RAJALAKSHMI ENGINEERING COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CS19P18- DEEP LEARNING CONCEPTS LABORATORY

LAB MANUAL

FINAL YEAR

SEVENTH SEMESTER

2025- 2026

ODD SEMESTER

RAJALAKSHMI ENGINEERING COLLEGE

(An Autonomous Institution)

RAJALAKSHMI NAGAR, THANDALAM- 602 105

BONAFIDE CERTIFICATE

NAME: Madhan Shankar G **BRANCH/SECTION:** CSE I

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

REGISTER NO: 2116220701149

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**

LIST OF EXPERIMENTS

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

RAJALAKSHMI ENGINEERING COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CS19P18- DEEP LEARNING CONCEPTS LABORATORY

LAB PLAN

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

HARDWARE AND SOFTWARE REQUIREMENTS

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

Course Outcomes (COs)

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

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

CO-PO –PSO matrices of course

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

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

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

INSTALLATION AND CONFIGURATION OF TENSORFLOW

Aim:

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

Procedure:

1. Download Anaconda Navigator and install.

2. Open Anaconda prompt

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

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

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

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

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

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

9. Lastly, install jupyter notebook
   pip install jupyter notebook

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

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

12. Give the name to the file

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

Useful Learning Resources:

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

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

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

Aim:

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

Procedure:

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

Useful Learning Resources:

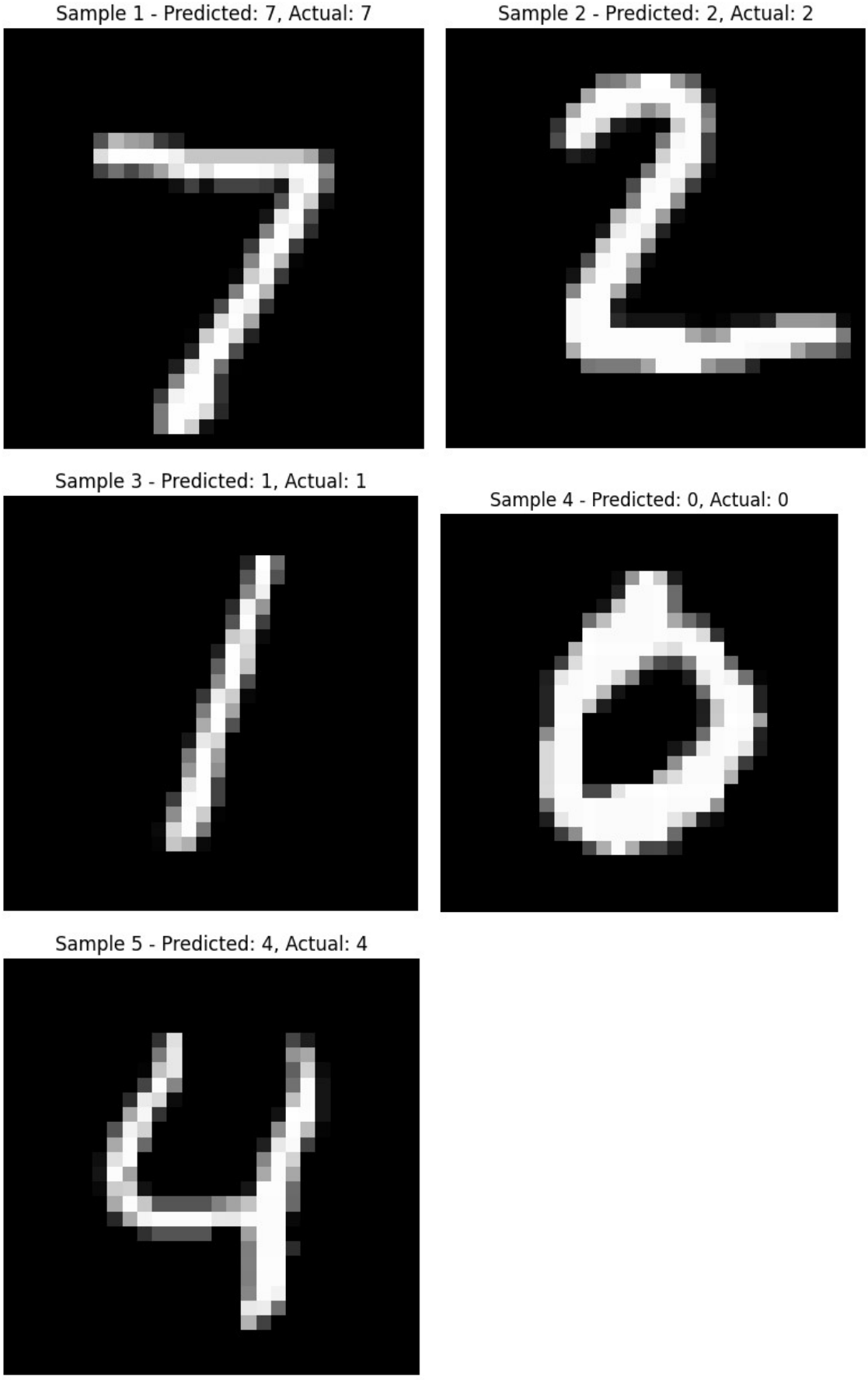
1.https://www.analyticsvidhya.com/blog/2022/07/handwritten-digit-recognition-using-tensorflow/

