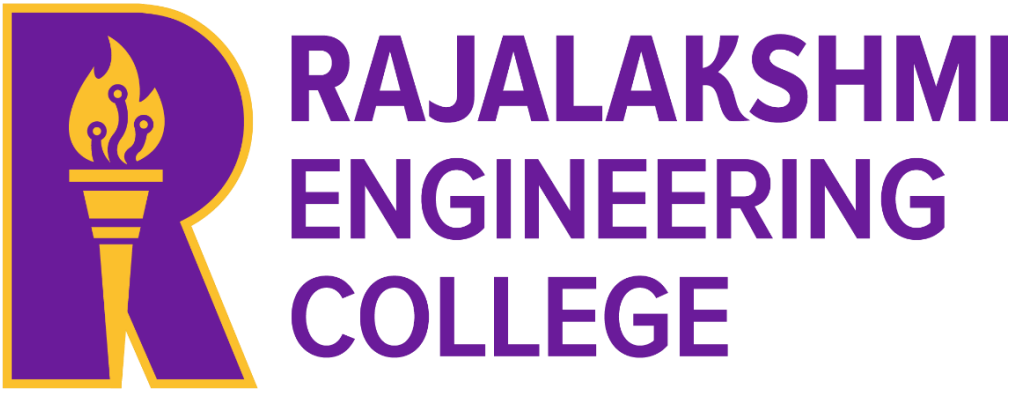
RAJALAKSHMI ENGINEERING COLLEGE

(An Autonomous Institution)

RAJALAKSHMI NAGAR, THANDALAM- 602 105



CS19P18 - DEEP LEARNING CONCEPTS

LABORATORY RECORD NOTEBOOK

NAME: S LOKESHWAR

YEAR/SEMESTER: IV / VII

BRANCH: COMPUTER SCIENCE AND ENGINEERING

REGISTER NO: 220701146

COLLEGE ROLL NO: 2116220701146

ACADEMIC YEAR: 2025-2026

RAJALAKSHMI ENGINEERING COLLEGE

(An Autonomous Institution)

RAJALAKSHMI NAGAR, THANDALAM- 602 105

BONAFIDE CERTIFICATE

NAME: S LOKESHWAR **BRANCH/SECTION:** COMPUTER SCIENCE AND ENGINEERING **ACADEMIC YEAR:** 2025 -2026
SEMESTER: SEVENTH

REGISTER NO:	**220701146**

Certified that this is a Bonafide record of work done by the above student in the **CS19P18 - DEEP LEARNING CONCEPTS** during the year 2025 - 2026

Signature of Faculty In-charge

Submitted for the Practical Examination Held on: ……..

Internal Examiner **External Examiner**

INDEX

EX.NO	DATE	NAME OF THE EXPERIMENT	PAGE NO	STAFF SIGN
1	14/07/2025	Create a neural network to recognize handwritten digits using MNIST dataset	5	
2	21/07/2025	Build a Convolutional Neural Network with Keras/TensorFlow	8	
3	28/07/2025	Image Classification on CIFAR-10 Dataset using CNN	11	
4	01/08/2025	Transfer learning with CNN and Visualization	13	
5	27/08/2025	Build a Recurrent Neural Network using Keras/Tensorflow	17	
6	15/09/2025	Sentiment Classification of Text using RNN	19	
7	22/09/2025	Build autoencoders with keras/tensorflow	21	
8	06/10/2025	Object detection with yolo3	24	
9	06/10/2025	Build GAN with Keras/TensorFlow	27	
10	14/10/2025	Mini Project	31	

INSTALLATION AND CONFIGURATION OF TENSORFLOW

Aim:

To install and configure TensorFlow in the 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)

EX NO: 1 CREATE A NEURAL NETWORK TO RECOGNIZE HANDWRITTEN

DATE:14/07/2025 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.

Code:

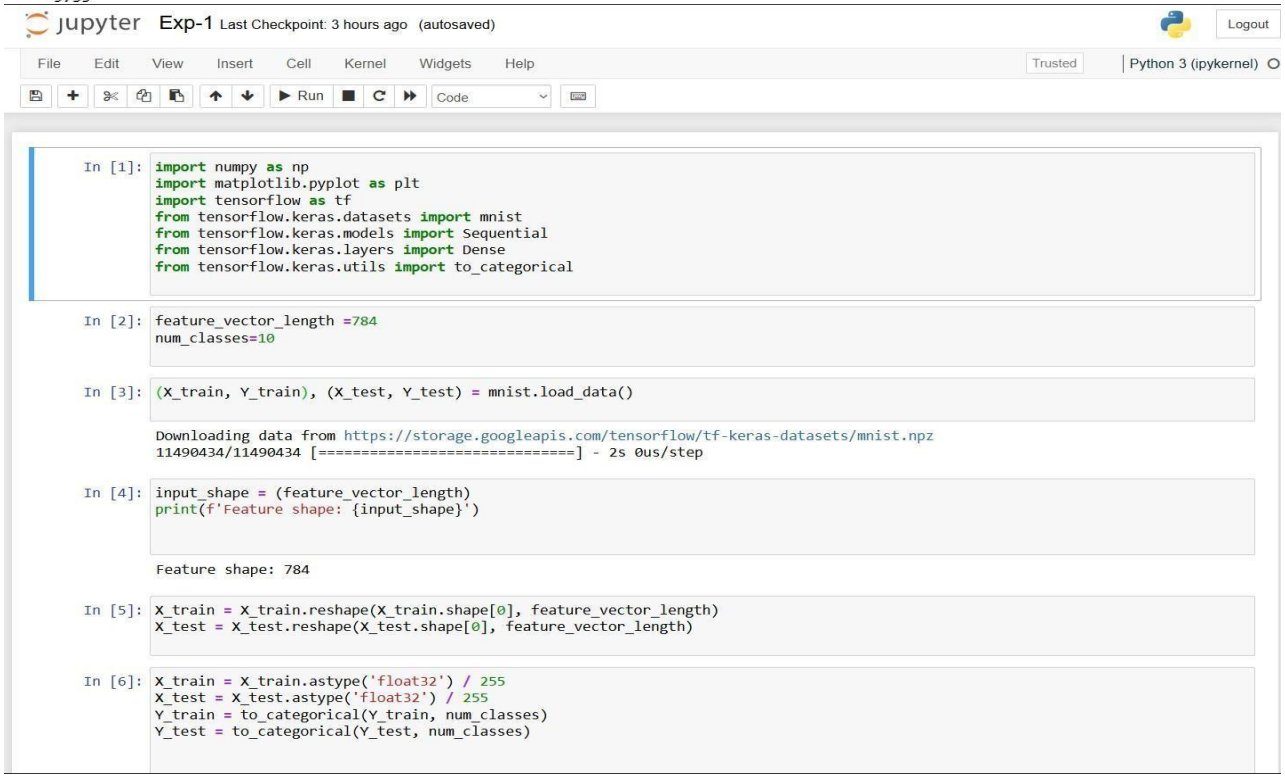
```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# Generate a synthetic dataset
X, y = make_classification(n_samples=1000, n_features=20, random_state=42)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

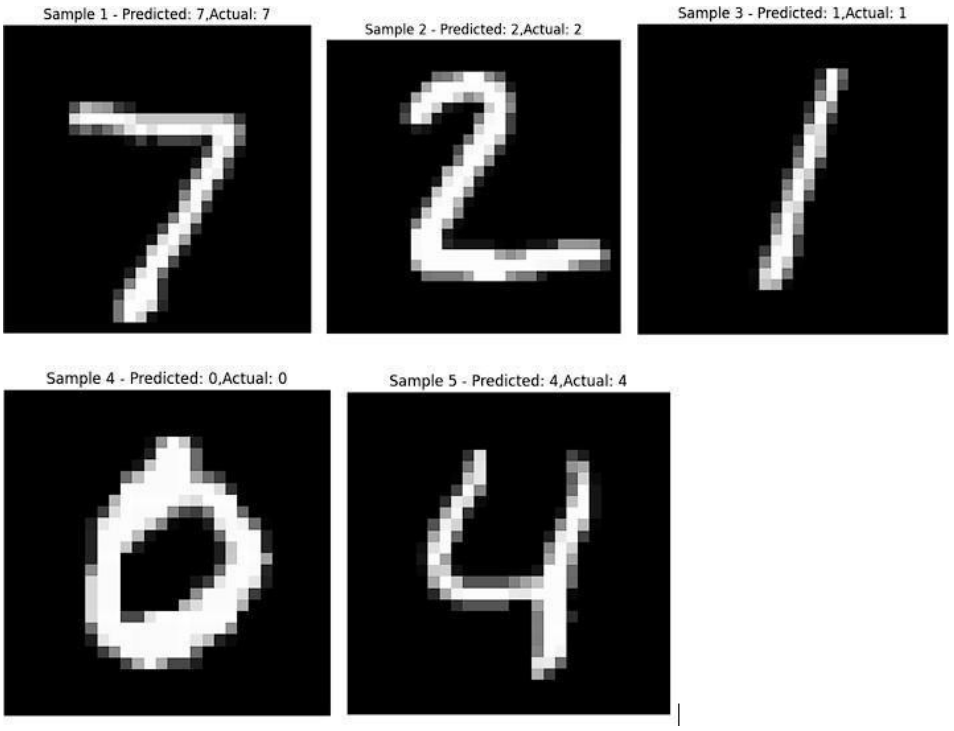
```
# Define the model
model = keras.Sequential([
    keras.layers.Input(shape=(X_train.shape[1],)),  # Input layer
    keras.layers.Dense(64, activation='relu'),      # Hidden layer
    keras.layers.Dense(1, activation='sigmoid')     # Output layer
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.1)

