220701045
BHUVAANESH R

RAJALAKSHMI ENGINEERING COLLEGE

An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CS19P18- DEEP LEARNING CONCEPTS LABORATORY

LAB MANUAL

FINAL YEAR

SEVENTH SEMESTER

2025- 2026

ODD SEMESTER

LIST OF EXPERIMENTS

1. Create a neural network to recognize handwritten digits using MNIST dataset
2. Build a Convolutional Neural Network with Keras/TensorFlow
3. Image Classification on CIFAR-10 Dataset using Convolutional Neural Networks
4. Transfer learning with CNN and Visualization
5. Build a Recurrent Neural Network using Keras/Tensorflow
6. Sentiment Classification of Text using Recurrent Neural Network (RNN)
7. Build autoencoders with Keras/TensorFlow
8. Build GAN with Keras/TensorFlow
9. Perform object detection with YOLO3
10. Mini Project – CNN based or RNN based applications

HARDWARE AND SOFTWARE REQUIREMENTS

Hardware Requirements	Core i5 and above/M1 Chip with minimum 8 GB RAM and 512 GB HDD
Software Requirements	Windows 10/Apple MacOS, Tensorflow, Keras, Numpy, Pandas and Scikit-learn

Course Outcomes (COs)

Course Name: Deep Learning Concepts **Course Code: CS19P18**

Outcome 1	Understand the fundamentals of deep learning based on optimizations and backpropagation and machine learning.
Outcome 2	Train neural network models that converge well without overfitting.
Outcome 3	Learn how to improve the deep learning model performance using error analysis, regularization, hyper parameter tuning.
Outcome 4	Build networks to perform sentiment analysis and work on real-time time series data.
Outcome 5	Analyse different supervised, unsupervised, and reinforcement deep learning models and their applications in real world scenarios; Build, train, test and evaluate neural networks for different applications and data types.

CO-PO –PSO matrices of course

PO/PSO CO	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO 1	PSO 2	PSO 3
CS19P18.1	3	2	2	-	1	-	-	-	-	-	1	1	2	1	1
CS19P18.2	2	2	2	-	2	-	-	-	-	-	1	2	3	2	2
CS19P18.3	3	3	1	3	2	-	-	-	-	-	1	2	2	2	2
CS19P18.4	2	1	3	-	2	1	1	1	-	1	2	3	3	3	3
CS19P18.5	3	1	1	3	2	2	1	1	1	2	3	3	2	3	3
Average	2.6	1.8	1.8	3.0	1.8	1.5	1.0	1.0	1.0	1.5	1.6	2.2	2.4	2.2	2.2

Note: Enter correlation levels1,2 or 3 as defined below:

1: Slight (Low) 2: Moderate (Medium) 3: Substantial (High) If there is no correlation, put "-"

RAJALAKSHMI ENGINEERING COLLEGE

An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

CS19P18 - DEEP LEARNING CONCEPTS LABORATORY

INDEX

RAJALAKSHMI ENGINEERING COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CS19P18- DEEP LEARNING CONCEPTS LABORATORY

LAB PLAN

Sl.No.	Name of the Experiment	Hours Planned
1	Create a neural network to recognize handwritten digits using MNIST dataset	2
2	Build a Convolutional Neural Network with Keras/TensorFlow	2
3	Image Classification on CIFAR-10 Dataset using CNN	2
4	Transfer learning with CNN and Visualization	2
5	Build a Recurrent Neural Network using Keras/Tensorflow	2
6	Sentiment Classification of Text using RNN	2
7	Build autoencoders with Keras/TensorFlow	2
8	Perform object detection with YOLO3	2
9	Build GAN with Keras/TensorFlow	2
10	Mini Project – CNN or RNN based applications	8

INSTALLATION AND CONFIGURATION OF TENSORFLOW

Aim:

To install and configure TensorFlow in anaconda environment in Windows 10.

Procedure:

1. Download Anaconda Navigator and install.

2. Open Anaconda prompt

3. Create a new environment dlc with python 3.7 using the following command:
   conda create -n dlc python=3.7

4. Activate newly created environment dlc using the following command:
   conda activate dlc

5. In dlc prompt, install tensorflow using the following command:
   pip install tensorflow

6. Next install Tensorflow-datasets using the following command:
   pip install tensorflow-datasets

7. Install scikit-learn package using the following command:
   pip install scikit-learn

8. Install pandas package using the following command:
   pip install pandas

9. Lastly, install jupyter notebook
   pip install jupyter notebook

10. Open jupyter notebook by typing the following in dlc prompt:
    jupyter notebook

11. Click create new and then choose python 3 (ipykernel)

12. Give the name to the file

13. Type the code and click Run button to execute (eg. Type import tensorflow and then run)

Useful Learning Resources:

1. https://docs.anaconda.com/free/anaconda/applications/tensorflow/

2. https://conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html#activating-an-environment

EX NO: 1 CREATE A NEURAL NETWORK TO RECOGNIZE HANDWRITTEN DIGITS USING MNIST DATASET

Aim:

To build a handwritten digit's recognition with MNIST dataset.

Procedure:

1. Download and load the MNIST dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a simple neural network model using Keras/TensorFlow.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

Useful Learning Resources:

1.https://www.analyticsvidhya.com/blog/2022/07/handwritten-digit-recognition-using-tensorflow/

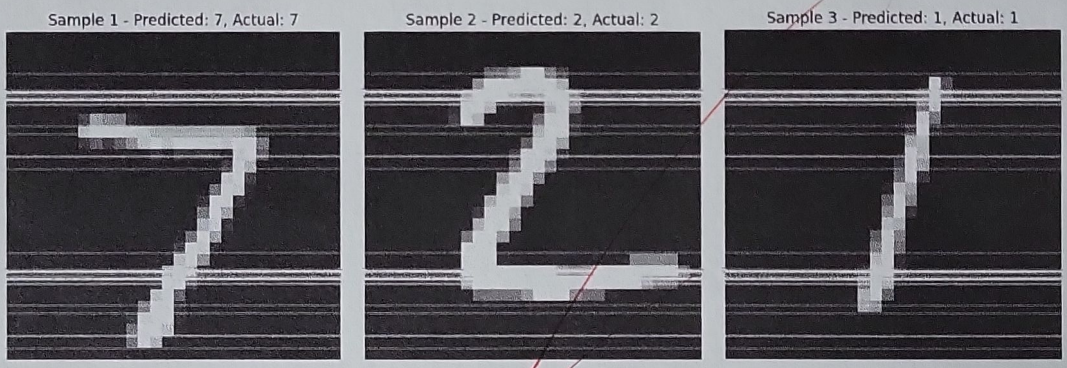
2. https://www.milindsoorya.com/blog/handwritten-digits-classification

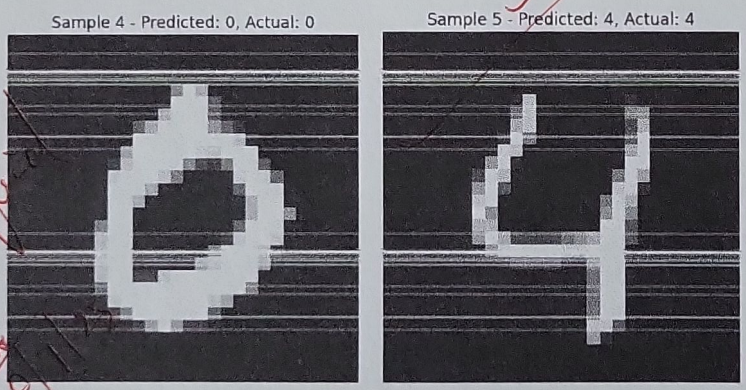
1. Create Neural Networks to Recognize Handwritten digits using MNIST Dataset.

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical

feature_vector_length = 784
num_classes = 10
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
X_train = X_train.reshape(X_train.shape[0], feature_vector_length)
X_test = X_test.reshape(X_test.shape[0], feature_vector_length)
X_train = X_train.astype('float32')/255
X_test = X_test.astype('float32')/255

Y_train = to_categorical(Y_train, num_classes)
Y_test = to_categorical(Y_test, num_classes)

model = Sequential()
model.add(Dense(350, input_shape=(feature_vector_length,), activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
```

```
model.compile(loss='categorical_crossentropy', optimizer
              optimizer='adam', metrics=['accuracy'])

model.fit(X_train, Y_train, epochs=10, batch_size=250,
          verbose=1, validation_split=0.2)

test_results = model.evaluate(X_test, Y_test, verbose=1)
print(f'In Test results - Loss: {test_results[0]} -
      Accuracy: {test_results[1]}')

predictions = model.predict(X_test[:5])
predicted_classes = np.argmax(predictions, axis=1)
true_classes = np.argmax(Y_test[:5], axis=1)

for i in range(5):
    plt.imshow(X_test[i].reshape(28,28), cmap='gray')
    plt.title(f"Sample {i+1} - Predicted: {predicted_classes[i]}
              Actual: {true_classes[i]}")
    plt.axis('off')
    plt.show()
```

