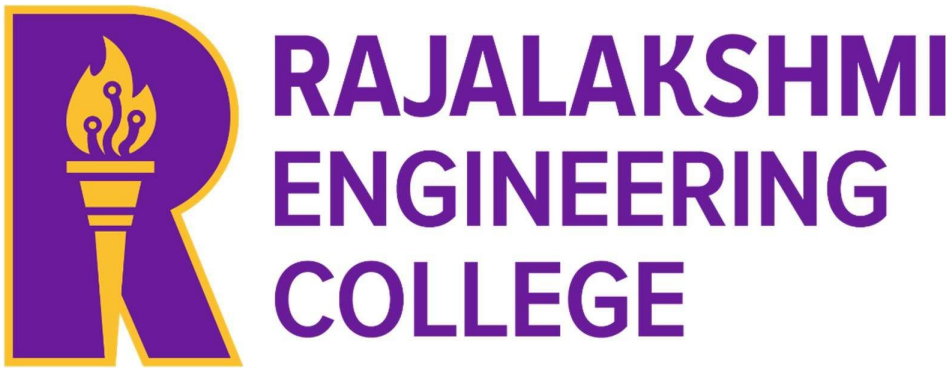
RAJALAKSHMI ENGINEERING COLLEGE

(An Autonomous Institution)

RAJALAKSHMI NAGAR, THANDALAM- 602 105



CS19P18 - DEEP LEARNING CONCEPTS

LABORATORY RECORD NOTEBOOK

NAME: TAMILINI DK

YEAR/SEMESTER: IV/VII

BRANCH: CSE

REGISTER NO: 2116220701299

COLLEGE ROLL NO: 220701299

ACADEMIC YEAR: 2025 -2026

RAJALAKSHMI ENGINEERING COLLEGE

(An Autonomous Institution)

RAJALAKSHMI NAGAR, THANDALAM- 602 105

BONAFIDE CERTIFICATE

NAME: TAMILINI D K **BRANCH/SECTION:** CSE

ACADEMIC YEAR: 2025 -2026 **SEMESTER:** VII

REGISTER NO: **2116220701299**

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

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)

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.

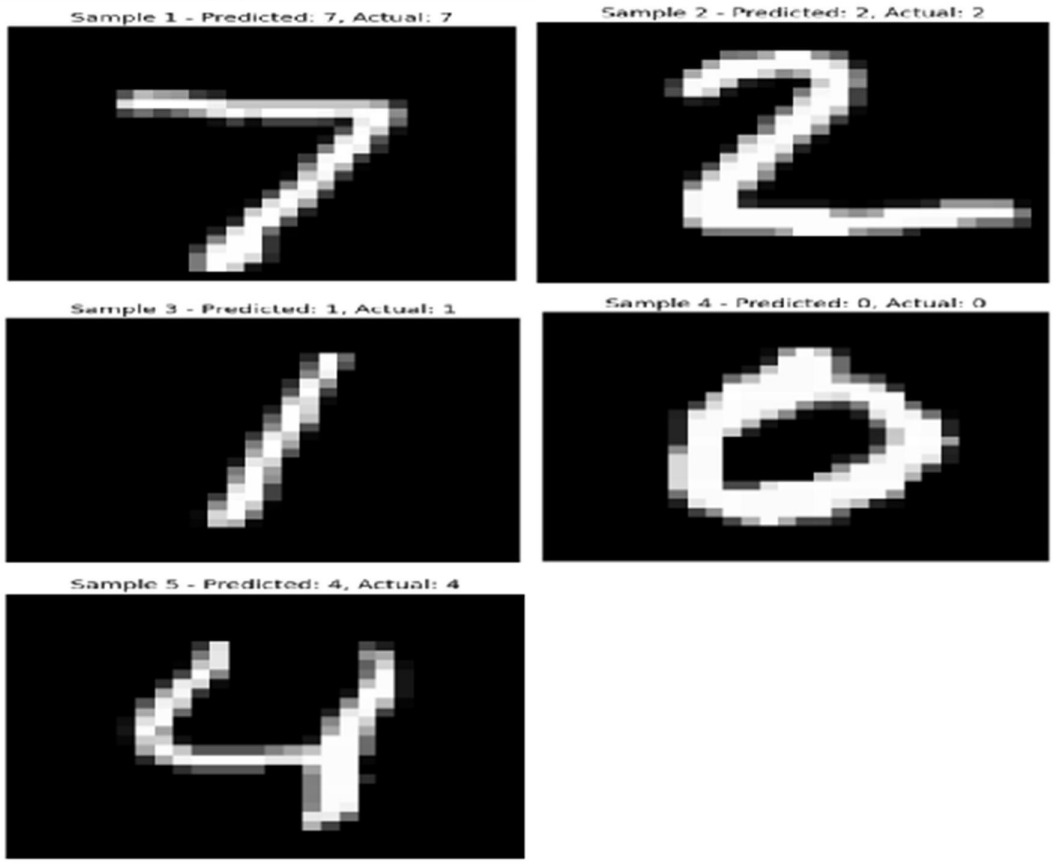
Code:

```
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()

input_shape = (feature_vector_length)
print(f'Feature shape: {input_shape}')

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=input_shape, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss='categorical_crossentropy', 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'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()
```