# Evaluate the model
y_pred = model.predict(X_test)
y_pred_classes = (y_pred > 0.5).astype(int)

# Calculate accuracy and loss
accuracy = accuracy_score(y_test, y_pred_classes)
test_loss, test_acc = model.evaluate(X_test, y_test)

print(f"\nTest Accuracy: {accuracy * 100:.2f}%")
print(f"Test Loss: {test_loss:.4f}")

# Plot training performance
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
```

```
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.tight_layout()
plt.show()
```

Output:

```
Epoch 1/10
192/192 [==============================] - 5s 17ms/step - loss: 0.3739 - accuracy: 0.8950 - val_loss: 0.1801 - val_accuracy: 0.
9480
Epoch 2/10
192/192 [==============================] - 3s 14ms/step - loss: 0.1492 - accuracy: 0.9562 - val_loss: 0.1261 - val_accuracy: 0.
9635
Epoch 3/10
192/192 [==============================] - 2s 13ms/step - loss: 0.0980 - accuracy: 0.9714 - val_loss: 0.1129 - val_accuracy: 0.
9676
Epoch 4/10
192/192 [==============================] - 2s 11ms/step - loss: 0.0711 - accuracy: 0.9795 - val_loss: 0.0962 - val_accuracy: 0.
9709
Epoch 5/10
192/192 [==============================] - 2s 10ms/step - loss: 0.0543 - accuracy: 0.9844 - val_loss: 0.0914 - val_accuracy: 0.
9725
Epoch 6/10
192/192 [==============================] - 2s 11ms/step - loss: 0.0402 - accuracy: 0.9888 - val_loss: 0.0866 - val_accuracy: 0.
9737
Epoch 7/10
192/192 [==============================] - 2s 12ms/step - loss: 0.0301 - accuracy: 0.9920 - val_loss: 0.0871 - val_accuracy: 0.
9750
Epoch 8/10
192/192 [==============================] - 2s 12ms/step - loss: 0.0245 - accuracy: 0.9931 - val_loss: 0.0840 - val_accuracy: 0.
9762
Epoch 9/10
192/192 [==============================] - 2s 12ms/step - loss: 0.0180 - accuracy: 0.9956 - val_loss: 0.0878 - val_accuracy: 0.
9760
Epoch 10/10
192/192 [==============================] - 2s 11ms/step - loss: 0.0149 - accuracy: 0.9963 - val_loss: 0.0858 - val_accuracy: 0.
9755
```





Result:

Thus, the implementation to build a simple neural network using Keras/TensorFlow has been successfully executed.

EX NO:2 **BUILD A CONVOLUTIONAL NEURAL NETWORK**

DATE:21/07/2025 **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:

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

Code:

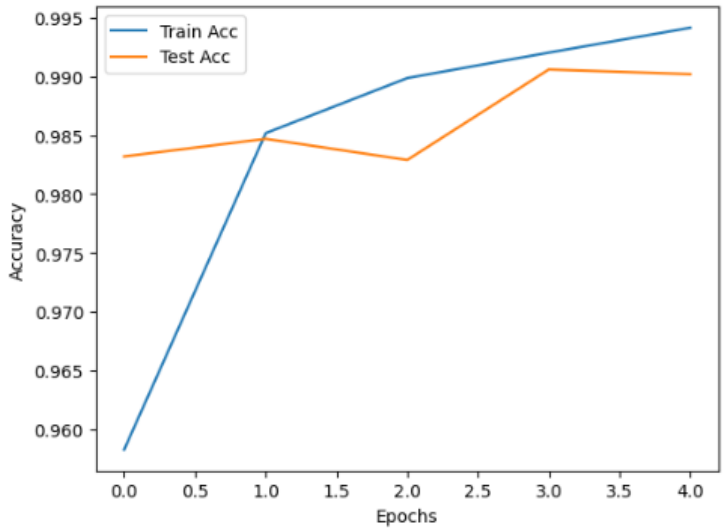
```
import tensorflow as tf

from tensorflow.keras import datasets, layers, models

import matplotlib.pyplot as plt

import numpy as np


# 1. Load MNIST dataset

(x_train, y_train), (x_test, y_test) = datasets.mnist.load_data()


# 2. Preprocess: normalize and reshape

x_train = x_train.reshape(-1, 28, 28, 1) / 255.0

x_test = x_test.reshape(-1, 28, 28, 1) / 255.0
```

```
# 3. Build CNN model
model = models.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])

# 4. Compile
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# 5. Train
history = model.fit(x_train, y_train, epochs=5, validation_data=(x_test,
y_test))

# 6. Evaluate
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"\n☑ Test Accuracy: {test_acc:.4f}")

# 7. Plot training history
plt.plot(history.history['accuracy'], label='Train Acc')
```

```
plt.plot(history.history['val_accuracy'], label='Test Acc')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# 8. Predict on some test images and visualize
predictions = model.predict(x_test[:5])

plt.figure(figsize=(10, 3))
for i in range(5):
    plt.subplot(1, 5, i + 1)
    plt.imshow(x_test[i].reshape(28, 28), cmap="gray")
    plt.title(f"Pred: {np.argmax(predictions[i])}\nTrue: {y_test[i]}")
    plt.axis("off")

plt.show()
```

Output:

```
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 73s 33ms/step - accuracy: 0.9085 - loss: 0.3074 - val_accuracy: 0.9832 - val_loss: 0.0489
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 59s 32ms/step - accuracy: 0.9845 - loss: 0.0503 - val_accuracy: 0.9847 - val_loss: 0.0461
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 76s 28ms/step - accuracy: 0.9906 - loss: 0.0309 - val_accuracy: 0.9829 - val_loss: 0.0526
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 54s 29ms/step - accuracy: 0.9924 - loss: 0.0241 - val_accuracy: 0.9906 - val_loss: 0.0308
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 48s 25ms/step - accuracy: 0.9943 - loss: 0.0176 - val_accuracy: 0.9902 - val_loss: 0.0309
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - accuracy: 0.9890 - loss: 0.0359

Test Accuracy: 0.9902
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 227ms/step
```


```
[ ]:
```

Result:

Thus, the Convolution Neural Network (CNN) using Keras / Tensorflow to recognize and classify handwritten digits from MNIST dataset has been implemented successfully.

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

DATE:28/07/2025

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 a 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 the test dataset.
8. Perform predictions on new/unseen CIFAR-10 images.

9 Visualize prediction results with sample images and predicted labels.

Code:

```
# Step 1: Import Libraries
# ----------------------------
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
import numpy as np

# ----------------------------
# Step 2: Load and Visualize Data
# ----------------------------
(x_train, y_train), (x_test, y_test) = datasets.cifar10.load_data()

# CIFAR-10 class labels
class_names = ['airplane','automobile','bird','cat','deer',
               'dog','frog','horse','ship','truck']