```
Epoch 1/10
192/192 [==============================] - 3s 11ms/step - loss: 0.3909 - accuracy: 0.8925 - val_loss: 0.1783 - val_accuracy: 0.
9585
Epoch 2/10
192/192 [==============================] - 2s 11ms/step - loss: 0.1483 - accuracy: 0.9574 - val_loss: 0.1281 - val_accuracy: 0.
9643
Epoch 3/10
192/192 [==============================] - 2s 11ms/step - loss: 0.1015 - accuracy: 0.9700 - val_loss: 0.1139 - val_accuracy: 0.
9653
Epoch 4/10
192/192 [==============================] - 2s 11ms/step - loss: 0.0733 - accuracy: 0.9780 - val_loss: 0.1032 - val_accuracy: 0.
9690
Epoch 5/10
192/192 [==============================] - 2s 10ms/step - loss: 0.0543 - accuracy: 0.9844 - val_loss: 0.0874 - val_accuracy: 0.
9752
Epoch 6/10
192/192 [==============================] - 2s 11ms/step - loss: 0.0413 - accuracy: 0.9887 - val_loss: 0.0840 - val_accuracy: 0.
9732
Epoch 7/10
192/192 [==============================] - 2s 11ms/step - loss: 0.0307 - accuracy: 0.9919 - val_loss: 0.0841 - val_accuracy: 0.
9742
Epoch 8/10
192/192 [==============================] - 2s 11ms/step - loss: 0.0257 - accuracy: 0.9929 - val_loss: 0.0976 - val_accuracy: 0.
9707
Epoch 9/10
192/192 [==============================] - 2s 11ms/step - loss: 0.0180 - accuracy: 0.9954 - val_loss: 0.0852 - val_accuracy: 0.
9747
Epoch 10/10
192/192 [==============================] - 2s 11ms/step - loss: 0.0141 - accuracy: 0.9966 - val_loss: 0.0823 - val_accuracy: 0.
9785
313/313 [==============================] - 1s 2ms/step - loss: 0.0700 - accuracy: 0.9793
Test results - Loss: 0.06997900456190109 - Accuracy: 0.9793000221252441
1/1 [==============================] - 0s 84ms/step
```





Result:

Thus building of MNIST Dataset to detect handwritten digit's recognition is sucessfully completed and verified.

EX NO:2 **BUILD A CONVOLUTIONAL NEURAL NETWORK USING KERAS/TENSORFLOW**

Aim:

To implement a Convolutional Neural Network (CNN) using Keras/TensorFlow to recognize and classify handwritten digits from the MNIST dataset with high accuracy.

Procedure:

1. Import required libraries (TensorFlow/Keras, NumPy, etc.).
2. Load the MNIST dataset from Keras.
3. Normalize and reshape the image data.
4. Convert labels to one-hot encoded vectors.
5. Build a CNN model with Conv2D, MaxPooling, Flatten, and Dense layers.
6. Compile the model using categorical crossentropy and Adam optimizer.
7. Train the model on training data.
8. Evaluate the model on test data.
9. Display accuracy and predictions.

Useful Learning Resources:

1. https://towardsdatascience.com/build-your-first-cnn-with-tensorflow-a9d7394eaa2e

2.https://www.analyticsvidhya.com/blog/2021/06/building-a-convolutional-neural-network-using-tensorflow-keras/

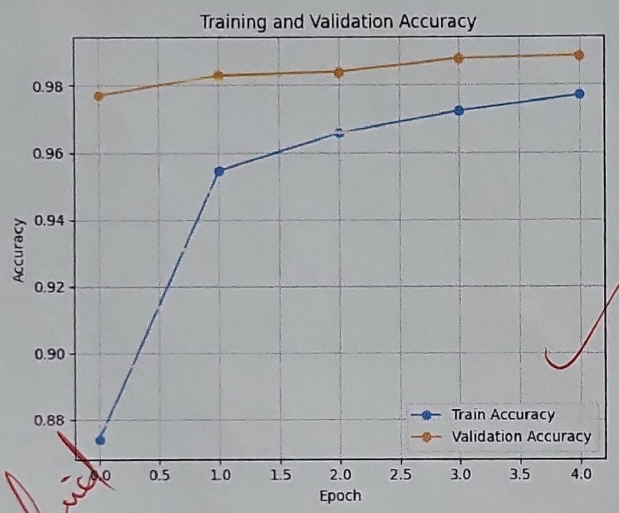
2. Build a Convolutional Neural Network using Keras / Tensorflow

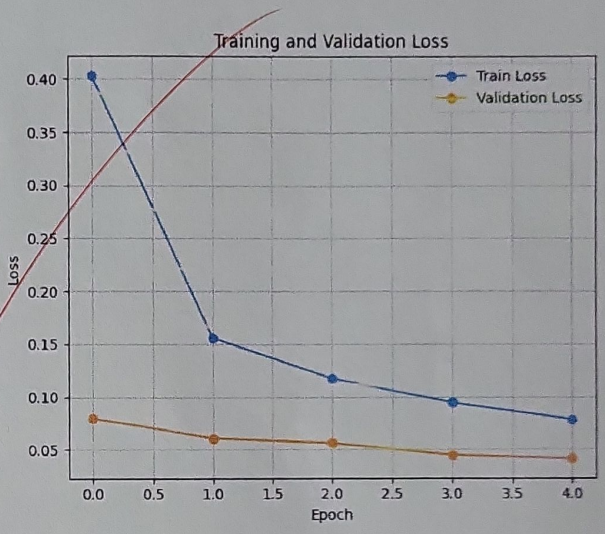
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D,
                                    Flatten, Dense, Dropout

import matplotlib.pyplot as plt
import numpy as np

(train_images, train_labels), (test_images, test_labels) =
                                        mnist.load_data()

train_images = train_images / 255.0
test_images = test_images / 255.0
train_images = train_images.reshape(-1, 28, 28, 1)
test_images = test_images.reshape(-1, 28, 28, 1)

model = Sequential([
    Conv2D(32, (3,3), activation = 'relu', input_shape = (28,28,1)),
    MaxPooling2D((2,2)),
    Conv2D(64, (3,3), activation = 'relu'),
    MaxPooling2D((2,2)),
    Flatten()
    Dense(64, activation = 'relu'),
    Dropout(0.5),
    Dense(10, activation = 'softmax')
])

model.compile(optimizer = 'adam', loss = 'sparse_categorial_
                                        crossentropy',
              metrics = ['accuracy'])
```

```
history = model.fit(train_images, train_labels, epochs=5,
                    batch_size=64, validation_split=0.2)

test_loss, test_acc = model.evaluate(test_images,
                                     test_labels)

print(f"\nTest accuracy: {test_acc:.4f}")
print(f"Test loss: {test_loss:.4f}")

plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Accuracy',
         marker='o')
plt.title('Trainning and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)

plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss', marker='o')
plt.plot(history.history[val_loss], label='Validation Loss',
         marker='o')
plt.title('Trainning and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```

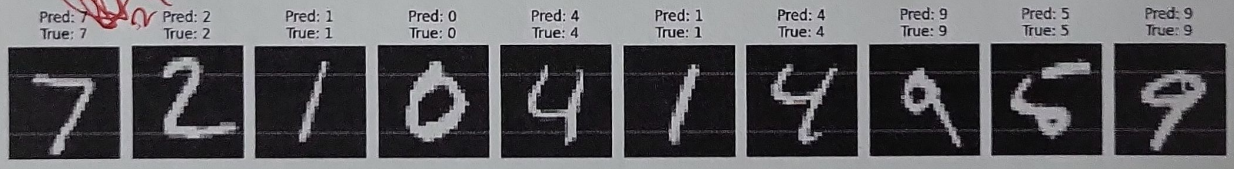
```
predictions = model.predict(test_images)
predicted_labels = np.argmax(predictions, axis=1)

num_samples = 10
plt.figure(figsize=(15,4))
plt.suptitle("Sample Predictions on Test Images",
             fontsize=16)

for i in range(num_samples):
    plt.subplot(1, num_samples, i+1)
    plt.imshow(test_images[i].reshape(28,28),
               cmap='gray')
    plt.title(f"Pred: {predicted_labels[i]}\nTrue:
              {test_labels[i]}")
    plt.axis('off')

plt.show()
```

```
Epoch 1/5
750/750 [==============================] - 20s 25ms/step - loss: 0.3594 - accuracy: 0.8900 - val_loss: 0.0721 - val_accuracy:
0.9793
Epoch 2/5
750/750 [==============================] - 19s 25ms/step - loss: 0.1389 - accuracy: 0.9586 - val_loss: 0.0497 - val_accuracy:
0.9859
Epoch 3/5
750/750 [==============================] - 19s 25ms/step - loss: 0.1019 - accuracy: 0.9704 - val_loss: 0.0463 - val_accuracy:
0.9870
Epoch 4/5
750/750 [==============================] - 18s 24ms/step - loss: 0.0840 - accuracy: 0.9755 - val_loss: 0.0457 - val_accuracy:
0.9868
Epoch 5/5
750/750 [==============================] - 19s 25ms/step - loss: 0.0727 - accuracy: 0.9786 - val_loss: 0.0424 - val_accuracy:
0.9885
313/313 [==============================] - 2s 7ms/step - loss: 0.0380 - accuracy: 0.9868

Test accuracy: 0.9868
Test loss: 0.0380
```





O/P Verified 27/8/25



Result:

Implementation of CNN using Keras/Tensorflow has been successfully completed.

EX NO: 3 IMAGE CLASSIFICATION ON CIFAR-10 DATASET USING CNN

Aim:

To build a Convolutional Neural Network (CNN) model for classifying images from the CIFAR-10 dataset into one of the ten categories such as airplanes, cars, birds, cats, etc.

Procedure:

1. Download and load the CIFAR-10 dataset using Keras/TensorFlow.
2. Visualize and analyze sample images from the dataset.
3, Preprocess the data:
    - Normalize the pixel values (divide by 255)
    - Convert class labels to one-hot encoded format
4. Build a CNN model using Keras/TensorFlow:
    - Include convolutional, pooling, flatten, and dense layers.
5. Compile the model with suitable loss function and optimizer.
6. Train the model using training data and validate using test data.
7. Evaluate the model using accuracy and loss on test dataset.
8. Perform predictions on new/unseen CIFAR-10 images.
9 Visualize prediction results with sample images and predicted labels.

Useful Learning Resources:

1.https://www.analyticsvidhya.com/blog/2021/01/image-classification-using-convolutional-neural-networks-a-step-by-step-guide/

2. https://www.geeksforgeeks.org/deep-learning/cifar-10-image-classification-in-tensorflow/

3 IMAGE CLASSIFICATION ON CIFAR-10 DATASET USING CNN

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.
                                        load_data()

x_train = x_train.astype("float32")/255.0
x_test = x_test.astype('float32')/255.0

y_train = tf.keras.utils.to_categorical(y_train,10)
y_test = tf.keras.utils.to_categorical(y_test,10)

model = tf.keras.Sequential()
model.add(tf.keras.layers.Conv2D(32,(3,3),activation='relu',
                                 input_shape=(32,32,3)))
model.add(tf.keras.layers.MaxPooling2D(2,2)))
model.add(tf.keras.layers.Conv2D(64,(3,3),activation='relu'))
model.add(tf.keras.layers.MaxPooling2D(2,2)))
model.add(tf.keras.layers.Conv2D(64,(3,3),activation='relu'))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(64,activation='relu'))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(64,activation='relu'))
model.add(tf.keras.layers.Dense(10,activation='softmax'))

model.compile(optimizer='adam',loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x_train,y_train,epochs=10,batch_size=64,
          validation_split=0.2)
class_names=['airplane','automobile','bird','cat','deer',
```

```
'dog', 'frog', 'horse', 'ship', 'truck']

index = int(input("Enter the index (0 to 9999) for test
                                    image: "))

if index<0 or index >= len(x_test):
    print("Invalid Index. Using index 0 by default")
    index = 0

test_image = x_test[index]
true_label = np.argmax(y_test[index])

prediction = model.predict(np.expand_dims(test_image,
                                    axis=0)

predicted_label = np.argmax(prediction)

plt.figure(figsize=(4,4))
resized_image = tf.image.resize(test_image,[128,128])
plt.imshow(resized_image)
plt.axis('off')
plt.title(f"Predicted: {class_names[predicted_label]}\n
                    Actual: {class_names[true_label]}")

plt.show()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
Epoch 1/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 65s 100ms/step - accuracy: 0.2980 - loss: 1.8798 - val_accuracy: 0.5005 - val_loss: 1.3788
Epoch 2/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 79s 95ms/step - accuracy: 0.5178 - loss: 1.3441 - val_accuracy: 0.5514 - val_loss: 1.2893
Epoch 3/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 89s 106ms/step - accuracy: 0.5740 - loss: 1.1922 - val_accuracy: 0.5963 - val_loss: 1.1365
Epoch 4/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 72s 91ms/step - accuracy: 0.6184 - loss: 1.0891 - val_accuracy: 0.6097 - val_loss: 1.1023
Epoch 5/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 90s 104ms/step - accuracy: 0.6458 - loss: 1.0144 - val_accuracy: 0.6533 - val_loss: 0.9941
Epoch 6/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 77s 95ms/step - accuracy: 0.6722 - loss: 0.9426 - val_accuracy: 0.6526 - val_loss: 0.9952
Epoch 7/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 79s 90ms/step - accuracy: 0.6884 - loss: 0.8952 - val_accuracy: 0.6586 - val_loss: 1.0034
Epoch 8/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 82s 91ms/step - accuracy: 0.7019 - loss: 0.8472 - val_accuracy: 0.6713 - val_loss: 0.9469
Epoch 9/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 82s 92ms/step - accuracy: 0.7127 - loss: 0.8211 - val_accuracy: 0.6445 - val_loss: 1.0233
Epoch 10/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 82s 92ms/step - accuracy: 0.7344 - loss: 0.7643 - val_accuracy: 0.6841 - val_loss: 0.9155
```

```
Enter an index (0 to 9999) for test image: 4
```