Output:

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
192/192 ━━━━━━━━━━━━━━━━━━━━ 4s 15ms/step - accuracy: 0.7990 - loss: 0.7140 - val_accuracy: 0.9495 - val_loss: 0.1838
Epoch 2/10
192/192 ━━━━━━━━━━━━━━━━━━━━ 4s 20ms/step - accuracy: 0.9548 - loss: 0.1602 - val_accuracy: 0.9618 - val_loss: 0.1274
Epoch 3/10
192/192 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step - accuracy: 0.9718 - loss: 0.0971 - val_accuracy: 0.9684 - val_loss: 0.1091
Epoch 4/10
192/192 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - accuracy: 0.9806 - loss: 0.0686 - val_accuracy: 0.9705 - val_loss: 0.0958
Epoch 5/10
192/192 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - accuracy: 0.9863 - loss: 0.0487 - val_accuracy: 0.9750 - val_loss: 0.0859
Epoch 6/10
192/192 ━━━━━━━━━━━━━━━━━━━━ 5s 13ms/step - accuracy: 0.9903 - loss: 0.0376 - val_accuracy: 0.9739 - val_loss: 0.0868
Epoch 7/10
192/192 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - accuracy: 0.9917 - loss: 0.0293 - val_accuracy: 0.9726 - val_loss: 0.0937
Epoch 8/10
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - accuracy: 0.9941 - loss: 0.0222 - val_accuracy: 0.9772 - val_loss: 0.0770
Epoch 9/10
192/192 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - accuracy: 0.9963 - loss: 0.0158 - val_accuracy: 0.9779 - val_loss: 0.0787
Epoch 10/10
192/192 ━━━━━━━━━━━━━━━━━━━━ 5s 12ms/step - accuracy: 0.9976 - loss: 0.0115 - val_accuracy: 0.9781 - val_loss: 0.0817
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9765 - loss: 0.0787
Test results - Loss: 0.07009122520685196 - Accuracy: 0.9800000190734863
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step
```



Result:

A handwritten digit's recognition with MNIST dataset is built and the output is 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.

Code:

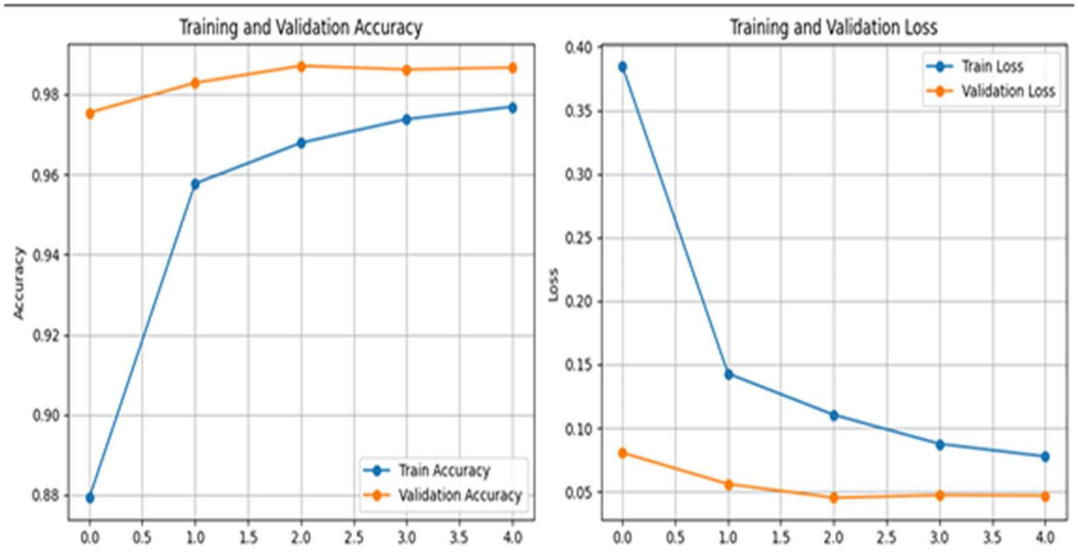
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt
import numpy as np

(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_categorical_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"\n Test 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.plot(history.history['val_accuracy'], label='Validation Accuracy', marker='o')
plt.title('Training 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('Training 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))
```