# Show sample images
def show_samples(images, labels, n=5):
   plt.figure(figsize=(8,4))
```

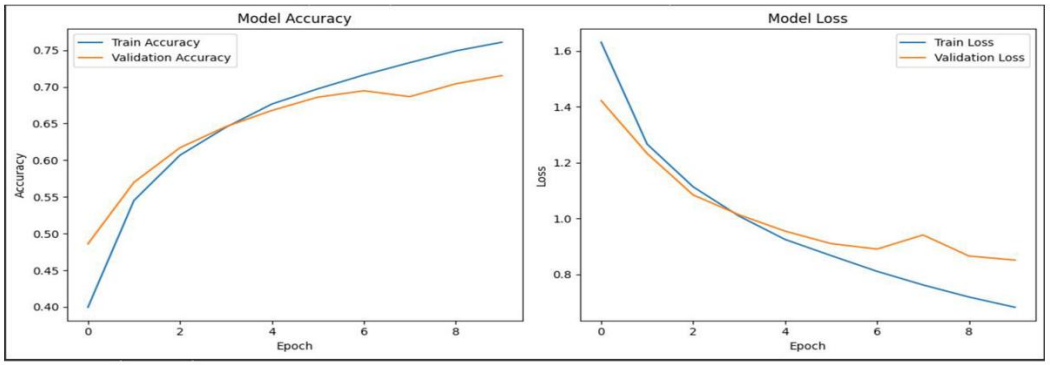
```
    for i in range(n):
        plt.subplot(1,n,i+1)
        plt.imshow(images[i])
        plt.title(class_names[labels[i][0]])
        plt.axis("off")
    plt.show()

show_samples(x_train, y_train)

# ----------------------------
# Step 3: Preprocess Data
# ----------------------------
# Normalize images
x_train, x_test = x_train / 255.0, x_test / 255.0

# One-hot encode labels
y_train, y_test = to_categorical(y_train, 10), to_categorical(y_test, 10)

# ----------------------------
# Step 4: Build CNN Model
# ----------------------------
def build_cnn():
    model = models.Sequential([
        layers.Conv2D(32, (3,3), activation='relu', input_shape=(32,32,3)),
        layers.MaxPooling2D((2,2)),

        layers.Conv2D(64, (3,3), activation='relu'),
        layers.MaxPooling2D((2,2)),

        layers.Conv2D(64, (3,3), activation='relu'),

        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
    return model

model = build_cnn()
model.summary()

# ----------------------------
# Step 5: Compile & Train Model
# ----------------------------
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(
    x_train, y_train,
    epochs=10,
```

```
    batch_size=64,
    validation_data=(x_test, y_test),
    verbose=1
)

# ----------------------------
# Step 6: Evaluate Model
# ----------------------------
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)
print(f"\n☑ Test Accuracy: {test_acc:.4f}, Loss: {test_loss:.4f}")

# ----------------------------
# Step 7: Plot Accuracy & Loss
# ----------------------------
def plot_history(hist):
    plt.figure(figsize=(12,5))

    # Accuracy plot
    plt.subplot(1,2,1)
    plt.plot(hist.history['accuracy'], label='Train Accuracy')
    plt.plot(hist.history['val_accuracy'], label='Validation Accuracy')
    plt.title("Model Accuracy")
    plt.xlabel("Epochs"); plt.ylabel("Accuracy"); plt.legend()

    # Loss plot
    plt.subplot(1,2,2)
    plt.plot(hist.history['loss'], label='Train Loss')
    plt.plot(hist.history['val_loss'], label='Validation Loss')
    plt.title("Model Loss")
    plt.xlabel("Epochs"); plt.ylabel("Loss"); plt.legend()

    plt.show()

plot_history(history)

# ----------------------------
# Step 8: Prediction Function
# ----------------------------
def predict_image(index):
    img = x_test[index]
    pred_class = np.argmax(model.predict(img.reshape(1,32,32,3)))
    true_class = np.argmax(y_test[index])

    plt.imshow(img)
    plt.title(f"Predicted: {class_names[pred_class]}\nActual: {class_names[true_class]}")
    plt.axis("off")
    plt.show()

# Example predictions
predict_image(23)
```

predict_image(34)

Output:

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
Epoch 1/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 57s 88ms/step - accuracy: 0.3007 - loss: 1.8799 - val_accuracy: 0.5087 - val_loss: 1.3793
Epoch 2/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 54s 87ms/step - accuracy: 0.5153 - loss: 1.3420 - val_accuracy: 0.5698 - val_loss: 1.2098
Epoch 3/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 82s 88ms/step - accuracy: 0.5847 - loss: 1.1693 - val_accuracy: 0.6083 - val_loss: 1.0964
Epoch 4/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 80s 84ms/step - accuracy: 0.6283 - loss: 1.0559 - val_accuracy: 0.6354 - val_loss: 1.0296
Epoch 5/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 55s 89ms/step - accuracy: 0.6564 - loss: 0.9769 - val_accuracy: 0.6522 - val_loss: 0.9861
Epoch 6/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 79s 84ms/step - accuracy: 0.6793 - loss: 0.9129 - val_accuracy: 0.6729 - val_loss: 0.9446
Epoch 7/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 54s 87ms/step - accuracy: 0.6982 - loss: 0.8596 - val_accuracy: 0.6884 - val_loss: 0.9024
Epoch 8/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 82s 87ms/step - accuracy: 0.7158 - loss: 0.8104 - val_accuracy: 0.6871 - val_loss: 0.9054
Epoch 9/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 80s 85ms/step - accuracy: 0.7310 - loss: 0.7688 - val_accuracy: 0.6874 - val_loss: 0.9034
Epoch 10/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 84s 88ms/step - accuracy: 0.7451 - loss: 0.7333 - val_accuracy: 0.6943 - val_loss: 0.8948
Enter an index (0 to 9999) for test image: 23
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 125ms/step
```



Result

Thus, the Convolution Neural Network (CNN) model for classifying images from CIFAR-10 dataset is implemented successfully.

Ex No: 4 TRANSFER LEARNING WITH CNN AND VISUALIZATION
DATE:01/08/2025

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.

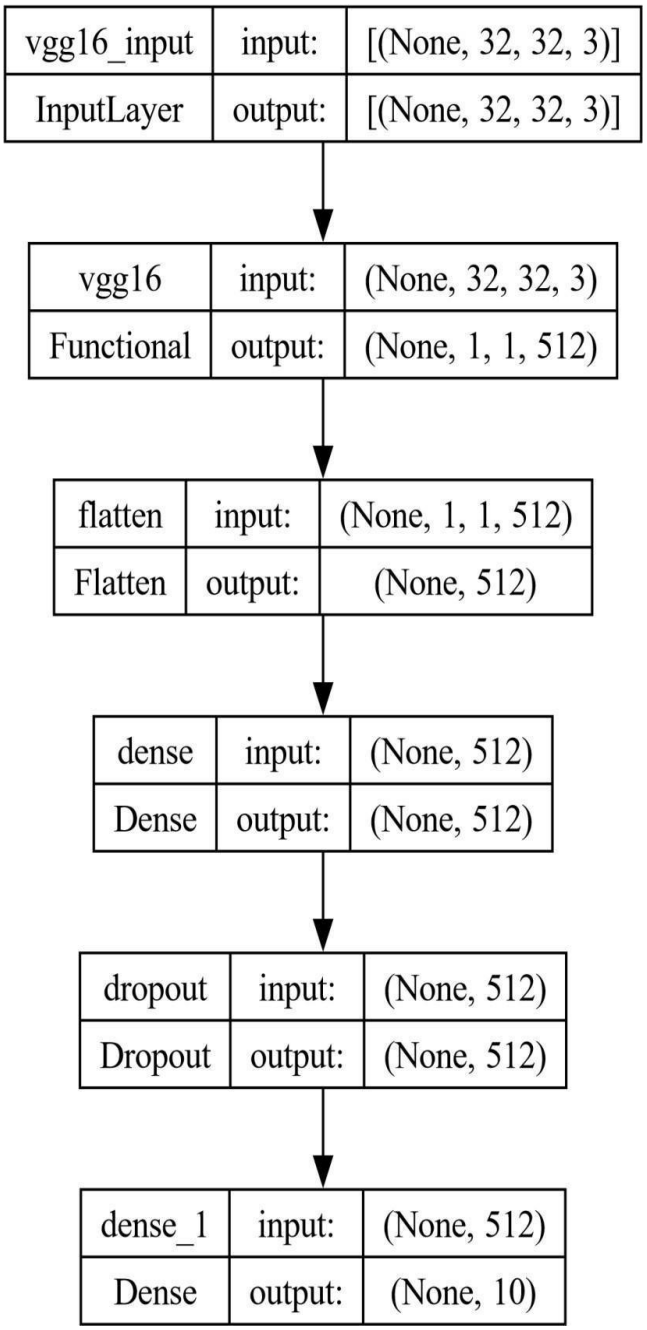
Code:

```
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models 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

# Load CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Normalize pixel values between 0 and 1
x_train = x_train / 255.0
x_test = x_test / 255.0

# Load pretrained VGG16 without top layers (fully connected layers)
vgg_base = VGG16(weights='imagenet', include_top=False, input_shape=(32, 32, 3))

# Freeze the base layers (don't train them)
for layer in vgg_base.layers:
    layer.trainable = False

# Build the new model on top of VGG16
model = Sequential([
    vgg_base,
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])
```