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

```
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:



Result:

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

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

Aim:

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

Procedure:

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

Useful Learning Resources:

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

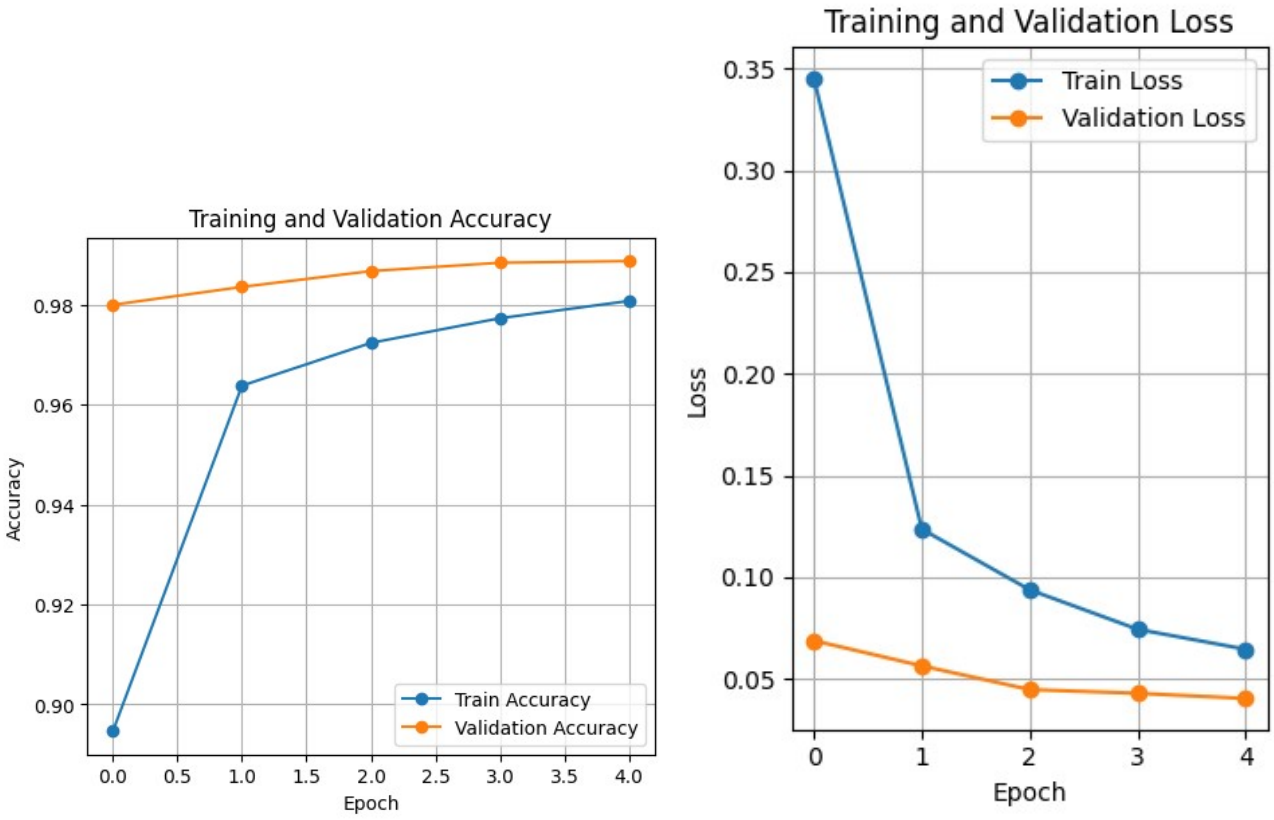
2.https://www.analyticsvidhya.com/blog/2021/06/building-a-convolutional-neural-network-using-tensorflow-keras/

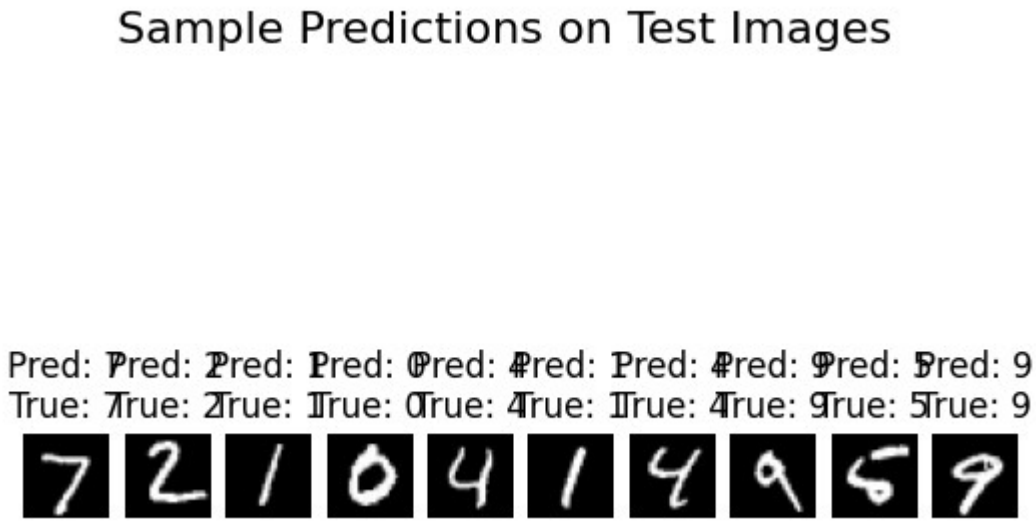
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt
import numpy as np

(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:





Result:

A Convolutional Neural Network (CNN) using Keras/TensorFlow to recognize and classify handwritten digits from the MNIST dataset with high accuracy is successfully built and the output is verfied.

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

Aim:

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

Procedure:

1. Download and load the CIFAR-10 dataset using Keras/TensorFlow.
2. Visualize and analyze sample images from the dataset.

3, Preprocess the data:

- Normalize the pixel values (divide by 255)
- Convert class labels to one-hot encoded format

4. Build a CNN model using Keras/TensorFlow:
    - Include convolutional, pooling, flatten, and dense layers.
5. Compile the model with suitable loss function and optimizer.
6. Train the model using training data and validate using test data.
7. Evaluate the model using accuracy and loss on test dataset.
8. Perform predictions on new/unseen CIFAR-10 images.

9 Visualize prediction results with sample images and predicted labels.

Useful Learning Resources:

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

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