Predicted: deer
Actual: frog

```
Enter an index (0 to 9999) for test image: 144
```

Predicted: ship
Actual: ship

O/P Verified

3/9/25

Result:

Thus CNN has been successfully built to classify images from CIFAR 10 dataset

3/9/25

Ex No: 4 **TRANSFER LEARNING WITH CNN AND VISUALIZATION**

Aim:

To build a convolutional neural network with transfer learning and perform visualization

Procedure:

1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a simple neural network model using Keras/TensorFlow.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

Useful Learning Resources:

1.https://medium.com/analytics-vidhya/car-brand-classification-using-vgg16-transfer-learning-f219a0f09765

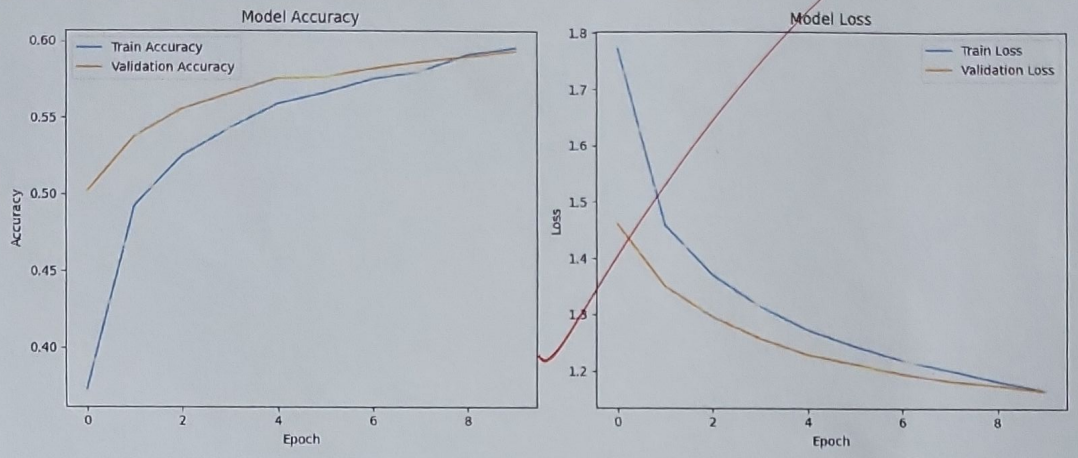
2. https://www.kaggle.com/code/kasmithh/transfer-learning-using-keras-vgg-16

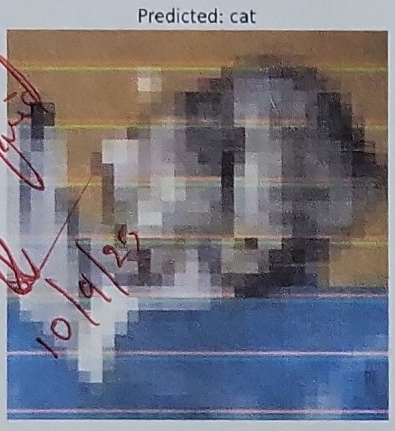
4.

```
conda install -c conda-forge python-graphviz -y
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.applications import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import plot_model
import matplotlib.pyplot as plt
import numpy as np
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train = x_train/255.0
x_test = x_test/255.0
vgg_base = VGG16(weights = 'imagenet', include_top = False,
                 input_shape = (32,32,3))

for layer in vgg_base.layers:
    layer.trainable = False

model = Sequential()
model.add(vgg_base)
model.add(Flatten())
model.add(Dense(512, activation = 'relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation = 'softmax'))

model.compile(optimizer=Adam(learning_rate=0.0001),
              loss='sparse_categorical_entropy',
              metrics=['accuracy'])

plot_model(model, to_file='cnn.png', show_shapes=True
```

```
show_layer_names=True, dpi=300)

plt.figure(figsize=(20,20))
img=plt.imread('cnn.png')
plt.imshow(img)
 plt.axis('off')
 plt.show()

class_names=['airplane', 'automobile', 'bird', 'cat',
             'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

sample_x_test[0].reshape(1,32,32,3)
prediction=model.predict(sample)
predicted_class=class_names[np.argmax(prediction)]

plt.imshow(x_test[0])
plt.title(f"Predicted: {predicted_class}")
plt.axis('off')
 plt.show()
```

```
Epoch 1/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 709s 566ms/step - accuracy: 0.2815 - loss: 1.9996 - val_accuracy: 0.5018 - val_loss: 1.4607
Epoch 2/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 736s 562ms/step - accuracy: 0.4801 - loss: 1.4928 - val_accuracy: 0.5378 - val_loss: 1.3503
Epoch 3/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 745s 564ms/step - accuracy: 0.5245 - loss: 1.3689 - val_accuracy: 0.5556 - val_loss: 1.2961
Epoch 4/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 706s 565ms/step - accuracy: 0.5392 - loss: 1.3180 - val_accuracy: 0.5656 - val_loss: 1.2581
Epoch 5/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 742s 565ms/step - accuracy: 0.5549 - loss: 1.2812 - val_accuracy: 0.5758 - val_loss: 1.2299
Epoch 6/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 706s 565ms/step - accuracy: 0.5645 - loss: 1.2534 - val_accuracy: 0.5765 - val_loss: 1.2129
Epoch 7/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 732s 557ms/step - accuracy: 0.5739 - loss: 1.2265 - val_accuracy: 0.5823 - val_loss: 1.1959
Epoch 8/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 743s 558ms/step - accuracy: 0.5827 - loss: 1.2013 - val_accuracy: 0.5866 - val_loss: 1.1832
Epoch 9/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 745s 561ms/step - accuracy: 0.5952 - loss: 1.1754 - val_accuracy: 0.5901 - val_loss: 1.1753
Epoch 10/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 742s 561ms/step - accuracy: 0.5960 - loss: 1.1647 - val_accuracy: 0.5939 - val_loss: 1.1660
313/313 ━━━━━━━━━━━━━━━━━━━━ 138s 442ms/step - accuracy: 0.5919 - loss: 1.1740
Test Loss: 1.1809
Test Accuracy: 58.85%
```





O/P Verified 10/4/25

Result:

Thus CNN has been built to perform visualization successfully.

13/9/25

EX NO: 5 BUILD A RECURRENT NEURAL NETWORK (RNN) USING KERAS/TENSORFLOW

Aim:

To build a recurrent neural network with Keras/TensorFlow.

Procedure:

1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a simple neural network model using Keras/TensorFlow.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

Useful Learning Resources:

1. https://machinelearningmastery.com/understanding-simple-recurrent-neural-networks-in-keras/
2. https://victorzhou.com/blog/keras-rnn-tutorial/

5.

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense
from sklearn.metrics import r2_score
np.random.seed(0)
seq_length = 10
num_samples = 1000

X = np.random.randn(num_samples, seq_length, 1)
Y = X.sum(axis=1) + 0.1 * np.random.randn(num_samples, 1)
split_ratio = 0.8
split_index = int(split_ratio * num_samples)

X_train, X_test = X[:split_index], X[split_index:]
y_train, y_test = Y[:split_index], Y[split_index:]
model = Sequential()
model.add(SimpleRNN(units=50, activation='relu', input_shape
                                    =(seq_length, 1)))

model.add(Dense(units=1))

model.compile(optimizer='adam', loss='mean_squared_error')
model.summary()

batch_size = 30
epochs = 50
history = model.fit(
    X_train, y_train
    batch_size = batch_size
    epochs = epochs,
    validation_split = 0.2
)
```

```
test_loss = model.evaluate(X_test, Y_test)
print(f'Test Loss: {test_loss:.4f}')

y_pred = model.predict(X_test)
r2 = r2_score(y_test, y_pred)
print(f'Test Accuracy (R^2): {r2:.4f}')

new_data = np.random.random(5, seq_length, 1)
predictions = model.predict(new_data)
print("Predictions for New data:")
print(predictions)
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn (SimpleRNN)      (None, 50)                2600

 dense (Dense)               (None, 1)                 51

=================================================================
Total params: 2,651
Trainable params: 2,651
Non-trainable params: 0
_________________________________________________________________
Epoch 1/50
22/22 [==============================] - 2s 22ms/step - loss: 9.2267 - val_loss: 7.4353
Epoch 2/50
22/22 [==============================] - 0s 7ms/step - loss: 7.1066 - val_loss: 2.5489
Epoch 3/50
22/22 [==============================] - 0s 6ms/step - loss: 1.5503 - val_loss: 0.7508
Epoch 4/50
22/22 [==============================] - 0s 6ms/step - loss: 0.6063 - val_loss: 0.5178
Epoch 5/50
22/22 [==============================] - 0s 6ms/step - loss: 0.3665 - val_loss: 0.2358
Epoch 6/50
22/22 [==============================] - 0s 6ms/step - loss: 0.1886 - val_loss: 0.1647
Epoch 7/50
22/22 [==============================] - 0s 6ms/step - loss: 0.1215 - val_loss: 0.1211
Epoch 8/50
22/22 [==============================] - 0s 6ms/step - loss: 0.0847 - val_loss: 0.1006
Epoch 9/50
22/22 [==============================] - 0s 6ms/step - loss: 0.0753 - val_loss: 0.0876
Epoch 10/50
22/22 [==============================] - 0s 7ms/step - loss: 0.0485 - val_loss: 0.0484
Epoch 11/50
22/22 [==============================] - 0s 6ms/step - loss: 0.0366 - val_loss: 0.0487
Epoch 12/50
22/22 [==============================] - 0s 6ms/step - loss: 0.0309 - val_loss: 0.0408
Epoch 13/50
22/22 [==============================] - 0s 6ms/step - loss: 0.0327 - val_loss: 0.0475
Epoch 14/50
22/22 [==============================] - 0s 6ms/step - loss: 0.0303 - val_loss: 0.0340
Epoch 15/50
22/22 [==============================] - 0s 6ms/step - loss: 0.0287 - val_loss: 0.0374
Epoch 16/50
22/22 [==============================] - 0s 6ms/step - loss: 0.0265 - val_loss: 0.0315
Epoch 17/50
22/22 [==============================] - 0s 6ms/step - loss: 0.0228 - val_loss: 0.0317
Epoch 18/50
22/22 [==============================] - 0s 6ms/step - loss: 0.0202 - val_loss: 0.0379
Epoch 19/50
22/22 [==============================] - 0s 6ms/step - loss: 0.0287 - val_loss: 0.0326
Epoch 20/50
22/22 [==============================] - 0s 6ms/step - loss: 0.0297 - val_loss: 0.0323
Epoch 21/50
22/22 [==============================] - 0s 6ms/step - loss: 0.0225 - val_loss: 0.0299
Epoch 22/50
22/22 [==============================] - 0s 6ms/step - loss: 0.0208 - val_loss: 0.0260
Epoch 23/50
22/22 [==============================] - 0s 8ms/step - loss: 0.0248 - val_loss: 0.0484
Epoch 24/50
22/22 [==============================] - 0s 6ms/step - loss: 0.0311 - val_loss: 0.0454
Epoch 25/50
```