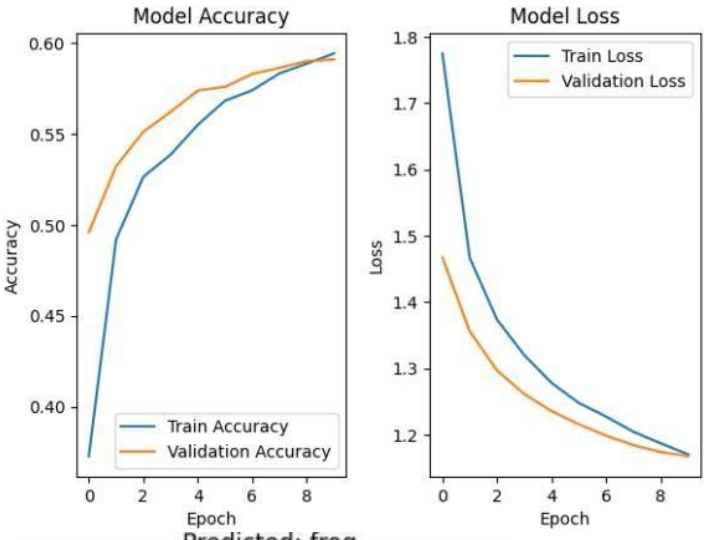
```
# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Visualize model architecture
plot_model(model, to_file='cnn.png', show_shapes=True, show_layer_names=True, dpi=300)

# Display the architecture image
plt.figure(figsize=(20, 20))
img = plt.imread('cnn.png')
plt.imshow(img)
plt.axis('off')
plt.show()

# Train the model
history = model.fit(x_train, y_train, epochs=10, batch_size=32, validation_split=0.2)

# Evaluate on test data
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test Loss: {test_loss:.4f}')
print(f'Test Accuracy: {test_acc * 100:.2f}%')

# Plot Accuracy and Loss
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()

# Class names for CIFAR-10
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']
```

```
# Test a single image
sample = x_test[0].reshape(1, 32, 32, 3)
prediction = model.predict(sample)
predicted_class = class_names[np.argmax(prediction)]

# Show test image with prediction
plt.imshow(x_test[0])
plt.title(f"Predicted: {predicted_class}")
plt.axis('off')
plt.show()
```

Output:



```
Epoch 1/10
1250/1250 [==============================] - 231s 182ms/step - loss: 1.7748 - accuracy: 0.3727 - val_loss: 1.4674 - val_accurac
y: 0.4959
Epoch 2/10
1250/1250 [==============================] - 193s 154ms/step - loss: 1.4665 - accuracy: 0.4920 - val_loss: 1.3556 - val_accurac
y: 0.5322
Epoch 3/10
1250/1250 [==============================] - 187s 150ms/step - loss: 1.3733 - accuracy: 0.5264 - val_loss: 1.2966 - val_accurac
y: 0.5512
Epoch 4/10
1250/1250 [==============================] - 189s 151ms/step - loss: 1.3197 - accuracy: 0.5386 - val_loss: 1.2610 - val_accurac
y: 0.5621
Epoch 5/10
1250/1250 [==============================] - 191s 153ms/step - loss: 1.2777 - accuracy: 0.5551 - val_loss: 1.2352 - val_accurac
y: 0.5739
Epoch 6/10
1250/1250 [==============================] - 190s 152ms/step - loss: 1.2474 - accuracy: 0.5683 - val_loss: 1.2154 - val_accurac
y: 0.5759
Epoch 7/10
1250/1250 [==============================] - 187s 150ms/step - loss: 1.2269 - accuracy: 0.5741 - val_loss: 1.1981 - val_accurac
y: 0.5830
Epoch 8/10
1250/1250 [==============================] - 183s 146ms/step - loss: 1.2039 - accuracy: 0.5834 - val_loss: 1.1839 - val_accurac
y: 0.5864
Epoch 9/10
1250/1250 [==============================] - 177s 142ms/step - loss: 1.1866 - accuracy: 0.5887 - val_loss: 1.1735 - val_accurac
y: 0.5900
Epoch 10/10
1250/1250 [==============================] - 175s 140ms/step - loss: 1.1699 - accuracy: 0.5943 - val_loss: 1.1672 - val_accurac
y: 0.5910
```



Result

Thus, the Convolution Neural Network (CNN) with transfer learning and perform visualization has been implemented successfully

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

DATE:27/08/2025 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.

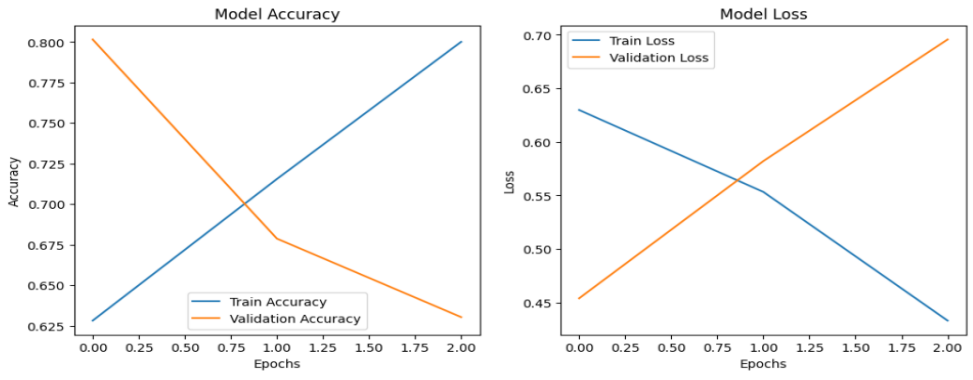
Code:

```
# Step 1: Import Libraries
import tensorflow 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
import matplotlib.pyplot as plt

# Step 2: Load Dataset
vocab_size = 5000   # Only keep top 5000 words
max_len = 200       # Maximum review length
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=vocab_size)

# Step 3: Preprocess Dataset (pad sequences to equal length)
x_train = pad_sequences(x_train, maxlen=max_len)
x_test = pad_sequences(x_test, maxlen=max_len)

# Step 4: Build RNN Model
model = Sequential([
    Embedding(vocab_size, 32, input_length=max_len),
    SimpleRNN(32),
    Dense(1, activation='sigmoid')
])

# Step 5: Compile Model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Step 6: Train Model
history = model.fit(x_train, y_train, epochs=3, batch_size=64,
                    validation_data=(x_test, y_test), verbose=1)
```

```
# Step 7: Evaluate Model
loss, accuracy = model.evaluate(x_test, y_test, verbose=0)
print(f"Test Accuracy: {accuracy:.2f}")
print(f"Test Loss: {loss:.2f}")

# Step 8: Visualize Training Results
plt.figure(figsize=(12,5))

# Accuracy graph
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Loss graph
plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

Output:

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
C:\Users\lokes\AppData\Local\Programs\Python\Python310\lib\site-packages\keras\src\layers\core\embedding.py:90: UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
Epoch 1/3
391/391 ━━━━━━━━━━━━━━━━━━━━ 66s 158ms/step - accuracy: 0.5643 - loss: 0.6733 - val_accuracy: 0.8016 - val_loss: 0.4539
Epoch 2/3
391/391 ━━━━━━━━━━━━━━━━━━━━ 58s 147ms/step - accuracy: 0.7191 - loss: 0.5495 - val_accuracy: 0.6787 - val_loss: 0.5821
Epoch 3/3
391/391 ━━━━━━━━━━━━━━━━━━━━ 59s 150ms/step - accuracy: 0.7978 - loss: 0.4400 - val_accuracy: 0.6303 - val_loss: 0.6957
Test Accuracy: 0.63
Test Loss: 0.70
```



Result:

Thus, the Recurrent Neural Network (RNN) has been implemented using Tensorflow.