```
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:

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 verfied.

Ex No: 4 TRANSFER LEARNING WITH CNN AND VISUALIZATION

Aim:

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

Procedure:

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

Useful Learning Resources:

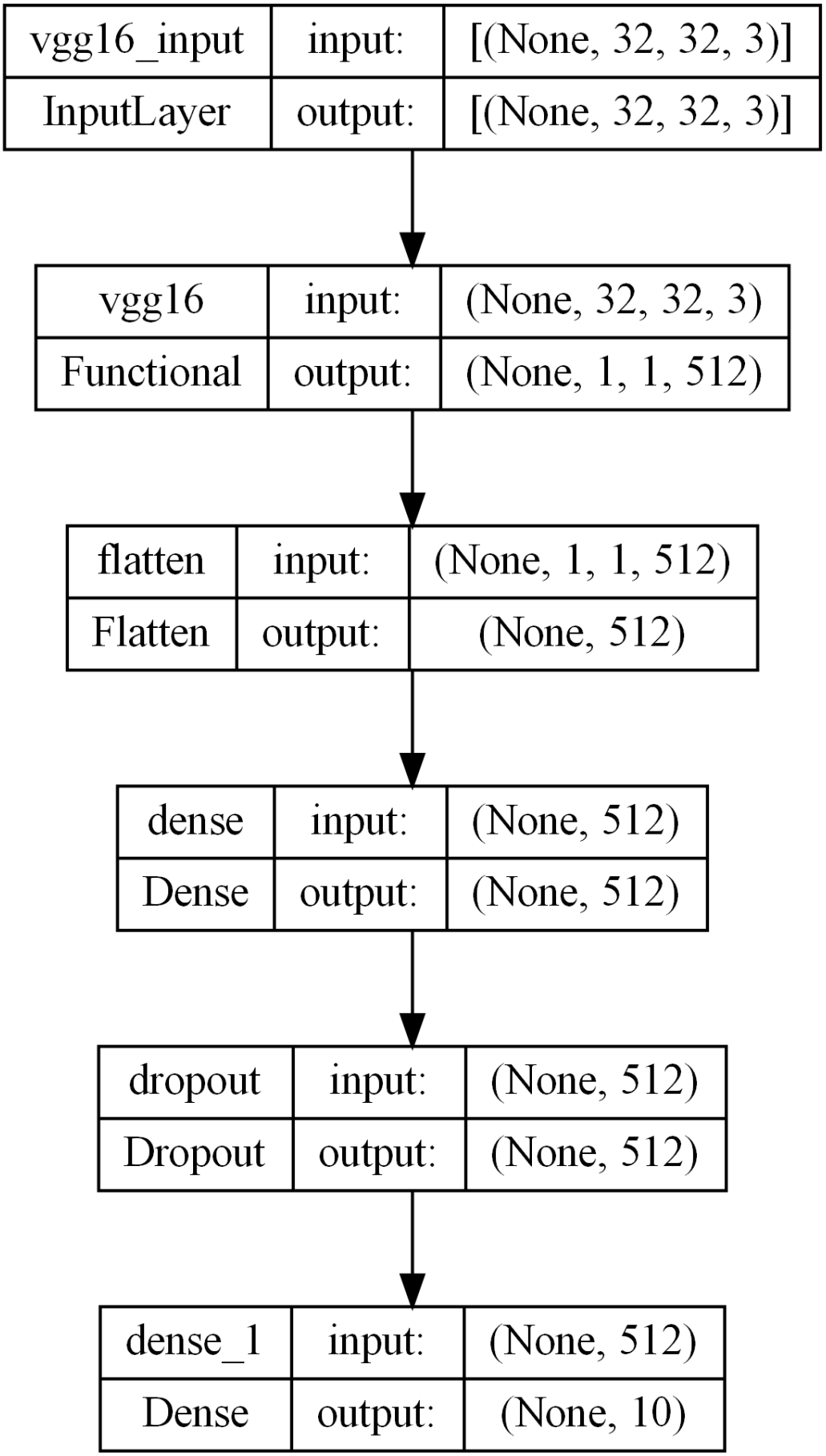
1.https://medium.com/analytics-vidhya/car-brand-classification-using-vgg16-transfer-learning-f219a0f09765