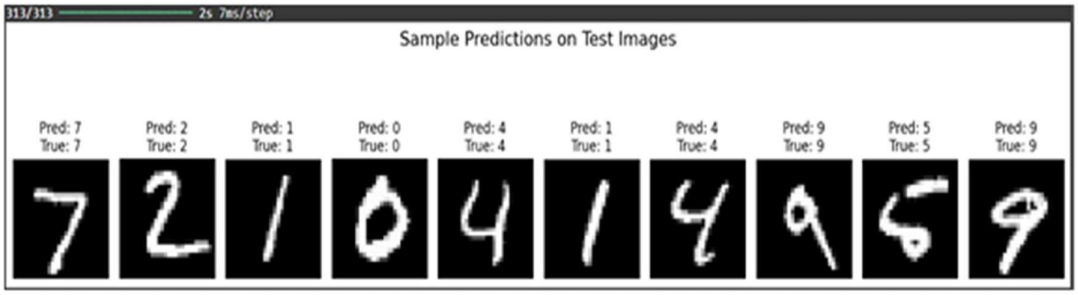
```
for i in range(num_samples):
    plt.subplot(1, num_samples, i + 1)
    plt.imshow(test_images[i].reshape(28, 28), cmap='gray')
    plt.title(f"Pred: {predicted_labels[i]}\nTrue: {test_labels[i]}")
    plt.axis('off')
plt.suptitle("Sample Predictions on Test Images", fontsize=16)
plt.show()
```

Output:

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a lay
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/5
750/750 ━━━━━━━━━━━━━━━━━━━━ 38s 48ms/step - accuracy: 0.7641 - loss: 0.7198 - val_accuracy: 0.9754 - val_loss: 0.0806
Epoch 2/5
750/750 ━━━━━━━━━━━━━━━━━━━━ 41s 49ms/step - accuracy: 0.9542 - loss: 0.1575 - val_accuracy: 0.9828 - val_loss: 0.0559
Epoch 3/5
750/750 ━━━━━━━━━━━━━━━━━━━━ 41s 49ms/step - accuracy: 0.9667 - loss: 0.1147 - val_accuracy: 0.9871 - val_loss: 0.0452
Epoch 4/5
750/750 ━━━━━━━━━━━━━━━━━━━━ 41s 49ms/step - accuracy: 0.9737 - loss: 0.0872 - val_accuracy: 0.9862 - val_loss: 0.0472
Epoch 5/5
750/750 ━━━━━━━━━━━━━━━━━━━━ 41s 49ms/step - accuracy: 0.9753 - loss: 0.0801 - val_accuracy: 0.9867 - val_loss: 0.0468
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - accuracy: 0.9845 - loss: 0.0458
```




```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step
```



Result:

A CNN using Keras/TensorFlow to recognize and classify handwritten digits from the MNIST dataset with high accuracy is built and the output is verified.

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.

Code:

```
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.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 an 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]}\nActual: {class_names[true_label]}")
        plt.show()
```

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 ━━━━━━━━━━━━━━━━━━━━ 51s 79ms/step - accuracy: 0.3104 - loss: 1.8651 - val_accuracy: 0.5242 - val_loss: 1.3433
Epoch 2/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 51s 81ms/step - accuracy: 0.5359 - loss: 1.3072 - val_accuracy: 0.5898 - val_loss: 1.1679
Epoch 3/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 88s 91ms/step - accuracy: 0.5960 - loss: 1.1372 - val_accuracy: 0.6009 - val_loss: 1.1314
Epoch 4/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 50s 81ms/step - accuracy: 0.6345 - loss: 1.0355 - val_accuracy: 0.6460 - val_loss: 1.0102
Epoch 5/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 84s 83ms/step - accuracy: 0.6662 - loss: 0.9524 - val_accuracy: 0.6433 - val_loss: 1.0243
Epoch 6/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 51s 82ms/step - accuracy: 0.6868 - loss: 0.8809 - val_accuracy: 0.6774 - val_loss: 0.9246
Epoch 7/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 49s 78ms/step - accuracy: 0.7089 - loss: 0.8233 - val_accuracy: 0.6843 - val_loss: 0.9061
Epoch 8/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 49s 78ms/step - accuracy: 0.7337 - loss: 0.7592 - val_accuracy: 0.6893 - val_loss: 0.8919
Epoch 9/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 83s 80ms/step - accuracy: 0.7495 - loss: 0.7176 - val_accuracy: 0.7007 - val_loss: 0.8667
Epoch 10/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 83s 82ms/step - accuracy: 0.7608 - loss: 0.6766 - val_accuracy: 0.6884 - val_loss: 0.9299
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 121ms/step
```

Result:

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. is successfully built and the output is verified.