```
Epoch 25/50
22/22 [==============================] - 0s 6ms/step - loss: 0.0258 - val_loss: 0.0282
Epoch 26/50
22/22 [==============================] - 0s 6ms/step - loss: 0.0218 - val_loss: 0.0386
Epoch 27/50
22/22 [==============================] - 0s 6ms/step - loss: 0.0198 - val_loss: 0.0231
Epoch 28/50
22/22 [==============================] - 0s 6ms/step - loss: 0.0184 - val_loss: 0.0260
Epoch 29/50
22/22 [==============================] - 0s 6ms/step - loss: 0.0159 - val_loss: 0.0223
Epoch 30/50
22/22 [==============================] - 0s 6ms/step - loss: 0.0160 - val_loss: 0.0352
Epoch 31/50
22/22 [==============================] - 0s 6ms/step - loss: 0.0208 - val_loss: 0.0229
Epoch 32/50
22/22 [==============================] - 0s 6ms/step - loss: 0.0189 - val_loss: 0.0329
Epoch 33/50
22/22 [==============================] - 0s 6ms/step - loss: 0.0219 - val_loss: 0.0287
Epoch 34/50
22/22 [==============================] - 0s 7ms/step - loss: 0.0189 - val_loss: 0.0306
Epoch 35/50
22/22 [==============================] - 0s 6ms/step - loss: 0.0193 - val_loss: 0.0249
Epoch 36/50
22/22 [==============================] - 0s 6ms/step - loss: 0.0248 - val_loss: 0.0232
Epoch 37/50
22/22 [==============================] - 0s 7ms/step - loss: 0.0171 - val_loss: 0.0286
Epoch 38/50
22/22 [==============================] - 0s 7ms/step - loss: 0.0181 - val_loss: 0.0227
Epoch 39/50
22/22 [==============================] - 0s 7ms/step - loss: 0.0209 - val_loss: 0.0500
Epoch 40/50
22/22 [==============================] - 0s 6ms/step - loss: 0.0264 - val_loss: 0.0244
Epoch 41/50
22/22 [==============================] - 0s 6ms/step - loss: 0.0377 - val_loss: 0.0243
Epoch 42/50
22/22 [==============================] - 0s 6ms/step - loss: 0.0171 - val_loss: 0.0250
Epoch 43/50
22/22 [==============================] - 0s 6ms/step - loss: 0.0154 - val_loss: 0.0243
Epoch 44/50
22/22 [==============================] - 0s 6ms/step - loss: 0.0170 - val_loss: 0.0200
Epoch 45/50
22/22 [==============================] - 0s 6ms/step - loss: 0.0155 - val_loss: 0.0330
Epoch 46/50
22/22 [==============================] - 0s 6ms/step - loss: 0.0147 - val_loss: 0.0237
Epoch 47/50
22/22 [==============================] - 0s 7ms/step - loss: 0.0128 - val_loss: 0.0201
Epoch 48/50
22/22 [==============================] - 0s 6ms/step - loss: 0.0127 - val_loss: 0.0223
Epoch 49/50
22/22 [==============================] - 0s 6ms/step - loss: 0.0145 - val_loss: 0.0368
Epoch 50/50
22/22 [==============================] - 0s 6ms/step - loss: 0.0155 - val_loss: 0.0196

Test Loss: 0.0186
Test R² Score: 0.9980
Predictions for new data:
[[ 1.6310315 ]
 [ 0.36440063]
 [-2.092144  ]
 [-0.47431982]
 [-3.7944572 ]]
```

```
Training...
Epoch 1/2
313/313 ━━━━━━━━━━━━━━━━━━━━ 18s 48ms/step - accuracy: 0.6393 - loss: 0.6146 - val_accuracy: 0.7982 - val_loss: 0.4475
Epoch 2/2
313/313 ━━━━━━━━━━━━━━━━━━━━ 15s 46ms/step - accuracy: 0.8536 - loss: 0.3623 - val_accuracy: 0.8254 - val_loss: 0.3868
<keras.src.callbacks.history.History at 0x1fe25e23290>

782/782 ━━━━━━━━━━━━━━━━━━━━ 8s 11ms/step - accuracy: 0.8284 - loss: 0.3914

Test Accuracy: 0.8305

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 290ms/step
Review text: ? please give this one a miss br br ? ? and the rest of the cast ? terrible performances the show is flat flat flat br br i don't know how m
ichael ? could have allowed this one on his ? he almost seemed to know this wasn't going to work out and his performance was quite ? so all you ? fans gi
ve this a miss
Predicted Sentiment: Negative
```

O/P Verified 17/9/25

Result:

Thus RNN has been built successfully using Tensorflow / Keras.

17/9/25

EX NO: 6 SENTIMENT CLASSIFICATION OF TEXT USING RNN

Aim:

To implement a Recurrent Neural Network (RNN) using Keras/TensorFlow for classifying the sentiment of text data (e.g., movie reviews) as positive or negative.

Procedure:

1. Import necessary libraries.
2. Load and preprocess the text dataset (e.g., IMDb).
3. Pad sequences and prepare labels.
4. Build an RNN model with Embedding and SimpleRNN layers.
5. Compile the model with loss and optimizer.
6. Train the model on training data.
7. Evaluate the model on test data.
8. Predict sentiment for new inputs

Useful Learning Resources:

1.https://medium.com/@muhammadluay45/sentiment-analysis-using-recurrent-neural-network-rnn-long-short-term-memory-lstm-and-38d6e670173f

2. https://www.ijert.org/text-classification-using-rnn

6. SENTIMENT CLASSIFICATION OF TEXT USING RNN

```
import numpy as np
import tensorflow.keras.datasets.imdb as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence
                        import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding,
                        SimpleRNN, Dense

max_words = 5000
max_len = 200
(x_train, y_train), (x_test, y_test) = imdb.load_data
                        (num_words=max_words)

model = Sequential()
model.add (Embedding (input_dim=max_words,
            output_dim=32, input_length=max_len))
model.add (SimpleRNN(32))
model.add (Dense(1, activation='sigmoid'))
model.compile (optimizer='adam', loss='binary_crossentropy'
                        metrics=['accuracy'])

print("Training...")
model.fit(x_train, y_train, epochs=32, batch_size=64,
                        validation_split=0.2)

loss, acc = model.evaluate(x_test, y_test)
print(f"\n Test Accuracy: {acc:.4f}")
word_index = imdb.get_word_index()
reverse_word_index = {v: k for (k, v) in word_index.items()}
```

```
def decode_review(review):
    return " ".join([reverse_word_index.get(i-3, "?")
                     for i in review])

prediction = model.predict(sample_review.reshape
                           (1, -1))[0][0]

print("Predicted Sentiment:" "Positive" if Prediction >0.5
                             else "Negative")
```

```
Epoch 1/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - loss: 0.3847 - val_loss: 0.1863
Epoch 2/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.1774 - val_loss: 0.1529
Epoch 3/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.1488 - val_loss: 0.1338
Epoch 4/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.1318 - val_loss: 0.1215
Epoch 5/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.1205 - val_loss: 0.1131
Epoch 6/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.1127 - val_loss: 0.1068
Epoch 7/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.1068 - val_loss: 0.1025
Epoch 8/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.1030 - val_loss: 0.0991
Epoch 9/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0998 - val_loss: 0.0969
Epoch 10/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0977 - val_loss: 0.0954
Epoch 11/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0963 - val_loss: 0.0943
Epoch 12/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0952 - val_loss: 0.0938
Epoch 13/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0950 - val_loss: 0.0933
Epoch 14/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0944 - val_loss: 0.0930
Epoch 15/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0940 - val_loss: 0.0928
Epoch 16/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0940 - val_loss: 0.0925
Epoch 17/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0940 - val_loss: 0.0924
Epoch 18/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0936 - val_loss: 0.0923
Epoch 19/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0936 - val_loss: 0.0922
Epoch 20/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0934 - val_loss: 0.0921
Epoch 21/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0933 - val_loss: 0.0921
Epoch 22/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0934 - val_loss: 0.0920
Epoch 23/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0932 - val_loss: 0.0920
Epoch 24/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0931 - val_loss: 0.0919
Epoch 25/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0932 - val_loss: 0.0919
```

```
Epoch 26/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0930 - val_loss: 0.0919
Epoch 27/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0928 - val_loss: 0.0918
Epoch 28/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0928 - val_loss: 0.0918
Epoch 29/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0930 - val_loss: 0.0918
Epoch 30/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0929 - val_loss: 0.0917
Epoch 31/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0929 - val_loss: 0.0918
Epoch 32/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0929 - val_loss: 0.0917
Epoch 33/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0931 - val_loss: 0.0916
Epoch 34/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0927 - val_loss: 0.0917
Epoch 35/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0929 - val_loss: 0.0917
Epoch 36/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0927 - val_loss: 0.0917
Epoch 37/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0926 - val_loss: 0.0917
Epoch 38/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0928 - val_loss: 0.0917
Epoch 39/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0927 - val_loss: 0.0916
Epoch 40/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0926 - val_loss: 0.0916
Epoch 41/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0927 - val_loss: 0.0917
Epoch 42/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0928 - val_loss: 0.0916
Epoch 43/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0929 - val_loss: 0.0916
Epoch 44/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0926 - val_loss: 0.0916
Epoch 45/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0926 - val_loss: 0.0915
Epoch 46/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0926 - val_loss: 0.0916
Epoch 47/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0928 - val_loss: 0.0916
Epoch 48/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0926 - val_loss: 0.0915
Epoch 49/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0926 - val_loss: 0.0915
Epoch 50/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0924 - val_loss: 0.0915
<keras.src.callbacks.history.History at 0x1d49cb551f0>
```

```
313/313 ——————————— 1s 2ms/step - loss: 0.0922
313/313 ——————————— 1s 2ms/step
```

```
Test Loss: 0.09151268750429153
Test Accuracy: 0.9713605867346938
```

O/P Verified
24/9/25.

Result:

RNN has been implemented successfully for classifying the sentiment of Text data.
24/9/25

Ex No: 7 **BUILD AUTOENCODERS WITH KERAS/TENSORFLOW**

Aim:

To build autoencoders with Keras/TensorFlow.

Procedure:

1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a simple neural network model using Keras/TensorFlow.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

Useful Learning Resources:

1. https://blog.keras.io/building-autoencoders-in-keras.html
2. https://towardsdatascience.com/how-to-make-an-autoencoder-2f2d99cd5103

1.