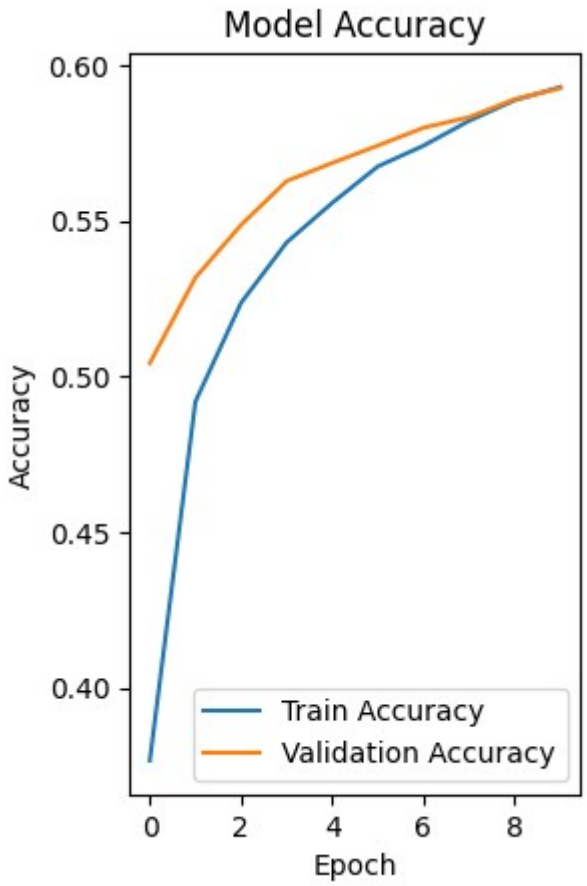
2. https://www.kaggle.com/code/kasmithh/transfer-learning-using-keras-vgg-16

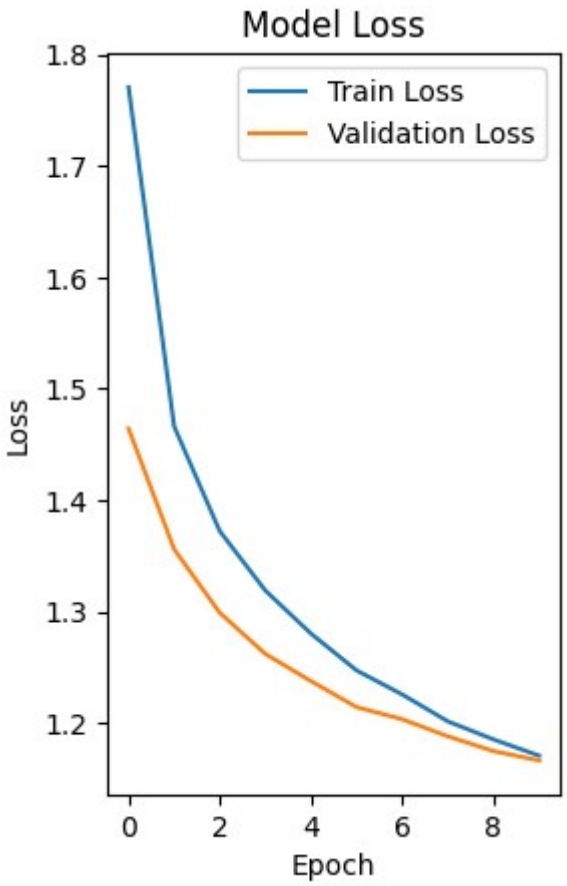
```
conda install -c conda-forge python-graphviz -y
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.datasets 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')
```