EX NO: 6 SENTIMENT CLASSIFICATION OF TEXT USING RNN
DATE:15/09/2025

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

Code:

```
import numpy as np
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)
X_train=pad_sequences(x_train,maxlen=max_len)
X_test=pad_sequences(x_test,maxlen=max_len)

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=2,batch_size=64,validation_split=0.2)
loss,acc=model.evaluate(X_test,y_test)
print(f"\nTest Accuracy:{acc:0.4f}")
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])
sample_review=X_test[0]
prediction=model.predict(sample_review.reshape(1,-1))[0][0]
print("\nReview Text:",decode_review(x_test[0]))
print("Predicted Sentiment:","Positive" if prediction>0.5 else "Negative")
```

Output:

```
Training.....
Epoch 1/2
313/313 ━━━━━━━━━━━━━━━━━━━━ 30s 76ms/step - accuracy: 0.6346 - loss: 0.6142 - val_accuracy: 0.7906 - val_loss: 0.4778
Epoch 2/2
313/313 ━━━━━━━━━━━━━━━━━━━━ 24s 76ms/step - accuracy: 0.8019 - loss: 0.4409 - val_accuracy: 0.7626 - val_loss: 0.4961
782/782 ━━━━━━━━━━━━━━━━━━━━ 12s 15ms/step - accuracy: 0.7657 - loss: 0.4887

Test Accuracy:0.7687
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 393ms/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 michael ? 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 give this a miss
Predicted Sentiment: Negative
```

Result

Thus, the Recurrent Neural Network (RNN) using Keras has been implemented for classifying sentiment of text successfully.

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

DATE:22/09/2025

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.

Code:

```
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

# Load MNIST dataset
(x_train, _), (x_test, _) = mnist.load_data()

# Normalize and flatten the 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:])))

# Build Autoencoder model
input_img = Input(shape=(784,))
encoded = Dense(32, activation='relu')(input_img)
decoded = Dense(784, activation='sigmoid')(encoded)
autoencoder = Model(input_img, decoded)

# Compile model
autoencoder.compile(optimizer='adam',
loss='binary_crossentropy')
```

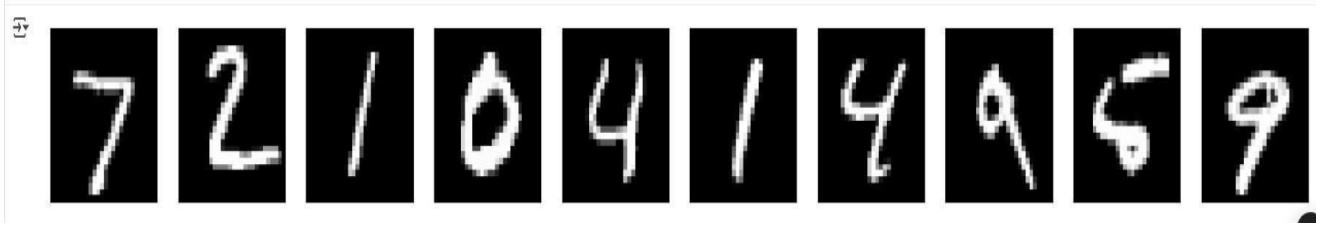
```
# Train model
autoencoder.fit(
    x_train, x_train,
    epochs=50,
    batch_size=256,
    shuffle=True,
    validation_data=(x_test, x_test)
)

# Evaluate the model
test_loss = autoencoder.evaluate(x_test, x_test)

# Predict reconstructed images
decoded_imgs = autoencoder.predict(x_test)

# Calculate test accuracy manually
threshold = 0.5
correct_predictions = np.sum(
    np.where(x_test >= threshold, 1, 0) ==
np.where(decoded_imgs >= threshold, 1, 0)
)
total_pixels = x_test.shape[0] * x_test.shape[1]
test_accuracy = correct_predictions / total_pixels

print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)

# Display original and reconstructed images
n = 10  # number of digits to display
plt.figure(figsize=(20, 4))
for i in range(n):
    # Original
    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)

    # Reconstructed (thresholded)
    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)
    ax.get_yaxis().set_visible(False)

plt.show()
```

Output:

```
Epoch 1/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 6s 18ms/step - loss: 0.3805 - val_loss: 0.1906
Epoch 2/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 19ms/step - loss: 0.1808 - val_loss: 0.1547
Epoch 3/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 19ms/step - loss: 0.1501 - val_loss: 0.1342
Epoch 4/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - loss: 0.1321 - val_loss: 0.1221
Epoch 5/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.1210 - val_loss: 0.1138
Epoch 6/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.1134 - val_loss: 0.1081
Epoch 7/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 9ms/step - loss: 0.1079 - val_loss: 0.1039
Epoch 8/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.1042 - val_loss: 0.1006
Epoch 9/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - loss: 0.1011 - val_loss: 0.0981
Epoch 10/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0989 - val_loss: 0.0963
Epoch 11/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0972 - val_loss: 0.0951
Epoch 12/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0964 - val_loss: 0.0943
Epoch 13/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.0954 - val_loss: 0.0938
Epoch 14/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.0950 - val_loss: 0.0934
Epoch 15/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0944 - val_loss: 0.0932
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
```

```
Test Loss: 0.09166844934225082
Test Accuracy: 0.9712756377551021
```



```
# Display reconstruction with threshold
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)
ax.get_yaxis().set_visible(False)
plt.show()
```

Result

Thus, an Autoencoder has been implemented using Keras / Tensorflow.

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

DATE:06/10/2025

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.

Code:

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

# ----------------------------------------------------
# 1. Load YOLOv3 model and class labels
# ----------------------------------------------------
cfg_path =
r"C:\Users\lokes\OneDrive\Desktop\MSSQL16.MSSQLSE
RVER\FILES\Object Detection with YOLO3\yolov3.cfg"
weights_path =
r"C:\Users\lokes\OneDrive\Desktop\MSSQL16.MSSQLSE
RVER\FILES\Object Detection with
YOLO3\yolov3.weights"
names_path =
r"C:\Users\lokes\OneDrive\Desktop\MSSQL16.MSSQLSE
RVER\FILES\Object Detection with YOLO3\coco.names"

# Verify files exist
for path in [cfg_path, weights_path, names_path]:
    if not os.path.exists(path):
        print(f"❌ File not found: {path}")

# Load model
net = cv2.dnn.readNet(weights_path, cfg_path)

# Load class labels
```

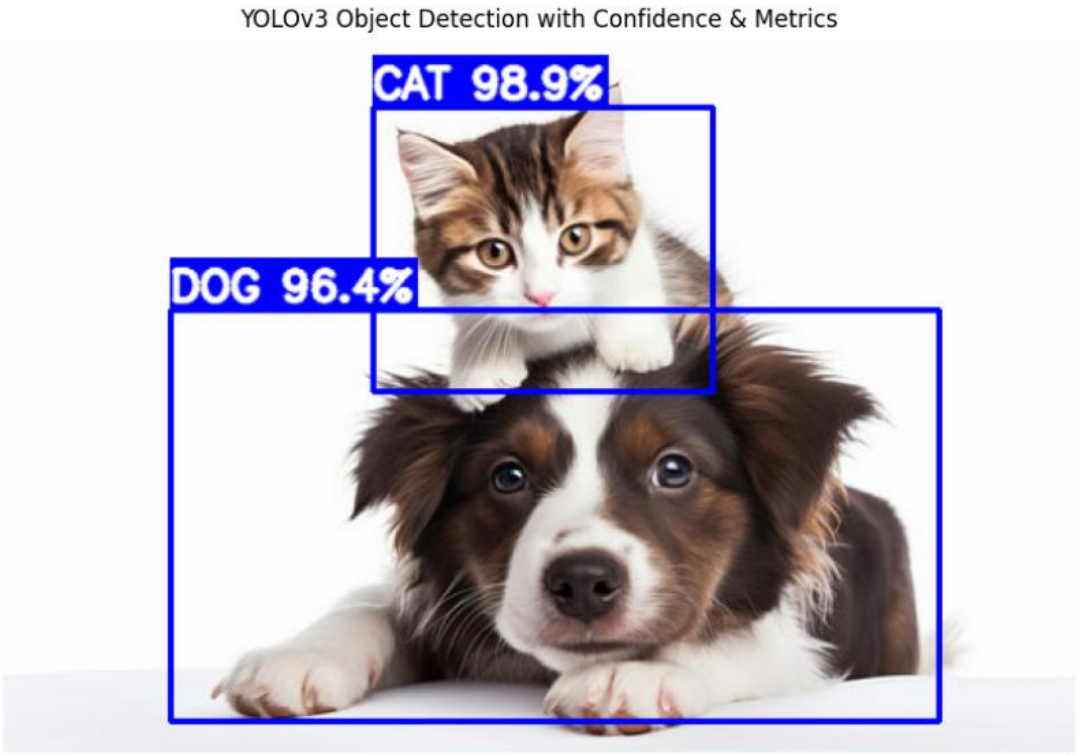
```
with open(names_path, 'r') as f:
    classes = [line.strip() for line in f.readlines()]