```
import numpy as np
import matplotlib.pyplot as plt
from keras.layers import Input, Dense
from keras.models import Model
from keras.datasets import mnist

(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:])))

input_img = Input(shape=(784,))
encoded = Dense(784, activation='sigmoid')(encoded)
autoencoder = Model(input_img, decoded)

autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

autoencoder.fit(x_train, x_train,
                epochs=50
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))

test_loss = autoencoder.evaluate(x_test, x_test)
decoded_imgs = autoencoder.predict(x_test)
```

```
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_xaxis().set_visible(False)

  ax = plt.subplot(2,n,i+1+n)
  reconstruction = decoded_imgs[i].reshape(28,28)
  plt.imshow(np.where(reconstruction >= threshold,
                      1.0,0.0))
  plt.gray()
  ax.get_xaxis().set_visible(False)
plt.show()
```

```
loading weights of convolution #0
loading weights of convolution #1
loading weights of convolution #2
loading weights of convolution #3
no convolution #4
loading weights of convolution #5
loading weights of convolution #6
loading weights of convolution #7
no convolution #8
loading weights of convolution #9
loading weights of convolution #10
no convolution #11
loading weights of convolution #12
loading weights of convolution #13
loading weights of convolution #14
no convolution #15
loading weights of convolution #16
loading weights of convolution #17
no convolution #18
loading weights of convolution #19
loading weights of convolution #20
no convolution #21
loading weights of convolution #22
loading weights of convolution #23
no convolution #24
loading weights of convolution #25
loading weights of convolution #26
no convolution #27
loading weights of convolution #28
loading weights of convolution #29
no convolution #30
loading weights of convolution #31
loading weights of convolution #32
no convolution #33
loading weights of convolution #34
loading weights of convolution #35
no convolution #36
loading weights of convolution #37
loading weights of convolution #38
loading weights of convolution #39
```

```
no convolution #40
loading weights of convolution #41
loading weights of convolution #42
no convolution #43
loading weights of convolution #44
loading weights of convolution #45
no convolution #46
loading weights of convolution #47
loading weights of convolution #48
no convolution #49
loading weights of convolution #50
loading weights of convolution #51
no convolution #52
loading weights of convolution #53
loading weights of convolution #54
no convolution #55
loading weights of convolution #56
loading weights of convolution #57
no convolution #58
loading weights of convolution #59
loading weights of convolution #60
no convolution #61
loading weights of convolution #62
loading weights of convolution #63
loading weights of convolution #64
no convolution #65
loading weights of convolution #66
loading weights of convolution #67
no convolution #68
loading weights of convolution #69
loading weights of convolution #70
no convolution #71
loading weights of convolution #72
loading weights of convolution #73
no convolution #74
loading weights of convolution #75
loading weights of convolution #76
loading weights of convolution #77
loading weights of convolution #78
loading weights of convolution #79
loading weights of convolution #80
```

```
loading weights of convolution #81
no convolution #82
no convolution #83
loading weights of convolution #84
no convolution #85
no convolution #86
loading weights of convolution #87
loading weights of convolution #88
loading weights of convolution #89
loading weights of convolution #90
loading weights of convolution #91
loading weights of convolution #92
loading weights of convolution #93
no convolution #94
no convolution #95
loading weights of convolution #96
no convolution #97
no convolution #98
loading weights of convolution #99

 loading weights of convolution #100
 loading weights of convolution #101
 loading weights of convolution #102
 loading weights of convolution #103
 loading weights of convolution #104
 loading weights of convolution #105
```

O/P. Verified 29/8/22

Result:

Autoencoders has been built successfully using Tensorflow / Keras.

Ex No: 8 **OBJECT DETECTION WITH YOLO3**

Aim:

To build an object detection model with YOLO3 using Keras/TensorFlow.

Procedure:

1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a simple neural network model using Keras/TensorFlow.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

Useful Learning Resources:

1. https://machinelearningmastery.com/how-to-perform-object-detection-with-yolov3-in-keras/
2. https://www.kaggle.com/code/roobansappani/yolo-v3-object-detection-using-keras

8.

```
import cv2
import matplotlib.pyplot as plt
import numpy as np

cfg_file='/content/yolov3.cfg'
weight_file='/content/yolov3.weights'
namesfile='/content/coco.names'

net=cv2.dnn.readNet(weight_file, cfg_file)

with open(namesfile,'r') as f:
    classes=f.read().strip().split('\n')

image_path='/content/hit.jpg'
image=cv2.imread(image_path)
height, width=image.shape[:2]

layer_names=net.getUnconnectedOutLayersNames()
outs=net.forward(layer_names)

class_ids=[]
confidences=[]
boxes=[]

conf_threshold=0.5
for out in outs:
    for detection in out:
        scores=detection[5:]
        class_id=np.argmax(scores)
        confidence=scores[class_id]
```

```
if confidence > conf_threshold:

    center_x = int(detection[0] * width)
    center_y = int(detection[1] * height)

    w = int(detection[2] * width)
    h = int(detection[3] * height)

    x = int(center_x - w/2)
    y = int(center_y - h/2)

    class_ids.append(class_id)
    confidences.append(float(confidence))
    boxes.append([x, y, w, h])

nms_threshold = 0.4
indices = cv2.dnn.NMSBoxes(boxes, confidences,
                           conf_threshold, nms_threshold)

    cv2.rectangle(image, (x, y), (x+w, y+h),
                  (0, 255, 0), 2)

    cv2.putText(image, f'{label} {confidence:.2f}',
                (x, y-10)

    cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 255, 0), 2)

plt.figure(figsize=(10, 8))
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))

plt.axis('off')
plt.show()
```

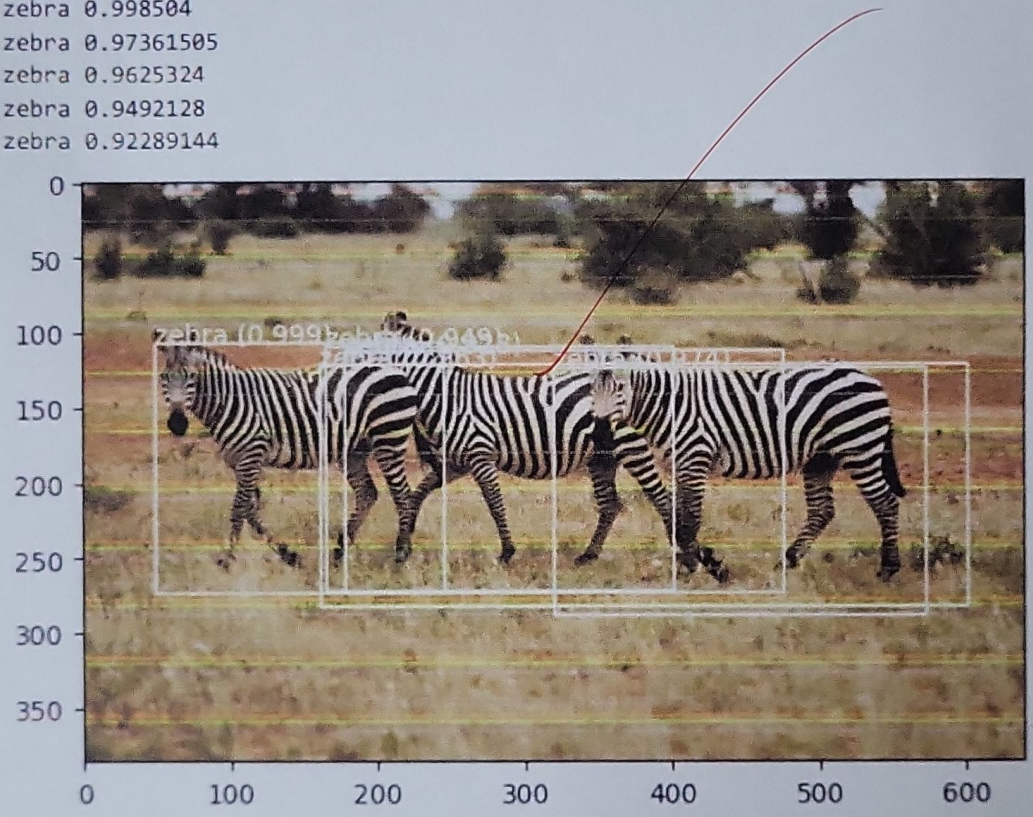
```
<matplotlib.image.AxesImage at 0x1f3e1dc9d88>
```


```
1/1 [==============================] - 2s 2s/step
[(1, 13, 13, 255), (1, 26, 26, 255), (1, 52, 52, 255)]

10647

car 0.9799808
car 0.90150243
car 0.9822434
car 0.82281214
car 0.9342798
car 0.93157935
car 0.98184437
car 0.9982241
car 0.981548
car 0.9050933
car 0.76686305
car 0.98158294
car 0.92317057
car 0.65974027
car 0.9845722
car 0.98843783
car 0.8808287
```


```
car 0.9982241 status: kept
car 0.98843783 status: kept
car 0.9845722 status: removed
car 0.9822434 status: kept
car 0.98184437 status: removed
car 0.98158294 status: kept
car 0.981548 status: removed
car 0.9799808 status: removed
car 0.9342798 status: kept
car 0.93157935 status: removed
car 0.92317057 status: removed
car 0.9050933 status: removed
car 0.90150243 status: removed
car 0.8808287 status: removed
car 0.82281214 status: removed
car 0.78529125 status: kept
car 0.76686305 status: kept
car 0.65974027 status: removed
car 0.6020881 status: removed
traffic light 0.964606 status: kept
traffic light 0.87137073 status: kept
traffic light 0.8328069 status: kept
traffic light 0.77623147 status: kept
traffic light 0.75971967 status: kept
traffic light 0.70200676 status: removed
traffic light 0.618828 status: removed
```

car 0.9982241
car 0.98843783
car 0.9822434
car 0.98158294
car 0.9342798
car 0.78529125
car 0.76686305
traffic light 0.964606
traffic light 0.87137073
traffic light 0.8328069
traffic light 0.77623147
traffic light 0.75971967