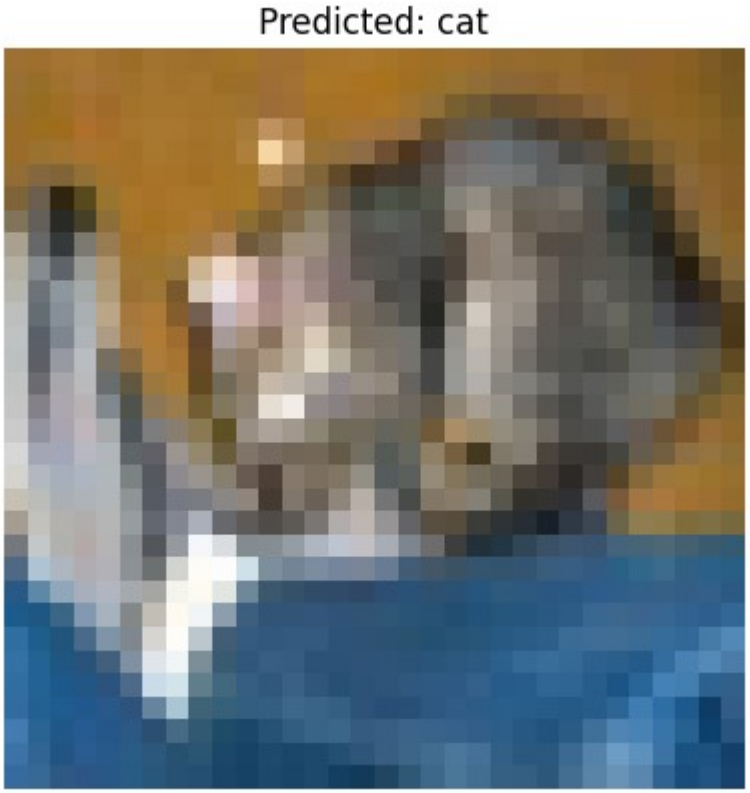
```
plt.ylabel('Accuracy')
plt.legend()


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:


vgg16_input | input: | [(None, 32, 32, 3)]
InputLayer | output: | [(None, 32, 32, 3)]
vgg16 | input: | (None, 32, 32, 3)
Functional | output: | (None, 1, 1, 512)
flatten | input: | (None, 1, 1, 512)
Flatten | output: | (None, 512)
dense | input: | (None, 512)
Dense | output: | (None, 512)
dropout | input: | (None, 512)
Dropout | output: | (None, 512)
dense_1 | input: | (None, 512)
Dense | output: | (None, 10)







Result:

A convolutional neural network with transfer learning and perform visualization is successfully built and the output is verfied.

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

Aim:

To build a recurrent neural network with Keras/TensorFlow.

Procedure:

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

Useful Learning Resources:

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

```
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 verfied.

EX NO: 6 SENTIMENT CLASSIFICATION OF TEXT USING RNN

Aim:

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

Procedure:

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

Useful Learning Resources:

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

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

```
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:

```
Test Accuracy: 0.8502
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb_word_index.json
1641221/1641221 [==============================] - 0s 0us/step
```

In [6]:

```
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 ")
```

```
1/1 [==============================] - 0s 244ms/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 wor
k 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 verfied.

Ex No: 7 BUILD AUTOENCODERS WITH KERAS/TENSORFLOW

Aim:

To build autoencoders with Keras/TensorFlow.