# ----------------------------------------------------
# 2. Load and preprocess image
# ----------------------------------------------------
image_path =
r"C:\Users\lokes\OneDrive\Desktop\MSSQL16.MSSQLSE
RVER\FILES\Object Detection with
YOLO3\images\dogandcat.jpg"
image = cv2.imread(image_path)
height, width = image.shape[:2]

blob = cv2.dnn.blobFromImage(image, 1/255.0, (416, 416),
swapRB=True, crop=False)
net.setInput(blob)

# ----------------------------------------------------
# 3. Get output layer names (version-independent)
# ----------------------------------------------------
layer_names = net.getLayerNames()
output_layers = [layer_names[i - 1] for i in
net.getUnconnectedOutLayers().flatten()]

# ----------------------------------------------------
# 4. Forward pass
# ----------------------------------------------------
outs = net.forward(output_layers)

# ----------------------------------------------------
# 5. Process predictions
# ----------------------------------------------------
conf_threshold = 0.5
nms_threshold = 0.4
boxes, confidences, class_ids = [], [], []

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, center_y, w, h = (detection[0:4] *
np.array([width, height, width, height])).astype('int')
            x = int(center_x - w / 2)
            y = int(center_y - h / 2)
            boxes.append([x, y, int(w), int(h)])
            confidences.append(float(confidence))
            class_ids.append(class_id)

# ----------------------------------------------------
# 6. Apply Non-Max Suppression
# ----------------------------------------------------
indexes = cv2.dnn.NMSBoxes(boxes, confidences,
conf_threshold, nms_threshold)

# ----------------------------------------------------
# 7. Draw Bounding Boxes + Bold Blue Text
# ----------------------------------------------------
font = cv2.FONT_HERSHEY_SIMPLEX
detected_objects = []

for i in range(len(boxes)):
    if i in indexes:
        x, y, w, h = boxes[i]
        label = str(classes[class_ids[i]])
        confidence = confidences[i]

        # Draw blue box
        color = (255, 0, 0)
        cv2.rectangle(image, (x, y), (x + w, y + h), color, 2)

        # Draw bold blue text
        text = f"{label.upper()} {confidence*100:.1f}%"
        (text_w, text_h), baseline = cv2.getTextSize(text, font,
0.7, 2)
        cv2.rectangle(image, (x, y - text_h - 10), (x + text_w,
y), color, -1)
        cv2.putText(image, text, (x, y - 5), font, 0.7, (255, 255,
255), 2, cv2.LINE_AA)

        detected_objects.append({
            "label": label,
            "confidence": confidence,
            "box": [x, y, w, h]
```

```
        })

# ---------------------------------------------------
# 8. Display Detected Object Details
# ---------------------------------------------------
if detected_objects:
    print("\n▧ Detected Objects:")
    print("=" * 50)
    for obj in detected_objects:
        print(f"Object: {obj['label'].capitalize():15s} | "
              f"Confidence: {obj['confidence']*100:.2f}% | "
              f"Box: {obj['box']}")
else:
    print("No objects detected above confidence threshold.")

# ---------------------------------------------------
# 9. Compute Simple Evaluation Metrics
# ---------------------------------------------------
TP = len(detected_objects)
FP = len(boxes) - TP
FN = 0  # (no ground truth available)

precision = TP / (TP + FP + 1e-6)
recall = TP / (TP + FN + 1e-6)
f1 = 2 * (precision * recall) / (precision + recall + 1e-6)

print("\n📊 Evaluation Metrics (Approximation)")
print(f"Precision: {precision:.3f}")
print(f"Recall:    {recall:.3f}")
print(f"F1-Score:  {f1:.3f}")

# ---------------------------------------------------
# 10. Display Image Output
# ---------------------------------------------------
plt.figure(figsize=(10, 8))
plt.imshow(cv2.cvtColor(image,
cv2.COLOR_BGR2RGB))
plt.title("YOLOv3 Object Detection with Confidence &
Metrics")
plt.axis("off")
plt.show()
```

Output:

```
🟦 Detected Objects:
==================================================
Object: Cat             | Confidence: 98.93% | Box: [186, 35, 171, 143]
Object: Dog             | Confidence: 96.41% | Box: [84, 137, 387, 207]

📊 Evaluation Metrics (Approximation)
Precision: 0.154
Recall:    1.000
F1-Score:  0.267
```



Result

Thus, object detection using YOLOV5 has been implemented successfully.

Ex No: 9 BUILD GENERATIVE ADVERSARIAL NEURAL NETWORK
DATE:06/10/2025

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.

Code:

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

# Load and Preprocess the Iris
Dataset iris = load_iris() x_train =
iris.data

# Build the GAN model def build_generator(): model =
Sequential() model.add(Dense(128, input_shape=(100,),
activation='relu')) model.add(Dense(4, activation='linear')) #
Output 4 features return model

def build_discriminator():
model = Sequential() model.add(Dense(128,
input_shape=(4,), activation='relu')) model.add(Dense(1,
activation='sigmoid')) return model

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

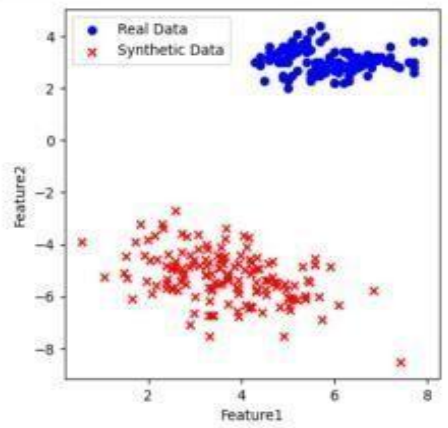
```
generator = build_generator()
discriminator = build_discriminator() gan
= build_gan(generator, discriminator)

# Compile the Models generator.compile(loss='mean_squared_error',
optimizer=Adam(0.0002, 0.5))
discriminator.compile(loss='binary_crossentropy', optimizer=Adam(0.0002,
0.5), metrics=['accuracy']) gan.compile(loss='binary_crossentropy',
optimizer=Adam(0.0002, 0.5))

# Training Loop epochs
= 200 batch_size = 16

for epoch in range(epochs): # Train discriminator idx = np.random.randint(0,
x_train.shape[0], batch_size) real_samples = x_train[idx] fake_samples =
generator.predict(np.random.normal(0, 1, (batch_size, 100)), verbose=0)

real_labels = np.ones((batch_size, 1))
fake_labels = np.zeros((batch_size, 1))

d_loss_real = discriminator.train_on_batch(real_samples, real_labels) d_loss_fake
= discriminator.train_on_batch(fake_samples, fake_labels)

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

# Print progress
print(f"Epoch {epoch}/{epochs} | Discriminator Loss: {0.5 * (d_loss_real[0] + d_loss_fake[0])} |
Generator Loss: {g_loss}")