```
1/1 [==============================] - 0s 296ms/step
zebra 0.998504 status: kept
zebra 0.9980972 status: removed
zebra 0.99749166 status: removed
zebra 0.99720335 status: removed
zebra 0.97361505 status: kept
zebra 0.9625324 status: kept
zebra 0.9492128 status: kept
zebra 0.9479591 status: removed
zebra 0.92289144 status: kept
zebra 0.8875617 status: removed
zebra 0.7587733 status: removed
zebra 0.71294904 status: removed
zebra 0.6686954 status: removed
zebra 0.998504
zebra 0.97361505
zebra 0.9625324
zebra 0.9492128
zebra 0.92289144
```



```
1/1 [==============================] - 0s 285ms/step
horse 0.9981355 status: kept
horse 0.9928219 status: removed
horse 0.9498802 status: removed
horse 0.8879705 status: removed
horse 0.79800767 status: removed
horse 0.7887686 status: removed
horse 0.6308995 status: removed
horse 0.9981355
```

O/P Verified
29/9/25.

Result:

Object detection model has successfully been built with YOLO3 using Tensorflow/Keras.
29/9/25

Ex No: 9 BUILD GENERATIVE ADVERSARIAL NEURAL NETWORK

Aim:

To build a generative adversarial neural network using Keras/TensorFlow.

Procedure:

1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a simple neural network model using Keras/TensorFlow.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

Useful Learning Resources:

1. https://pyimagesearch.com/2020/11/16/gans-with-keras-and-tensorflow/

2.https://www.analyticsvidhya.com/blog/2021/06/a-detailed-explanation-of-gan-with-implementation-using-tensorflow-and-keras/

9.

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.model import Adam
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt

iris = load_iris()
X_train = iris.data

def build_generator():
    model = Sequential()
    model.add(Dense(128, input_shape=(100,), activation
                                        ='relu'))
    model.add(Dense(4, activation='linear'))
    return model

def build_gan(generator, discriminator):
    discriminator.trainable = False
    model = Sequential()
    model.add(generator)
    model.add(discriminator
    return model

generator = build_generator()
discriminator = build_discriminator()
gan = build_gan(generator, discriminator)
generator.compile(loss='mean_squared_error',
                  optimizer=Adam(0.0002, 0.5))
```

```
epochs = 200
batch_size = 16

d_loss_real = discriminator.train_on_batch
                    (real_samples, real_labels)

d_loss_fake = discriminator.train_on_batch
                    (fake_samples, fake_labels)

noise = np.random.normal(0, 1, (batch_size, 100))
g_loss = gan.train_on_batch(noise, real_labels)

plt.figure(figsize=(12,8))
plot_idx = 1

  for i in range(4):
    for j in range(i+1, 4):
      plt.subplot(2,3,plot_idx)
      plt.scatter(x_train[:,i], x_train[:,j], label=
                        'Real Data', c='blue',
                        marker='o', s=30)
      plt.xlabel(f'Feature {i+1}')
      plt.ylabel(f'Feature {j+1}')
      plt.legend()
      plt_idx+=1

plt.tight_layout()
plt.show().
```

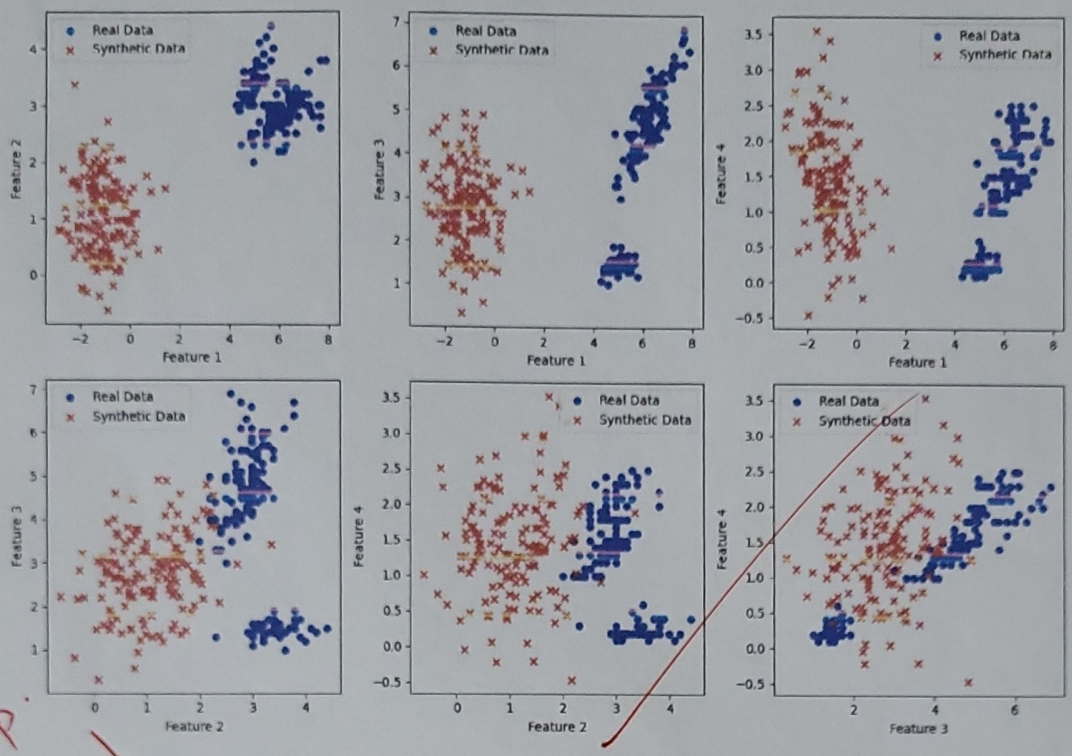
Epoch 1/200 | Discriminator Loss: 0.5367496609687805 | Generator Loss: 0.8079249858856201
Epoch 2/200 | Discriminator Loss: 0.5273558497428894 | Generator Loss: 0.825336217880249
Epoch 3/200 | Discriminator Loss: 0.5393378436565399 | Generator Loss: 0.7887856364250183
Epoch 4/200 | Discriminator Loss: 0.5399681031703949 | Generator Loss: 0.796552836894989
Epoch 5/200 | Discriminator Loss: 0.5556007325649261 | Generator Loss: 0.8075708746910095
Epoch 6/200 | Discriminator Loss: 0.5442723035812378 | Generator Loss: 0.8555177450180054
Epoch 7/200 | Discriminator Loss: 0.5600491166114807 | Generator Loss: 0.7774984836578369
Epoch 9/200 | Discriminator Loss: 0.5588854253292084 | Generator Loss: 0.846459150314331
Epoch 10/200 | Discriminator Loss: 0.5351185202598572 | Generator Loss: 0.866773247718811
Epoch 11/200 | Discriminator Loss: 0.5646434724330902 | Generator Loss: 0.8025287389755249
Epoch 12/200 | Discriminator Loss: 0.5537577271461487 | Generator Loss: 0.781479001045227
Epoch 13/200 | Discriminator Loss: 0.5329684317111969 | Generator Loss: 0.8493124842643738
Epoch 14/200 | Discriminator Loss: 0.5579888820648193 | Generator Loss: 0.7836759686470032
Epoch 15/200 | Discriminator Loss: 0.5424736142158508 | Generator Loss: 0.8189789056777954
Epoch 17/200 | Discriminator Loss: 0.5760356783866882 | Generator Loss: 0.8018962144851685
Epoch 18/200 | Discriminator Loss: 0.5639258325099945 | Generator Loss: 0.8068743944168091
Epoch 19/200 | Discriminator Loss: 0.5675747394561768 | Generator Loss: 0.7301404476165771
Epoch 20/200 | Discriminator Loss: 0.5899735987186432 | Generator Loss: 0.7989237904548645
Epoch 21/200 | Discriminator Loss: 0.5722223818302155 | Generator Loss: 0.7802469730377197
Epoch 22/200 | Discriminator Loss: 0.5774404406547546 | Generator Loss: 0.7919635772705078
Epoch 23/200 | Discriminator Loss: 0.5490640699863434 | Generator Loss: 0.8464027643203735
Epoch 24/200 | Discriminator Loss: 0.5382504463195801 | Generator Loss: 0.7698981761932373
Epoch 26/200 | Discriminator Loss: 0.5500295758247375 | Generator Loss: 0.7546350955963135
Epoch 27/200 | Discriminator Loss: 0.5844375491142273 | Generator Loss: 0.7997597455978394
Epoch 28/200 | Discriminator Loss: 0.5803672671318054 | Generator Loss: 0.7773357629776001
Epoch 29/200 | Discriminator Loss: 0.548518717288971 | Generator Loss: 0.7321611642837524
Epoch 30/200 | Discriminator Loss: 0.584658682346344 | Generator Loss: 0.7975304126739502
Epoch 31/200 | Discriminator Loss: 0.5977251529693604 | Generator Loss: 0.7843732833862305
Epoch 32/200 | Discriminator Loss: 0.5775752663612366 | Generator Loss: 0.7508641481399536
Epoch 34/200 | Discriminator Loss: 0.5939444899559021 | Generator Loss: 0.7141955494880676
Epoch 35/200 | Discriminator Loss: 0.5534800589084625 | Generator Loss: 0.7329733371734619
Epoch 36/200 | Discriminator Loss: 0.5729694366455078 | Generator Loss: 0.7628708481788635
Epoch 37/200 | Discriminator Loss: 0.5690068304538727 | Generator Loss: 0.7406902313232422
Epoch 38/200 | Discriminator Loss: 0.5793368816375732 | Generator Loss: 0.7083196640014648
Epoch 39/200 | Discriminator Loss: 0.5739973783493042 | Generator Loss: 0.7296966314315796
Epoch 40/200 | Discriminator Loss: 0.5791043937206268 | Generator Loss: 0.7200328707695007
Epoch 42/200 | Discriminator Loss: 0.6019871830940247 | Generator Loss: 0.7177163362503052
Epoch 43/200 | Discriminator Loss: 0.577749028801918 | Generator Loss: 0.7343310117721558
Epoch 44/200 | Discriminator Loss: 0.5850207060575485 | Generator Loss: 0.7176611423492432
Epoch 45/200 | Discriminator Loss: 0.5907089114189148 | Generator Loss: 0.7254678606987
Epoch 46/200 | Discriminator Loss: 0.5936924517154694 | Generator Loss: 0.8064104318618774
Epoch 47/200 | Discriminator Loss: 0.5817234516143799 | Generator Loss: 0.6598802208900452
Epoch 48/200 | Discriminator Loss: 0.5784140229225159 | Generator Loss: 0.7024240493774414
Epoch 50/200 | Discriminator Loss: 0.5743369609117508 | Generator Loss: 0.706048309803009
Epoch 51/200 | Discriminator Loss: 0.595258891582489 | Generator Loss: 0.6617811322212219
Epoch 52/200 | Discriminator Loss: 0.6189641654491425 | Generator Loss: 0.6768359541893005
Epoch 53/200 | Discriminator Loss: 0.5956382155418396 | Generator Loss: 0.7037990689277649
Epoch 54/200 | Discriminator Loss: 0.5758906304836273 | Generator Loss: 0.6549841165542603
Epoch 55/200 | Discriminator Loss: 0.5934228301048279 | Generator Loss: 0.6768755316734314
Epoch 56/200 | Discriminator Loss: 0.6097339987754822 | Generator Loss: 0.666079044342041
Epoch 57/200 | Discriminator Loss: 0.6268243491649628 | Generator Loss: 0.7235430479049683
Epoch 59/200 | Discriminator Loss: 0.614975063123703 | Generator Loss: 0.6703068820724487
Epoch 60/200 | Discriminator Loss: 0.6177686154842377 | Generator Loss: 0.6552383899688721
Epoch 61/200 | Discriminator Loss: 0.5769155770540237 | Generator Loss: 0.6605809926986694
Epoch 62/200 | Discriminator Loss: 0.5967360138893127 | Generator Loss: 0.6605456471443176
Epoch 63/200 | Discriminator Loss: 0.6020719707012177 | Generator Loss: 0.6851934790611267
Epoch 64/200 | Discriminator Loss: 0.589726984500885 | Generator Loss: 0.6252405643463135
Epoch 65/200 | Discriminator Loss: 0.6127613633871078 | Generator Loss: 0.6218724250793457
Epoch 67/200 | Discriminator Loss: 0.6331568360328674 | Generator Loss: 0.6661688089370728
Epoch 68/200 | Discriminator Loss: 0.6119307279586792 | Generator Loss: 0.6597158908843994
Epoch 69/200 | Discriminator Loss: 0.6182061100332489 | Generator Loss: 0.6984419822692871