Procedure:

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

Useful Learning Resources:

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

```
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:

```
Test Loss: 0.09151389449834824
Test Accuracy: 0.9713761479591837
```

Result:

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

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

Aim:

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

Procedure:

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

Useful Learning Resources:

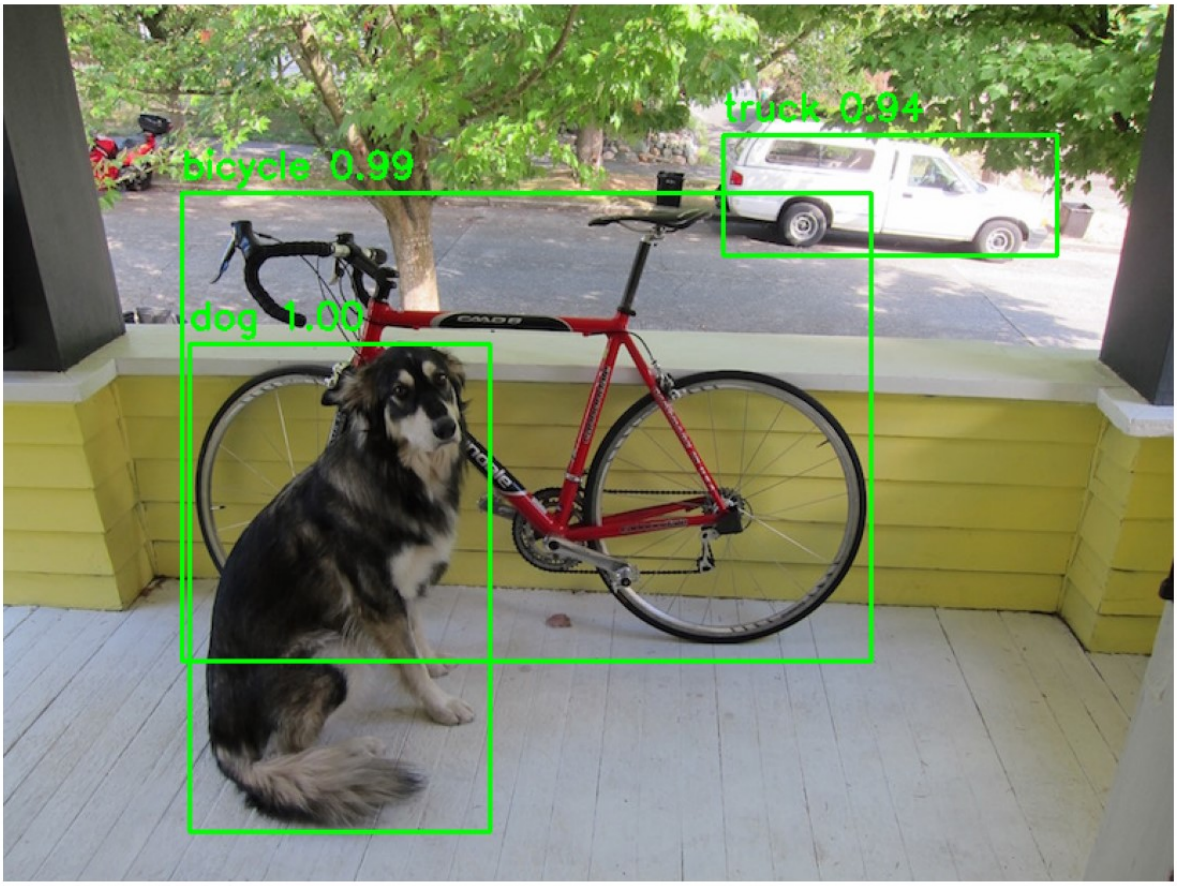
1. https://machinelearningmastery.com/how-to-perform-object-detection-with-yolov3-in-keras/
2. https://www.kaggle.com/code/roobansappani/yolo-v3-object-detection-using-keras

```
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 verfied.