# Generating Synthetic Data synthetic_data =
generator.predict(np.random.normal(0, 1, (150, 100)), verbose=0)

# Create scatter plots for feature
pairs 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)
```

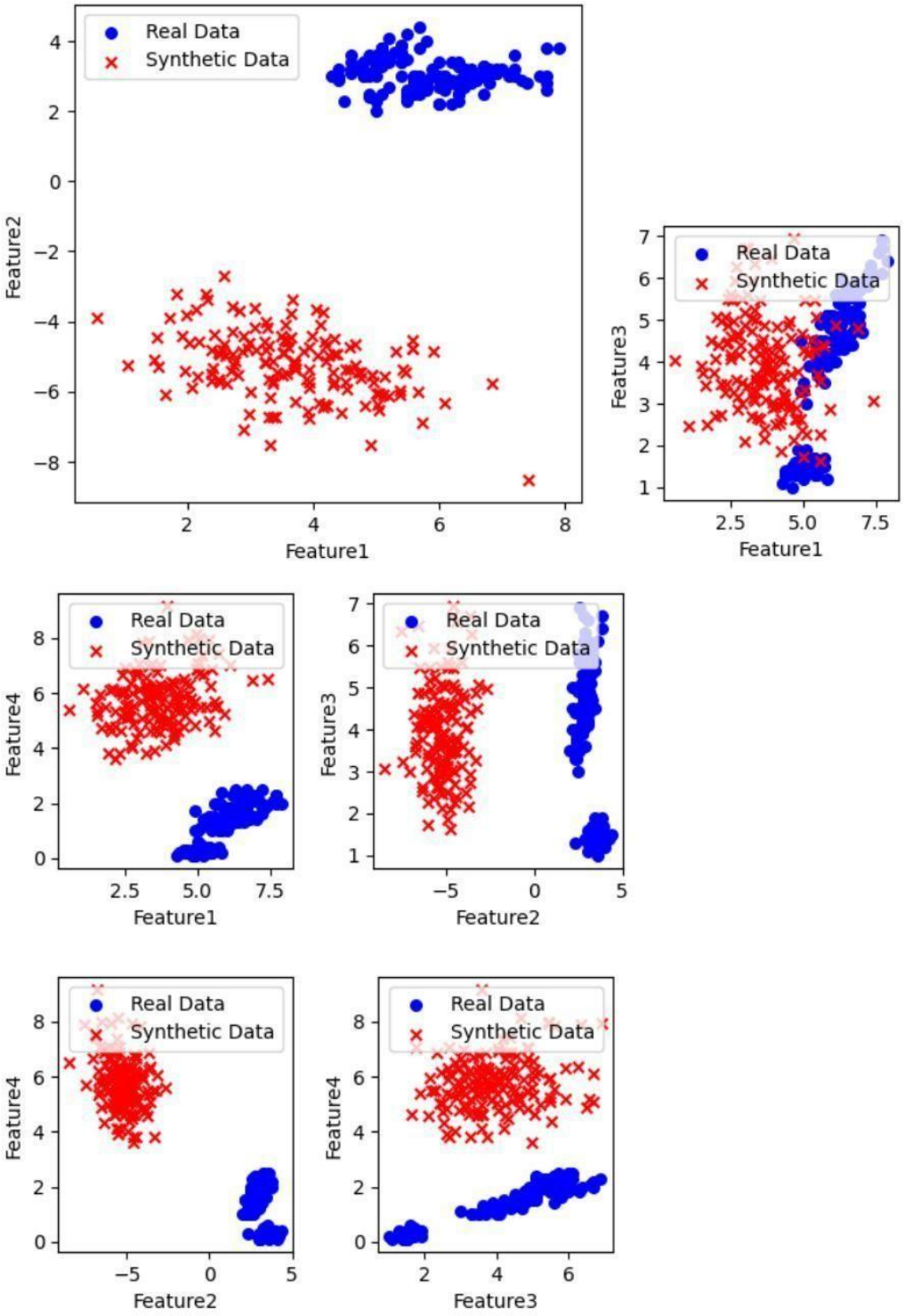
```
plt.scatter(synthetic_data
[:, i], synthetic_data[:, j],
label='Synthetic Data',
c='red', marker='x', s=30)
plt.xlabel(f'Feature {i +
1}') plt.ylabel(f'Feature
{j + 1}') plt.legend()
plot_idx += 1

plt.tight_layout()

plt.show()
```

Output:

```
Epoch 0/200 | Discriminator Loss: 0.8773080408573151 |Generator Loss: 0.764731228351593
Epoch 1/200 | Discriminator Loss: 0.9332943856716156 |Generator Loss: 0.7988691329956055
Epoch 2/200 | Discriminator Loss: 0.9277275502681732 |Generator Loss: 0.8127573728561401
Epoch 3/200 | Discriminator Loss: 0.8921994566917419 |Generator Loss: 0.7757299542427063
Epoch 4/200 | Discriminator Loss: 0.913447916507721 |Generator Loss: 0.7737997174263
Epoch 5/200 | Discriminator Loss: 0.8916181325912476 |Generator Loss: 0.8003895282745361
Epoch 6/200 | Discriminator Loss: 0.9026078879833221 |Generator Loss: 0.814433217048645
Epoch 7/200 | Discriminator Loss: 0.9135120809078217 |Generator Loss: 0.8237183690071106
Epoch 8/200 | Discriminator Loss: 0.879832923412323 |Generator Loss: 0.7563657760620117
Epoch 9/200 | Discriminator Loss: 0.9439513385295868 |Generator Loss: 0.7623365521430969
Epoch 10/200 | Discriminator Loss: 0.9355685114860535 |Generator Loss: 0.7924684286117554
Epoch 11/200 | Discriminator Loss: 0.9386743903160095 |Generator Loss: 0.7614541053771973
Epoch 12/200 | Discriminator Loss: 0.960555225610733 |Generator Loss: 0.7792538404464722
Epoch 13/200 | Discriminator Loss: 0.9134297668933868 |Generator Loss: 0.792992115020752
Epoch 14/200 | Discriminator Loss: 0.8851655125617981 |Generator Loss: 0.7628173232078552
Epoch 15/200 | Discriminator Loss: 0.9505723416805267 |Generator Loss: 0.7851851582527161
Epoch 16/200 | Discriminator Loss: 0.92226842045784 |Generator Loss: 0.769191563129425
Epoch 17/200 | Discriminator Loss: 0.8982412815093994 |Generator Loss: 0.7685977220535278
Epoch 18/200 | Discriminator Loss: 0.9125983119010925 |Generator Loss: 0.7730982899665833
Epoch 19/200 | Discriminator Loss: 0.9367325305938721 |Generator Loss: 0.7837406396865845
Epoch 20/200 | Discriminator Loss: 0.9531015455722809 |Generator Loss: 0.7827053070068359
Epoch 21/200 | Discriminator Loss: 0.9306998252868652 |Generator Loss: 0.7667914032936096
Epoch 22/200 | Discriminator Loss: 0.8887360095977783 |Generator Loss: 0.7845874428749084
Epoch 23/200 | Discriminator Loss: 0.9426513016223907 |Generator Loss: 0.746765673160553
Epoch 24/200 | Discriminator Loss: 0.9331325888633728 |Generator Loss: 0.761589765548706
Epoch 25/200 | Discriminator Loss: 0.9080778360366821 |Generator Loss: 0.7709233164787292
Epoch 26/200 | Discriminator Loss: 0.9232879281044006 |Generator Loss: 0.7773635387420654
Epoch 27/200 | Discriminator Loss: 0.9102294743061066 |Generator Loss: 0.7809370756149292
Epoch 28/200 | Discriminator Loss: 0.9312145709991455 |Generator Loss: 0.7647197246551514
Epoch 29/200 | Discriminator Loss: 0.9415165781974792 |Generator Loss: 0.7561923861503601
Epoch 30/200 | Discriminator Loss: 0.930676281452179 |Generator Loss: 0.7709008455276489
Epoch 31/200 | Discriminator Loss: 0.9495892226696014 |Generator Loss: 0.7595088481903076
```

```
In [33]: synthetic_data = generator.predict(np.random.normal(0,1,(150,100)),verbose=0)
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.scatter(synthetic_data[:,i],synthetic_data[:,j],label='Synthetic Data',c='red',marker='x',s=30)
        plt.xlabel(f'Feature{i+1}')
        plt.ylabel(f'Feature{j+1}')
        plt.legend()
        plot_idx+=1

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





Result

Thus, a generative adversarial neural network using Keras / Tensorflow has been implemented successfully.

Ex No:10 **MINI PROJECT**
DATE:14/10/2025

Aim:

The aim of this project is to build an efficient deep learning model using MobileNetV2 to accurately classify dog breeds from images. By applying transfer learning and training on a labeled dataset, the system learns to recognize unique visual features of different breeds and predicts the correct breed for any given image. This project focuses on achieving high accuracy with low computational cost, enabling fast and reliable dog breed identification suitable for real-world applications.