Epoch 70/200 | Discriminator Loss: 0.6830051396751709 | Generator Loss: 0.6607615987723389
Epoch 71/200 | Discriminator Loss: 0.650241881608963 | Generator Loss: 0.6442438364028931
Epoch 72/200 | Discriminator Loss: 0.6112033724784851 | Generator Loss: 0.6116329431533813
Epoch 73/200 | Discriminator Loss: 0.6188270449638367 | Generator Loss: 0.6056983470916748
Epoch 74/200 | Discriminator Loss: 0.642275333404541 | Generator Loss: 0.64552903175354
Epoch 75/200 | Discriminator Loss: 0.6342885494232178 | Generator Loss: 0.6651420593261719
Epoch 76/200 | Discriminator Loss: 0.6483060121536255 | Generator Loss: 0.6417959928512573
Epoch 77/200 | Discriminator Loss: 0.6233980655670166 | Generator Loss: 0.66825270652771
Epoch 78/200 | Discriminator Loss: 0.6010263413190842 | Generator Loss: 0.6668068170547485
Epoch 79/200 | Discriminator Loss: 0.6139638721942902 | Generator Loss: 0.6227089166641235
Epoch 80/200 | Discriminator Loss: 0.6367082893848419 | Generator Loss: 0.6008510589599609
Epoch 81/200 | Discriminator Loss: 0.6355765014886856 | Generator Loss: 0.6111114025115967
Epoch 82/200 | Discriminator Loss: 0.6465215682983398 | Generator Loss: 0.598038911819458
Epoch 83/200 | Discriminator Loss: 0.6360856592655182 | Generator Loss: 0.625205397605896
Epoch 84/200 | Discriminator Loss: 0.638536810874939 | Generator Loss: 0.6454572081565857
Epoch 85/200 | Discriminator Loss: 0.6163221597671509 | Generator Loss: 0.5766980051994324
Epoch 86/200 | Discriminator Loss: 0.6716514527797699 | Generator Loss: 0.5951347947120667
Epoch 87/200 | Discriminator Loss: 0.6493960022926331 | Generator Loss: 0.6074239015579224
Epoch 88/200 | Discriminator Loss: 0.656065970659256 | Generator Loss: 0.5834223031997681
Epoch 89/200 | Discriminator Loss: 0.6393578052520752 | Generator Loss: 0.637916088104248
Epoch 90/200 | Discriminator Loss: 0.6531704366207123 | Generator Loss: 0.5887624025344849
Epoch 91/200 | Discriminator Loss: 0.6536164283752441 | Generator Loss: 0.5976898670196533

Epoch 92/200 | Discriminator Loss: 0.6763930022716522 | Generator Loss: 0.6128095984458923
Epoch 93/200 | Discriminator Loss: 0.6402183175086975 | Generator Loss: 0.6389085054397583
Epoch 94/200 | Discriminator Loss: 0.6511092185974121 | Generator Loss: 0.6450672149658203
Epoch 95/200 | Discriminator Loss: 0.6386714279651642 | Generator Loss: 0.6253257989883423
Epoch 96/200 | Discriminator Loss: 0.6918444633483887 | Generator Loss: 0.6162148714065552
Epoch 97/200 | Discriminator Loss: 0.6467417180538177 | Generator Loss: 0.6068918704986572
Epoch 98/200 | Discriminator Loss: 0.6695604622364044 | Generator Loss: 0.5682520866394043
Epoch 99/200 | Discriminator Loss: 0.6725573837757111 | Generator Loss: 0.61757892370224
Epoch 100/200 | Discriminator Loss: 0.6413220763206482 | Generator Loss: 0.5490227937698364
Epoch 101/200 | Discriminator Loss: 0.6490455865859985 | Generator Loss: 0.5907291769981384
Epoch 102/200 | Discriminator Loss: 0.6883095123767853 | Generator Loss: 0.5948603045537883
Epoch 103/200 | Discriminator Loss: 0.645122617483139 | Generator Loss: 0.5800871849060059
Epoch 104/200 | Discriminator Loss: 0.6797735095024109 | Generator Loss: 0.5647033452987671
Epoch 105/200 | Discriminator Loss: 0.7075727581977844 | Generator Loss: 0.5164201259613037
Epoch 106/200 | Discriminator Loss: 0.6633673012256622 | Generator Loss: 0.5311208963394165
Epoch 107/200 | Discriminator Loss: 0.6969761252403259 | Generator Loss: 0.5315980315208435
Epoch 108/200 | Discriminator Loss: 0.6720565855503082 | Generator Loss: 0.539708137512207
Epoch 109/200 | Discriminator Loss: 0.6895166337490082 | Generator Loss: 0.5750694274902344
Epoch 110/200 | Discriminator Loss: 0.7023615837097168 | Generator Loss: 0.5173653874125671
Epoch 111/200 | Discriminator Loss: 0.6828137636184692 | Generator Loss: 0.5732543468475342
Epoch 112/200 | Discriminator Loss: 0.683556318283081 | Generator Loss: 0.6048802733421326
Epoch 113/200 | Discriminator Loss: 0.7215807437896729 | Generator Loss: 0.5765171051025391
Epoch 114/200 | Discriminator Loss: 0.6658097803592682 | Generator Loss: 0.5914000272750854
Epoch 115/200 | Discriminator Loss: 0.739637166261673 | Generator Loss: 0.5193003416061401
Epoch 116/200 | Discriminator Loss: 0.7233046293258667 | Generator Loss: 0.573081374168396
Epoch 117/200 | Discriminator Loss: 0.6887190937995911 | Generator Loss: 0.5282692909240723
Epoch 118/200 | Discriminator Loss: 0.6966548264026642 | Generator Loss: 0.5173132419586182
Epoch 119/200 | Discriminator Loss: 0.7160505950450897 | Generator Loss: 0.544549286365509
Epoch 120/200 | Discriminator Loss: 0.685859739780426 | Generator Loss: 0.5265844464302063
Epoch 121/200 | Discriminator Loss: 0.682758629322052 | Generator Loss: 0.5420533418655396
Epoch 122/200 | Discriminator Loss: 0.683195948600769 | Generator Loss: 0.45624735951423645
Epoch 123/200 | Discriminator Loss: 0.7026805877685547 | Generator Loss: 0.4897909462451935
Epoch 124/200 | Discriminator Loss: 0.6831173896789551 | Generator Loss: 0.5245143175125122
Epoch 125/200 | Discriminator Loss: 0.7281747460365295 | Generator Loss: 0.5076379776000977
Epoch 126/200 | Discriminator Loss: 0.7277912497520447 | Generator Loss: 0.5385361909866333
Epoch 127/200 | Discriminator Loss: 0.7040023803710938 | Generator Loss: 0.5361969470977783
Epoch 128/200 | Discriminator Loss: 0.715343177318573 | Generator Loss: 0.542757213115692 1
Epoch 129/200 | Discriminator Loss: 0.734138548374176 | Generator Loss: 0.5352092981338501
Epoch 130/200 | Discriminator Loss: 0.6976486593484879 | Generator Loss: 0.47202515602111816
Epoch 131/200 | Discriminator Loss: 0.6969233006238937 | Generator Loss: 0.548151314258575 4
Epoch 132/200 | Discriminator Loss: 0.7120088338851929 | Generator Loss: 0.4925209581851959
Epoch 133/200 | Discriminator Loss: 0.7257662117481232 | Generator Loss: 0.4979208707809448
Epoch 134/200 | Discriminator Loss: 0.7236302495002747 | Generator Loss: 0.5206350684165955
Epoch 135/200 | Discriminator Loss: 0.6859756709145142 | Generator Loss: 0.5057195425913669
Epoch 136/200 | Discriminator Loss: 0.7222448736429214 | Generator Loss: 0.48126089572906494
Epoch 137/200 | Discriminator Loss: 0.7202684283256531 | Generator Loss: 0.5272859334945679
Epoch 138/200 | Discriminator Loss: 0.7080051302909851 | Generator Loss: 0.5033074617385864

