epoch 8/10**  **157/157 [==============================] - ETA: 0s - loss: 0.2953 - accuracy: 0.8765WARNING:tensorflow:Early stopping conditioned on metric `val\_acc` which is not available. Available metrics are: loss,accuracy,val\_loss,val\_accuracy**  **157/157 [==============================] - 100s 640ms/step - loss: 0.2953 - accuracy: 0.8765 - val\_loss: 0.3800 - val\_accuracy: 0.8452**  **Epoch 9/10**  **157/157 [==============================] - ETA: 0s - loss: 0.2849 - accuracy: 0.8813WARNING:tensorflow:Early stopping conditioned on metric `val\_acc` which is not available. Available metrics are: loss,accuracy,val\_loss,val\_accuracy**  **157/157 [==============================] - 102s 650ms/step - loss: 0.2849 - accuracy: 0.8813 - val\_loss: 0.3489 - val\_accuracy: 0.8496**  **Epoch 10/10**  **157/157 [==============================] - ETA: 0s - loss: 0.2877 - accuracy: 0.8815WARNING:tensorflow:Early stopping conditioned on metric `val\_acc` which is not available. Available metrics are: loss,accuracy,val\_loss,val\_accuracy** |

|  |
| --- |
| print("Accuracy on Test set : {}".format(model.evaluate(x\_test, y\_test)[1]))  **782/782 [==============================] - 49s 62ms/step - loss: 0.3464 - accuracy: 0.8563**  **Accuracy on Test set : 0.8563200235366821** |

# Practical No.9

## Aim : Write a program for object detection from the image/video.

## Theory :

## **(Object Detection) identifies and locates objects in images or videos**. Object detection finds extensive applications across various sectors. The article aims to understand the fundamentals, of working, techniques, and applications of object detection.

### Key Components of Object Detection

#### **1. Image Classification**

Image classification assigns a label to an entire image based on its content. While it’s a crucial step in understanding visual data, it doesn’t provide information about the object’s location within the image.

#### **2. Object Localization**

Object localization goes a step further by not only identifying the object but also determining its position within the image. This involves drawing bounding boxes around the objects.

#### **3. Object Detection**

Object detection merges image classification and localization. It detects multiple objects in an image, assigns labels to them, and provides their locations through bounding boxes.

## 

**Program :**

|  |
| --- |
| !pip install -U --pre tensorflow=="2.\*"  !pip install tf\_slim  !pip install pycocotools |

|  |
| --- |
| import os  import pathlib  if "models" in pathlib.Path.cwd().parts:  while "models" in pathlib.Path.cwd.parts:  os.chdir("..")  elif not pathlib.Path("models").exists():  !git clone --depth 1 <https://github.com/tensorflow/models> |

|  |
| --- |
| %cd /content/models/research  !protoc object\_detection/protos/\*.proto --python\_out=.  **/content/models/research** |

|  |
| --- |
| %cd /content/models/research  **/content/models/research** |

|  |
| --- |
| import numpy as np  import os  import six.moves.urllib as urllib  import sys  import tarfile  import tensorflow as tf  import zipfile  from collections import defaultdict  from io import StringIO  from matplotlib import pyplot as plt  from PIL import Image  from IPython.display import display |

|  |
| --- |
| !pip install tensorflow-object-detection-api  from object\_detection.utils import ops as utils\_ops  from object\_detection.utils import label\_map\_util  from object\_detection.utils import visualization\_utils as vis\_util |

|  |
| --- |
| utils\_ops.tf = tf.compat.v1  tf.gfile = tf.io.gfile |

|  |
| --- |
| def load\_model(model\_name):  base\_url = "http://download.tensorflow.org/models/object\_detection/"  model\_file = model\_name + ".tar.gz"  model\_dir = tf.keras.utils.get\_file(  fname = model\_name,  origin = base\_url + model\_file,  untar = True)  model\_dir = pathlib.Path(model\_dir)/"saved\_model"  model = tf.saved\_model.load(str(model\_dir))  return model |

|  |
| --- |
| PATH\_TO\_LABELS = '/content/models/research/object\_detection/data/mscoco\_label\_map.pbtxt'  category\_index = label\_map\_util.create\_category\_index\_from\_labelmap(PATH\_TO\_LABELS, use\_display\_name = True) |

|  |
| --- |
| model\_name = 'ssd\_mobilenet\_v1\_coco\_2018\_01\_28'  detection\_model = load\_model(model\_name)  **Downloading data from http://download.tensorflow.org/models/object\_detection/ssd\_mobilenet\_v1\_coco\_2018\_01\_28.tar.gz**  **76541073/76541073 [==============================] - 1s 0us/step** |

|  |
| --- |
| print(detection\_model.signatures["serving\_default"].inputs)  **[<tf.Tensor 'image\_tensor:0' shape=(None, None, None, 3) dtype=uint8>]** |

|  |
| --- |
| detection\_model.signatures["serving\_default"].output\_shapes  **{'detection\_classes': TensorShape([None, 100]),**  **'detection\_boxes': TensorShape([None, 100, 4]),**  **'detection\_scores': TensorShape([None, 100]),**  **'num\_detections': TensorShape([None])}** |

|  |
| --- |
| def run\_inference\_for\_single\_Image(model, image):  image = np.asarray(image)  input\_tensor = tf.convert\_to\_tensor(image)  input\_tensor = input\_tensor[tf.newaxis, ...]  model\_fn = model.signatures['serving\_default']  output\_dict = model\_fn(input\_tensor)  num\_detections = int(output\_dict.pop('num\_detections'))  output\_dict = {key: value[0, :num\_detections].numpy() for key, value in output\_dict.items()}  output\_dict['num\_detections'] = num\_detections  output\_dict['detection\_classes'] = output\_dict['detection\_classes'].astype(np.int64)  if "detection\_masks in output\_dict":  detection\_masks\_reframed = utils\_ops.reframe\_box\_masks\_to\_image\_masks(  output\_dict['detection\_masks'], output\_dict['detection\_boxes'],  image.shape[0], image.shape[1])  detection\_masks\_reframed = tf.cast(detection\_masks\_reframed > 0.5, tf.uint8)  output\_dict['detection\_masks\_reframed'] = detection\_masks\_reframed.numpy()  return output\_dict |

|  |
| --- |
| def show\_inference(model, image\_path):  image\_np = np.array(Image.open(image\_path))  output\_dict = run\_inference\_for\_single\_image(model, image\_np)  vis\_util.visualize\_boxes\_and\_labels\_on\_image\_array(  image\_np,  output\_dict['detection\_boxes'],  output\_dict['detection\_classes'],  output\_dict['detection\_scores'],  category\_index,  instance\_masks = output\_dict.get( 'detection\_masks\_reframed', None),  use\_normalized\_coordinates = True,  line\_thickness = 8)  display(Image.fromarray(image\_np)) |

|  |
| --- |
| for image\_path in TEST\_IMAGE\_PATHS:  show\_inference(detection\_model, image\_path) |