Ex No: 4 TRANSFER LEARNING WITH CNN AND VISUALIZATION

Aim:

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

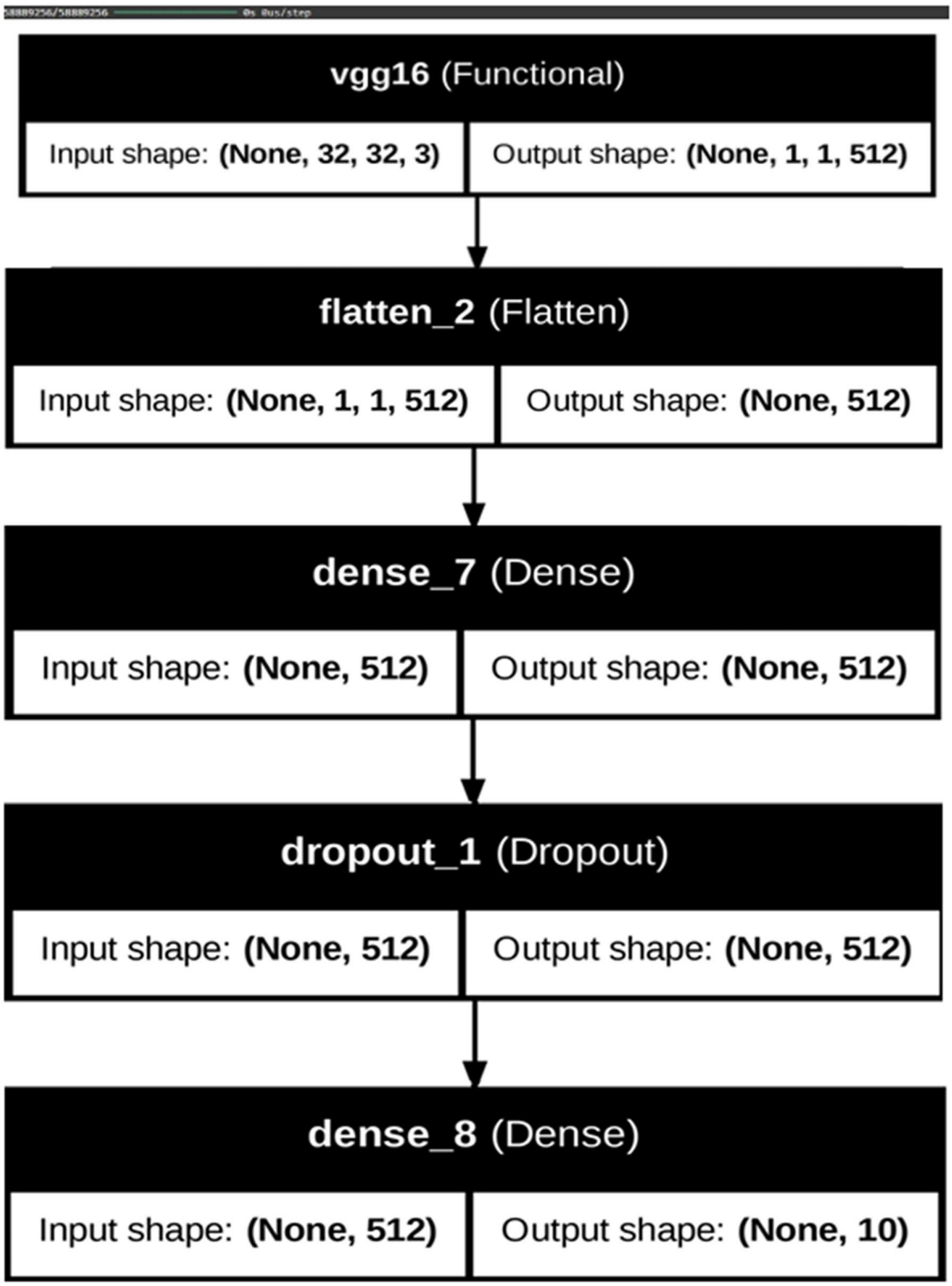
Procedure:

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.

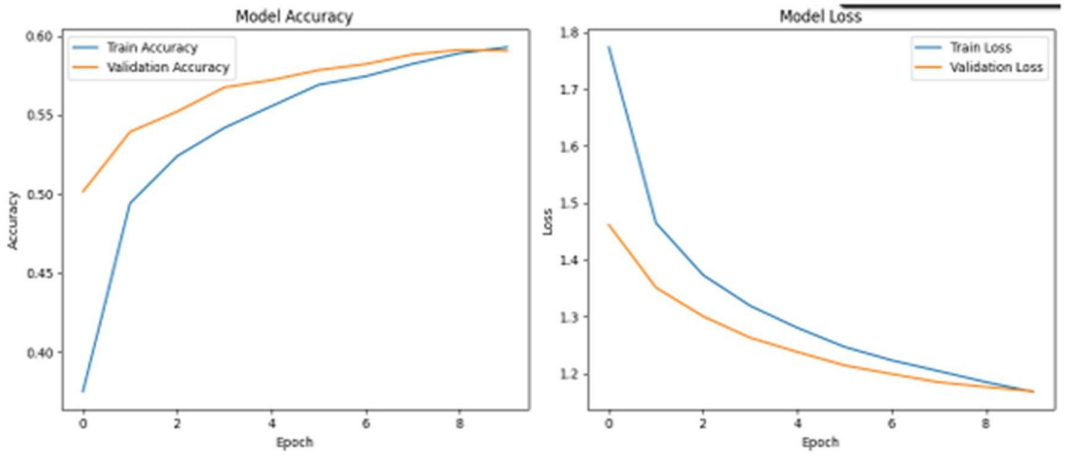
```
conda install -c conda-forge python-graphviz -y
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import plot_model
import matplotlib.pyplot as plt
import numpy as np
(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_crossentropy',
          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()

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


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}%')

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 = ['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()
```