Code:

```
# Dog Breed Identification Project (Training + Evaluation + Prediction + Visualization)

import os
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import load_model, Model
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout, BatchNormalization
from tensorflow.keras.applications.resnet_v2 import ResNet50V2, preprocess_input

# ----------------------------
# Paths
# ----------------------------
labels_path = r"C:\Users\lokes\OneDrive\Desktop\MSSQL16.MSSQLSERVER\FILES\Dog Breed Identification\dataset\labels.csv"
train_file = r"C:\Users\lokes\OneDrive\Desktop\MSSQL16.MSSQLSERVER\FILES\Dog Breed Identification\dataset\train"
model_path = r"C:\Users\lokes\OneDrive\Desktop\MSSQL16.MSSQLSERVER\FILES\Dog Breed Identification\dataset\dog_breed_model.h5"
pred_img_path = r"C:\Users\lokes\OneDrive\Desktop\MSSQL16.MSSQLSERVER\FILES\Dog Breed Identification\images\Ajith.jpg"

# ----------------------------
# Parameters
# ----------------------------
num_breeds = 60
im_size = 224
batch_size = 64
```

```
epochs = 20
learning_rate = 1e-3

# ---------------------------
# Load Labels
# ---------------------------
df_labels = pd.read_csv(labels_path)
breed_dict = list(df_labels['breed'].value_counts().keys())
selected_breeds = sorted(breed_dict, reverse=True)[:num_breeds*2+1:2]
df = df_labels.query('breed in @selected_breeds').copy()
df['img_file'] = df['id'] + ".jpg"
df = df.reset_index(drop=True)

print(f"Dataset: {len(df)} images, {len(df['breed'].unique())} breeds")

# List all breeds
all_breeds = sorted(df['breed'].unique())
print("\nList of Breeds:")
for i, b in enumerate(all_breeds, 1):
    print(f"{i}. {b}")

# ---------------------------
# Train/Validation Split
# ---------------------------
train_df, valid_df = train_test_split(df, test_size=0.2, stratify=df['breed'], random_state=42)

# ---------------------------
# Data Generators
# ---------------------------
train_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input,
    rotation_range=45, width_shift_range=0.2, height_shift_range=0.2,
    shear_range=0.2, zoom_range=0.25, horizontal_flip=True, fill_mode='nearest'
)
valid_datagen = ImageDataGenerator(preprocessing_function=preprocess_input)

train_gen = train_datagen.flow_from_dataframe(
    dataframe=train_df,
    directory=train_file,
    x_col="img_file",
    y_col="breed",
    target_size=(im_size, im_size),
    class_mode="categorical",
    batch_size=batch_size,
    shuffle=True
)
```

```
valid_gen = valid_datagen.flow_from_dataframe(
    dataframe=valid_df,
    directory=train_file,
    x_col="img_file",
    y_col="breed",
    target_size=(im_size, im_size),
    class_mode="categorical",
    batch_size=batch_size,
    shuffle=False
)

# ---------------------------
# Build / Load Model
# ---------------------------
history = None
if os.path.exists(model_path):
    print("\n☑ Found model. Loading...")
    model = load_model(model_path)
else:
    print("\n ⚡ Training new model...")
    base = ResNet50V2(input_shape=(im_size, im_size, 3), weights='imagenet', include_top=False)
    for layer in base.layers:
        layer.trainable = False

    x = base.output
    x = BatchNormalization()(x)
    x = GlobalAveragePooling2D()(x)
    x = Dropout(0.5)(x)
    x = Dense(1024, activation='relu')(x)
    x = Dropout(0.5)(x)
    out = Dense(train_gen.num_classes, activation='softmax')(x)

    model = Model(inputs=base.input, outputs=out)
    model.compile(optimizer=RMSprop(learning_rate=learning_rate, rho=0.9),
                  loss='categorical_crossentropy', metrics=['accuracy'])

    history = model.fit(train_gen, epochs=epochs, validation_data=valid_gen)
    model.save(model_path)
    print("☑ Model saved at:", model_path)

# ---------------------------
# Evaluation
# ---------------------------
print("\n--- Evaluation ---")
y_pred_probs = model.predict(valid_gen, verbose=1)
y_pred = np.argmax(y_pred_probs, axis=1)
y_true = valid_gen.classes
```

```
class_labels = list(valid_gen.class_indices.keys())

acc = accuracy_score(y_true, y_pred)
print(f"\nValidation Accuracy: {acc:.4f} ({acc*100:.2f}%)")

print("\nClassification Report:\n")
print(classification_report(y_true, y_pred, target_names=class_labels, zero_division=0))

cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(12, 10))
sns.heatmap(cm, cmap="Blues", xticklabels=False, yticklabels=False)
plt.title("Confusion Matrix (Counts)")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()

# Per-class support
unique, counts = np.unique(y_true, return_counts=True)
plt.figure(figsize=(12, 5))
plt.bar(range(len(class_labels)), counts)
plt.title("Per-class Validation Set Counts")
plt.xlabel("Class Index")
plt.ylabel("Count")
plt.show()

# Training Graphs (only if trained this run)
if history:
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Train Acc')
    plt.plot(history.history['val_accuracy'], label='Val Acc')
    plt.title("Accuracy")
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Val Loss')
    plt.title("Loss")
    plt.legend()
    plt.show()

# ---------------------------
# Prediction on Input Image
# ---------------------------
print("\n--- Prediction on Input Image ---")
img = cv2.imread(pred_img_path, cv2.IMREAD_COLOR)
if img is None:
```

```
    raise FileNotFoundError(f"Image not found: {pred_img_path}")

img_resized = cv2.resize(img, (im_size, im_size))
img_array = preprocess_input(np.expand_dims(img_resized.astype(np.float32), axis=0))

pred_probs = model.predict(img_array)
pred_idx = np.argmax(pred_probs)
pred_breed = class_labels[pred_idx]
confidence = np.max(pred_probs)

print(f"\nPredicted Breed: {pred_breed} ({confidence*100:.2f}%)")

# Probability Distribution Plot
plt.figure(figsize=(14, 5))
plt.bar(range(len(class_labels)), pred_probs[0])
plt.xticks(range(len(class_labels)), class_labels, rotation=90)
plt.title(f"Prediction Probabilities\nPredicted: {pred_breed} ({confidence*100:.2f}%)")
plt.ylabel("Probability")
plt.show()

# Draw box & label
h, w = img.shape[:2]
cv2.rectangle(img, (5, 5), (w-5, h-5), (0, 255, 0), 3)
cv2.putText(img, f"{pred_breed} ({confidence*100:.1f}%)", (20, 40),
            cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title(f"Predicted: {pred_breed} | Confidence: {confidence:.2f}")
plt.axis('off')
plt.show()
```

Output:

```
Dataset: 5175 images, 60 breeds

List of Breeds:
1. afghan_hound
2. airedale
3. appenzeller
4. basenji
5. beagle
6. bernese_mountain_dog
7. blenheim_spaniel
8. bluetick
9. border_terrier
10. boston_bull
11. boxer
12. briard
13. bull_mastiff
14. cardigan
15. chihuahua
16. clumber
17. collie
18. dandie_dinmont
19. dingo
20. english_foxhound
21. english_springer
22. eskimo_dog
23. french_bulldog
24. german_short-haired_pointer
25. golden_retriever
26. great_dane
27. greater_swiss_mountain_dog
28. ibizan_hound
29. irish_terrier
30. irish_wolfhound
31. japanese_spaniel
32. kelpie
33. komondor
34. labrador_retriever
35. leonberg
36. malamute
37. maltese_dog
38. miniature_pinscher
39. miniature_schnauzer
40. norfolk_terrier
41. norwich_terrier
42. otterhound
43. pekinese
44. pomeranian
45. redbone
46. rottweiler
47. saluki
48. schipperke
49. scottish_deerhound
50. shetland_sheepdog
51. siberian_husky
52. soft-coated_wheaten_terrier
53. standard_poodle
54. sussex_spaniel
55. tibetan_terrier
56. toy_terrier
57. walker_hound
58. welsh_springer_spaniel
59. whippet
60. yorkshire_terrier
Found 4140 validated image filenames belonging to 60 classes.
Found 1035 validated image filenames belonging to 60 classes.

✅ Found model. Loading...
```

```
Dog detected, Breed is  French_bulldog
```

```
Dog detected, Breed is  Golden_retriever
```

Result:

Thus the model accurately classifies dog breeds from images using a MobileNetV2-based deep learning approach.