Epoch 140/200 | Discriminator Loss: 0.7791062891483307 | Generator Loss: 0.48185691237449646
Epoch 141/200 | Discriminator Loss: 0.7155601680278778 | Generator Loss: 0.4770204722881317
Epoch 142/200 | Discriminator Loss: 0.7104989439249039 | Generator Loss: 0.541405200958252
Epoch 143/200 | Discriminator Loss: 0.7239029407501221 | Generator Loss: 0.5137744545936584
Epoch 144/200 | Discriminator Loss: 0.747321143746376 | Generator Loss: 0.4493764340877533
Epoch 145/200 | Discriminator Loss: 0.7685433924198151 | Generator Loss: 0.4587600827217102
Epoch 146/200 | Discriminator Loss: 0.7512906342744827 | Generator Loss: 0.5152004361152649
Epoch 147/200 | Discriminator Loss: 0.7518601119518328 | Generator Loss: 0.4742632806301117
Epoch 148/200 | Discriminator Loss: 0.7182693779468536 | Generator Loss: 0.4633033275604248
Epoch 149/200 | Discriminator Loss: 0.7651214003562927 | Generator Loss: 0.47064152359962463
Epoch 150/200 | Discriminator Loss: 0.7588287591934204 | Generator Loss: 0.469757080078125
Epoch 151/200 | Discriminator Loss: 0.7716760635375977 | Generator Loss: 0.4492041766643524
Epoch 152/200 | Discriminator Loss: 0.7958682775497437 | Generator Loss: 0.42562538385391235
Epoch 153/200 | Discriminator Loss: 0.7820593416690826 | Generator Loss: 0.4476546049118042
Epoch 154/200 | Discriminator Loss: 0.7890766263008118 | Generator Loss: 0.49128490686416626
Epoch 155/200 | Discriminator Loss: 0.7252300083637238 | Generator Loss: 0.5053080916404724
Epoch 156/200 | Discriminator Loss: 0.7600408792495728 | Generator Loss: 0.4953346252441406
Epoch 157/200 | Discriminator Loss: 0.7330844402313232 | Generator Loss: 0.45828336477279663
Epoch 158/200 | Discriminator Loss: 0.7615276575088501 | Generator Loss: 0.46765241026878357
Epoch 159/200 | Discriminator Loss: 0.798762708902359 | Generator Loss: 0.45628267526626587
Epoch 160/200 | Discriminator Loss: 0.7629557251930237 | Generator Loss: 0.47066861391067505
Epoch 161/200 | Discriminator Loss: 0.7729970067739487 | Generator Loss: 0.4746553897857666
Epoch 162/200 | Discriminator Loss: 0.7489473521709442 | Generator Loss: 0.41200196743011475
Epoch 163/200 | Discriminator Loss: 0.7956132888793945 | Generator Loss: 0.4565112292766571
Epoch 164/200 | Discriminator Loss: 0.7603410184383392 | Generator Loss: 0.4523635208606672
Epoch 165/200 | Discriminator Loss: 0.7581019401550293 | Generator Loss: 0.47844061255455017
Epoch 166/200 | Discriminator Loss: 0.787080705165863 | Generator Loss: 0.4474506676197052
Epoch 167/200 | Discriminator Loss: 0.7879129946231842 | Generator Loss: 0.4220748543739319
Epoch 168/200 | Discriminator Loss: 0.7647700905799866 | Generator Loss: 0.44674116373062134
Epoch 169/200 | Discriminator Loss: 0.7506582140922546 | Generator Loss: 0.4395178556442261
Epoch 170/200 | Discriminator Loss: 0.772972583770752 | Generator Loss: 0.44381022453308105
Epoch 171/200 | Discriminator Loss: 0.8217633962631226 | Generator Loss: 0.4642025828361511
Epoch 172/200 | Discriminator Loss: 0.7953487729835849 | Generator Loss: 0.4780427002429962
Epoch 173/200 | Discriminator Loss: 0.7900170087814331 | Generator Loss: 0.4631214439868927
Epoch 174/200 | Discriminator Loss: 0.7621420919895172 | Generator Loss: 0.420756995677948
Epoch 175/200 | Discriminator Loss: 0.8026988804340363 | Generator Loss: 0.41185203194618225
Epoch 176/200 | Discriminator Loss: 0.785543292760849 | Generator Loss: 0.385200560092926
Epoch 177/200 | Discriminator Loss: 0.7796034812927246 | Generator Loss: 0.39292874932289124
Epoch 178/200 | Discriminator Loss: 0.7672508060932159 | Generator Loss: 0.4299434423446655
Epoch 179/200 | Discriminator Loss: 0.8126103281974792 | Generator Loss: 0.4205746650695801
Epoch 180/200 | Discriminator Loss: 0.7706550955772401 | Generator Loss: 0.4235925078392029
Epoch 181/200 | Discriminator Loss: 0.7893175482749939 | Generator Loss: 0.40925073623657227
Epoch 182/200 | Discriminator Loss: 0.8122552037239075 | Generator Loss: 0.4211147427558899

Epoch 183/200 | Discriminator Loss: 0.8231731653213501 | Generator Loss: 0.3744174540042877
Epoch 184/200 | Discriminator Loss: 0.8017941117286682 | Generator Loss: 0.4059006869792938
Epoch 185/200 | Discriminator Loss: 0.8379355669021606 | Generator Loss: 0.37071555852890015
Epoch 186/200 | Discriminator Loss: 0.8176586627960205 | Generator Loss: 0.4078059196472168
Epoch 187/200 | Discriminator Loss: 0.7991514503955841 | Generator Loss: 0.4491428732872009
Epoch 188/200 | Discriminator Loss: 0.8243433833122253 | Generator Loss: 0.4152688980102539
Epoch 189/200 | Discriminator Loss: 0.7871273756027222 | Generator Loss: 0.39617204666137695
Epoch 190/200 | Discriminator Loss: 0.8173457682132721 | Generator Loss: 0.3896409273147583
Epoch 191/200 | Discriminator Loss: 0.827910304069519 | Generator Loss: 0.3743140697479248
Epoch 192/200 | Discriminator Loss: 0.7939517498016357 | Generator Loss: 0.39283424615859985
Epoch 193/200 | Discriminator Loss: 0.8027375340461731 | Generator Loss: 0.4169652462005615
Epoch 194/200 | Discriminator Loss: 0.8082454204559326 | Generator Loss: 0.4213721752166748
Epoch 195/200 | Discriminator Loss: 0.8045962303876877 | Generator Loss: 0.40519559383392334
Epoch 196/200 | Discriminator Loss: 0.8121471405029297 | Generator Loss: 0.3701239228248596
Epoch 197/200 | Discriminator Loss: 0.8393056690692902 | Generator Loss: 0.38568806648254395
Epoch 198/200 | Discriminator Loss: 0.8243976533412933 | Generator Loss: 0.3877955079078674
Epoch 199/200 | Discriminator Loss: 0.7965860068798065 | Generator Loss: 0.35657164454460144



O/P.

Verified

6/10/25

Result:

Adversial neural network using Tensorflow / Keras implemented successfully.

Ex No: 10 **MINI PROJECT – CNN OR RNN BASED APPLICATION**

Aim:

To develop an application that is based on convolutional neural network or recurrent neural network in Keras/TensorFlow.

Instructions:

1. Student has to choose one of the projects listed in the below section with a team of maximum two members.
2. Apart from these topics, if a team has innovative ideas to do implementation, then they can discuss with their faculty in-charge and get approval to do the same.
3. After implementation, a Mini Project report needs to be prepared and signed by HoD and the faculty in-charge.

Mini Project Titles;

1. Plant Disease Detection using Deep Learning
2. Fake News Detection using Deep Learning
3. Breast Cancer Detection using Deep Learning
4. Chatbot using Recurrent Neural Network
5. Drowsy Driver Detection using Deep Learning
6. A Review of Liver Patient Analysis Methods using Deep Learning
7. Deep Learning based Thyroid Disease Classification.
8. Music Genre Classification System
9. Dog Breed Identification using Deep Learning
10. Human Face Detection using Deep Learning
11. Automated Attendance monitoring using Deep Learning
12. Skin Cancer Detection using Deep Learning.

LAB VIVA QUESTIONS

1. What is Deep Learning, and how does it differ from traditional machine learning?
2. Explain the concept of neural networks and their role in Deep Learning.
3. What are the main components of a typical neural network?
4. Describe the backpropagation algorithm and its importance in training neural networks.
5. How do you choose the appropriate activation function for a neural network?
6. Discuss the vanishing gradient problem and its impact on Deep Learning.
7. What are some common regularization techniques used in Deep Learning, and how do they prevent overfitting?
8. Explain the concept of convolutional neural networks (CNNs) and their applications.
9. How does pooling (e.g., max pooling) work in CNNs, and what is its purpose?
10. What is data augmentation, and why is it used in CNNs?
11. Discuss the challenges and solutions when working with small datasets in Deep Learning.
12. Describe the architecture and advantages of recurrent neural networks (RNNs).
13. Explain the concept of Long Short-Term Memory (LSTM) cells in RNNs.
14. How do you handle the vanishing gradient problem in RNNs?
15. What is attention mechanism, and how does it improve the performance of sequence-to-sequence models?
16. Discuss the concept of transfer learning and its applications in Deep Learning.
17. How can you fine-tune a pre-trained neural network for a specific task?
18. Explain the differences between supervised, unsupervised, and reinforcement learning in the context of Deep Learning.
19. What are generative adversarial networks (GANs), and how do they work?
20. Describe the main components of a GAN architecture (generator and discriminator).
21. How can GANs be used for image synthesis and style transfer?
22. Discuss the challenges and potential solutions for training GANs.
23. What is the concept of autoencoders, and how are they used in unsupervised learning?
24. Explain the process of dimensionality reduction using autoencoders.
25. How can you use autoencoders for denoising or anomaly detection?
26. Discuss the concept of reinforcement learning and its use in training agents to perform tasks.
27. Explain the role of the reward function in reinforcement learning algorithms.
28. What is Q-learning, and how does it work in reinforcement learning?

29. How can you handle the exploration-exploitation trade-off in reinforcement learning?
30. Describe the challenges and approaches to dealing with high-dimensional action spaces in reinforcement learning.
31. Explain the concept of policy gradients and their advantages in certain reinforcement learning scenarios.
32. Discuss the concept of natural language processing (NLP) and its relation to Deep Learning.
33. How are recurrent neural networks used in natural language processing tasks like language modeling?
34. What is word embedding, and how does it improve the representation of words in NLP models?
35. Explain the architecture and applications of transformer models in NLP.
36. How does the attention mechanism work in transformer-based models like BERT?
37. Discuss the challenges of training large-scale language models and potential solutions.
38. Explain the concept of word2vec and its applications in NLP.
39. What are the differences between CBOW (Continuous Bag of Words) and Skip-gram word2vec models?
40. Describe the process of training a word2vec model.
41. How can word embeddings be visualized and evaluated?
42. Discuss the concept of auto-regressive models in natural language processing.
43. What is beam search, and how does it improve the output generation in sequence-to-sequence models?
44. Explain the concept of self-attention and its use in transformer-based models for NLP.
45. How can you apply transfer learning to pre-trained language models like GPT-3?
46. Discuss the challenges and solutions when working with noisy or unstructured text data in NLP.
47. Explain the concept of style transfer in NLP and its applications.
48. How can you use Deep Learning models for sentiment analysis on text data?
49. Discuss the potential ethical considerations and biases in Deep Learning models for NLP.
50. Describe the process of fine-tuning a pre-trained NLP model for a specific language-related task.