Output:


58889256/58889256 0s 0us/step
vgg16 (Functional)
Input shape: (None, 32, 32, 3)
Output shape: (None, 1, 1, 512)
flatten_2 (Flatten)
Input shape: (None, 1, 1, 512)
Output shape: (None, 512)
dense_7 (Dense)
Input shape: (None, 512)
Output shape: (None, 512)
dropout_1 (Dropout)
Input shape: (None, 512)
Output shape: (None, 512)
dense_8 (Dense)
Input shape: (None, 512)
Output shape: (None, 10)

```
Epoch 1/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 580s 463ms/step - accuracy: 0.2851 - loss: 2.0021 - val_accuracy: 0.5015 - val_loss: 1.4614
Epoch 2/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 618s 460ms/step - accuracy: 0.4840 - loss: 1.4918 - val_accuracy: 0.5393 - val_loss: 1.3512
Epoch 3/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 574s 460ms/step - accuracy: 0.5235 - loss: 1.3748 - val_accuracy: 0.5521 - val_loss: 1.3007
Epoch 4/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 624s 461ms/step - accuracy: 0.5373 - loss: 1.3335 - val_accuracy: 0.5674 - val_loss: 1.2630
Epoch 5/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 577s 462ms/step - accuracy: 0.5587 - loss: 1.2804 - val_accuracy: 0.5720 - val_loss: 1.2381
Epoch 6/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 538s 430ms/step - accuracy: 0.5692 - loss: 1.2482 - val_accuracy: 0.5783 - val_loss: 1.2146
Epoch 7/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 600s 461ms/step - accuracy: 0.5717 - loss: 1.2292 - val_accuracy: 0.5821 - val_loss: 1.1994
Epoch 8/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 620s 460ms/step - accuracy: 0.5770 - loss: 1.2138 - val_accuracy: 0.5882 - val_loss: 1.1849
Epoch 9/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 575s 460ms/step - accuracy: 0.5898 - loss: 1.1833 - val_accuracy: 0.5911 - val_loss: 1.1766
Epoch 10/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 624s 461ms/step - accuracy: 0.5937 - loss: 1.1666 - val_accuracy: 0.5905 - val_loss: 1.1690
313/313 ━━━━━━━━━━━━━━━━━━━━ 108s 345ms/step - accuracy: 0.5793 - loss: 1.1799
Test Loss: 1.1821
Test Accuracy: 58.40%
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 251ms/step
```

Result:

A CNN with transfer learning and perform visualization is built and the output is verified.

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.

```
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  # Reduced epochs for quick demonstration
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.randn(5, seq_length, 1)
predictions = model.predict(new_data)
print("Predictions for new data:")
print(predictions)
```

Output:

In [9]:
```
test_loss = model.evaluate(X_test, y_test)
print(f'Test Loss: {test_loss:.4f}')
```

```
7/7 [==============================] - 0s 5ms/step - loss: 0.0241
Test Loss: 0.0241
```

In [10]:
```
y_pred = model.predict(X_test)
r2 = r2_score(y_test, y_pred)
print(f'Test Accuracy (R^2): {r2:.4f}')
```

```
7/7 [==============================] - 0s 5ms/step
Test Accuracy (R^2): 0.9974
```

In [11]:
```
new_data = np.random.randn(5, seq_length, 1)
predictions = model.predict(new_data)
print("Predictions for new data:")
print(predictions)
```

```
1/1 [==============================] - 0s 52ms/step
Predictions for new data:
[[ 1.7613161 ]
 [ 0.40740597]
 [-2.23266   ]
 [-0.6163975 ]
 [-3.7167645 ]]
```

Result:

A recurrent neural network with Keras/TensorFlow is successfully built and the output is verified.

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

```
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:.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:

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 [==============================] - 3s 0us/step
Training...
Epoch 1/2
313/313 [==============================] - 27s 76ms/step - loss: 0.5892 - accuracy: 0.6730 - val_loss: 0.6065 - val_accuracy:
0.6728
Epoch 2/2
313/313 [==============================] - 23s 74ms/step - loss: 0.4183 - accuracy: 0.8093 - val_loss: 0.3735 - val_accuracy:
0.8356
782/782 [==============================] - 14s 18ms/step - loss: 0.3724 - accuracy: 0.8388

Test Accuracy: 0.8388
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb_word_index.json
1641221/1641221 [==============================] - 0s 0us/step
1/1 [==============================] - 0s 298ms/step

Review text: ? please give this one a miss br br ? ? and the rest of the cast ? terrible performances the show is flat flat fla
t 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:

A Recurrent Neural Network (RNN) using Keras/TensorFlow for classifying the sentiment of text data (e.g., movie reviews) as positive or negative is successfully built and the output is verified.

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.

```
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(32, activation='relu')(input_img)
decoded = 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)

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)

n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
    # Display 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)

    # 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()
```

Output:

```
Epoch 45/50
235/235 [==============================] - 2s 9ms/step - loss: 0.0928 - val_loss: 0.0916
Epoch 46/50
235/235 [==============================] - 2s 8ms/step - loss: 0.0928 - val_loss: 0.0916
Epoch 47/50
235/235 [==============================] - 3749s 16s/step - loss: 0.0928 - val_loss: 0.0917
Epoch 48/50
235/235 [==============================] - 2s 10ms/step - loss: 0.0928 - val_loss: 0.0916
Epoch 49/50
235/235 [==============================] - 2s 9ms/step - loss: 0.0927 - val_loss: 0.0916
Epoch 50/50
235/235 [==============================] - 2s 10ms/step - loss: 0.0927 - val_loss: 0.0916
313/313 [==============================] - 1s 3ms/step - loss: 0.0916
313/313 [==============================] - 1s 2ms/step
Test Loss: 0.09159169346094131
Test Accuracy: 0.9713468112244898
```

Result:

A hautoencoders with Keras/TensorFlow is successfully built and the output is verified.

Ex No: 8 OBJECT DETECTION WITH YOLO3

Aim:

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

Procedure:

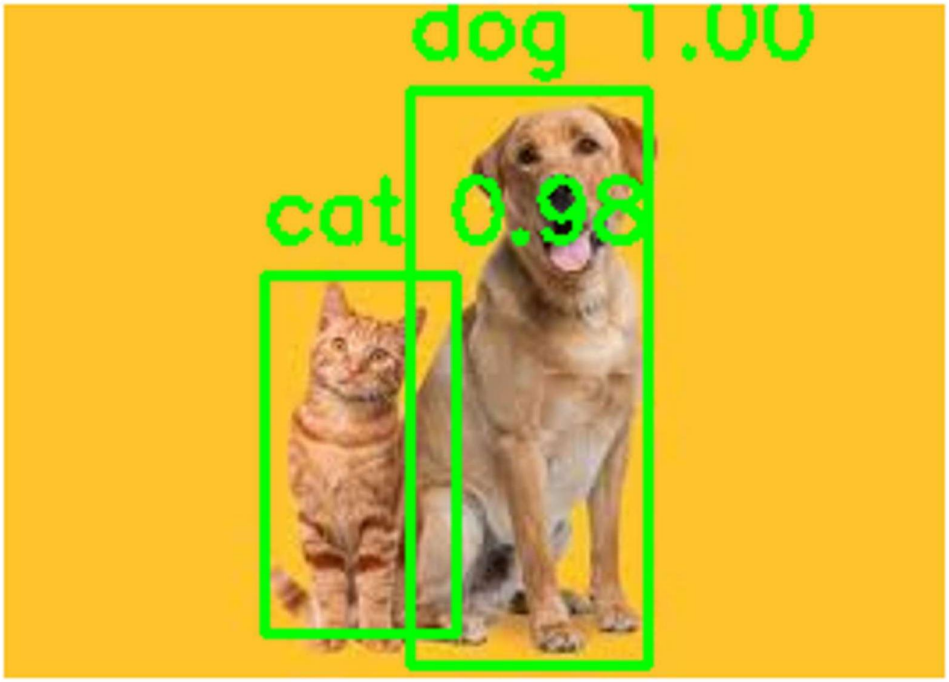
1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a simple neural network model using Keras/TensorFlow.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

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

# Define the paths to the YOLOv3 configuration, weights, and class names files
cfg_file = '/content/yolov3.cfg'
weight_file = '/content/yolov3.weights'
namesfile = '/content/coco.names'

# Load the YOLOv3 model
net = cv2.dnn.readNet(weight_file, cfg_file)

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

# Load an image for object detection
image_path = '/content/hit.jpg'
image = cv2.imread(image_path)

# Get the height and width of the image
height, width = image.shape[:2]

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

# Get the names of the output layers
layer_names = net.getUnconnectedOutLayersNames()

# Run forward pass
```

```
outs = net.forward(layer_names)

# Initialize lists to store detected objects' information
class_ids = []
confidences = []
boxes = []

# Define a confidence threshold for object detection
conf_threshold = 0.5

# Loop over the detections
for out in outs:
    for detection in out:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > conf_threshold:
            # Object detected
            center_x = int(detection[0] * width)
            center_y = int(detection[1] * height)
            w = int(detection[2] * width)
            h = int(detection[3] * height)

            # Rectangle coordinates
            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])
```

```
# Apply non-maximum suppression to eliminate overlapping boxes
nms_threshold = 0.4
indices = cv2.dnn.NMSBoxes(boxes, confidences, conf_threshold, nms_threshold)

# Draw bounding boxes and labels on the image
for i in indices.flatten():  # flatten for compatibility
    x, y, w, h = boxes[i]
    label = str(classes[class_ids[i]])
    confidence = confidences[i]

    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)

# Display the result in Jupyter Notebook
plt.figure(figsize=(10, 8))
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

Output:



Result:
An object detection model with YOLO3 using Keras/TensorFlow is successfully built and the output is verified.

Ex No: 9 BUILD GENERATIVE ADVERSARIAL NEURAL NETWORK

Aim:

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

Procedure:

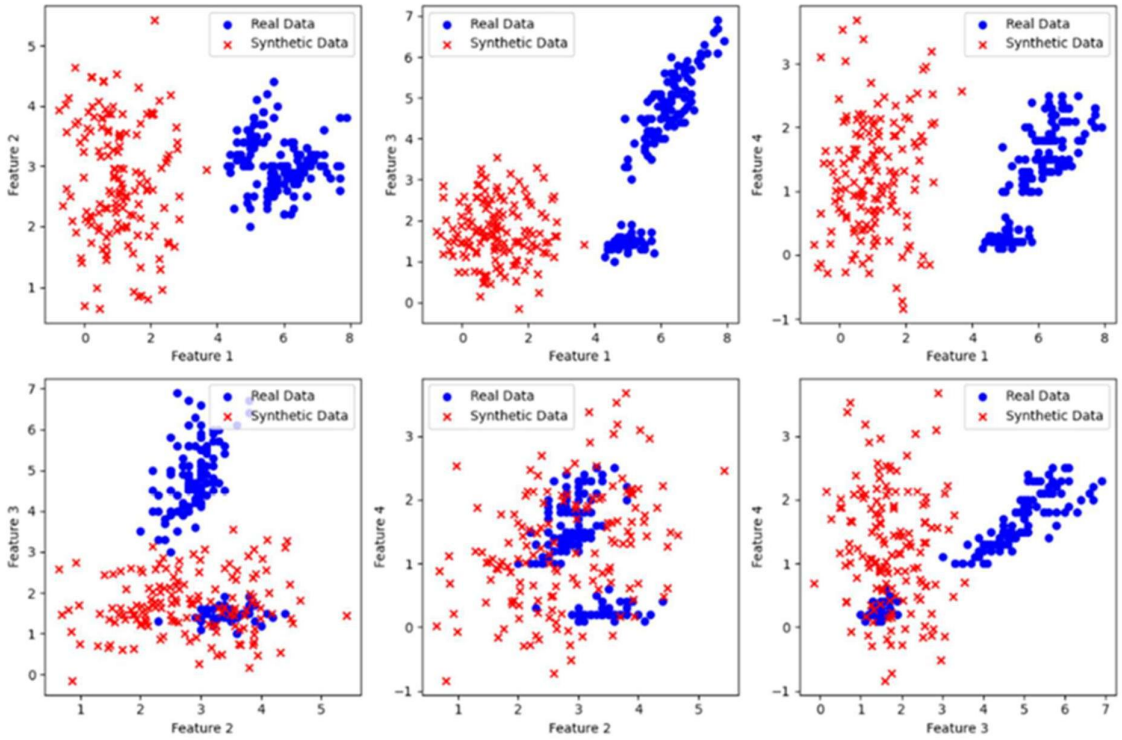
1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a simple neural network model using Keras/TensorFlow.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

```
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:



Feature 2 Feature 2 Feature 3

Result:

A generative adversarial neural network using Keras/TensorFlow is successfully built and the output is verified.

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.

Code:

```
from ultralytics import YOLO
import cv2
import math
import numpy as np
import time

# Load YOLO model
model = YOLO('yolov8n.pt')

# Video path
video_path = 'Overtake.mp4'
cap = cv2.VideoCapture(video_path)

# Configuration thresholds
CENTER_X_MARGIN = 0.20       # Vehicles within ±20% of center X considered front vehicles
MIN_Y_RATIO = 0.5            # Consider boxes with center Y > 50% of frame height
CLOSE_DIST_RATIO = 0.30      # Distance < 30% of frame height -> unsafe
APPROACH_SPEED_PIX = 8.0     # Pixels per frame considered fast approach
APPROACH_DIST_RATIO = 0.45   # Distance < 45% of frame height + fast approach -> unsafe

# Tracking initialization
prev_centroids = []          # List of (cx, cy)
prev_ids = []                # Corresponding IDs
next_id = 0                  # Synthetic ID generator
max_match_dist = 80          # Max distance for matching centroids between frames
font = cv2.FONT_HERSHEY_SIMPLEX


# Helper function for centroid matching
def match_centroids(prev, curr, prev_ids):
    global next_id
    mapping = {}
    used_prev = set()

    for i, c in enumerate(curr):
        best_j, best_d = None, 1e9
        for j, p in enumerate(prev):
            if j in used_prev:
                continue
            d = math.hypot(c[0] - p[0], c[1] - p[1])
            if d < best_d:
                best_d = d
```

```
            best_j = j

        if best_j is not None and best_d < max_match_dist:
            mapping[i] = prev_ids[best_j]
            used_prev.add(best_j)
        else:
            mapping[i] = next_id
            next_id += 1

    return mapping


# Main processing loop
last_time = time.time()

while True:
    ret, frame = cap.read()
    if not ret:
        break

    frame_h, frame_w = frame.shape[:2]
    center_x = frame_w // 2

    results = model(frame)
    boxes = results[0].boxes.xyxy.cpu().numpy() if hasattr(results[0].boxes, "xyxy") else np.array([])

    detected = []
    for b in boxes:
        x1, y1, x2, y2 = b[:4]
        cx, cy = (x1 + x2) / 2, (y1 + y2) / 2
        detected.append({'box': (int(x1), int(y1), int(x2), int(y2)), 'cx': cx, 'cy': cy})

    # Prepare centroid lists
    curr_centroids = [(d['cx'], d['cy']) for d in detected]

    # Match with previous centroids for stable ID assignment
    if prev_centroids:
        mapping = match_centroids(prev_centroids, curr_centroids, prev_ids)
    else:
        mapping = {i: i for i in range(len(curr_centroids))}
    next_id = len(curr_centroids)

    curr_ids = [mapping[i] for i in range(len(curr_centroids))]

    # Build a dict for current frame IDs
    curr_dict = {}
    for i, d in enumerate(detected):
        cid = curr_ids[i]
        x1, y1, x2, y2 = d['box']
        cx, cy = d['cx'], d['cy']
        bbox_h = y2 - y1
```

```
        curr_dict[cid] = {'box': d['box'], 'cx': cx, 'cy': cy, 'bbox_h': bbox_h}

    # Distance and approach checks
    approach_flag = False
    unsafe_flag = False
    front_id = None
    front_min_dist = float('inf')

    # Find "front" vehicle near bottom-center
    for cid, info in curr_dict.items():
        cx, cy = info['cx'], info['cy']
        if cy > frame_h * MIN_Y_RATIO and abs(cx - center_x) < frame_w * CENTER_X_MARGIN:
            dist_to_bottom = frame_h - cy
            if dist_to_bottom < front_min_dist:
                front_min_dist = dist_to_bottom
                front_id = cid

    # If a front vehicle is identified, check safety
    if front_id is not None:
        if front_min_dist < frame_h * CLOSE_DIST_RATIO:
            unsafe_flag = True

        if prev_centroids and front_id in prev_ids:
            idx_prev = prev_ids.index(front_id)
            prev_cx, prev_cy = prev_centroids[idx_prev]
            prev_dist = frame_h - prev_cy
            curr_cx = curr_dict[front_id]['cx']
            curr_cy = curr_dict[front_id]['cy']
            curr_dist = frame_h - curr_cy
            approach_speed = prev_dist - curr_dist

            if approach_speed > APPROACH_SPEED_PIX and curr_dist < frame_h *
APPROACH_DIST_RATIO:
                approach_flag = True
                unsafe_flag = True

    # Draw bounding boxes
    for cid, info in curr_dict.items():
        x1, y1, x2, y2 = info['box']
        if cid == front_id:
            color = (0, 0, 255) if unsafe_flag else (0, 255, 0)
        else:
            color = (200, 200, 200)
        cv2.rectangle(frame, (x1, y1), (x2, y2), color, 2)
        cv2.putText(frame, f"ID:{cid}", (x1, y1 - 6), font, 0.5, color, 1)

    # Display status
    status = "SAFE" if not unsafe_flag else "UNSAFE"
    status_color = (0, 255, 0) if not unsafe_flag else (0, 0, 255)
    cv2.putText(frame, f"{status} ALERT", (30, 40), font, 1.1, status_color, 3)
```

```
            if front_id is not None:
                cv2.putText(frame, f"FrontID:{front_id} dist_px:{int(front_min_dist)}",
                            (30, 80), font, 0.7, (255, 255, 0), 2)

                if prev_centroids and front_id in prev_ids:
                    idx_prev = prev_ids.index(front_id)
                    prev_cx, prev_cy = prev_centroids[idx_prev]
                    prev_dist = int(frame_h - prev_cy)
                    curr_dist = int(frame_h - curr_dict[front_id]['cy'])
                    approach_speed = prev_dist - curr_dist
                    cv2.putText(frame, f"approach_px/frame: {int(approach_speed)}",
                                (30, 110), font, 0.6, (255, 200, 0), 1)

            # Show frame
            cv2.imshow("Improved Unsafe Detection", frame)
            if cv2.waitKey(120) & 0xFF == ord('q'):
                break

            # Update previous frame data
            prev_centroids = curr_centroids
            prev_ids = curr_ids

        cap.release()
        cv2.destroyAllWindows()
```

OUTPUT:

RESULT:

The implemented deep learning model using pretrained VGG16 successfully extracted features from signatures and classified them as genuine or forged. The system achieved high accuracy on the test dataset, effectively distinguishing between authentic and forged signatures in real time.