Ex No: 9 BUILD GENERATIVE ADVERSARIAL NEURAL NETWORK

Aim:

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

Procedure:

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

Useful Learning Resources:

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

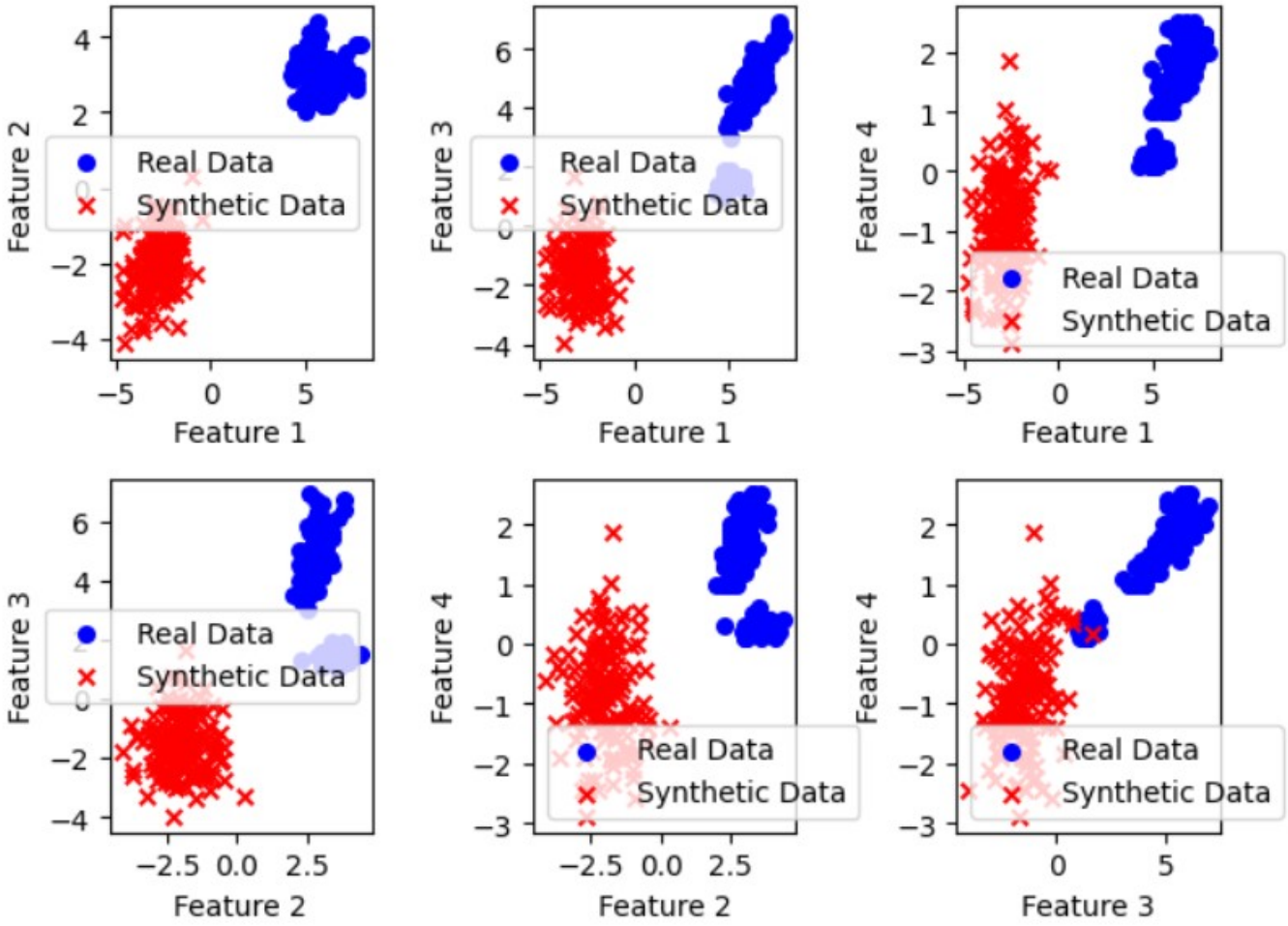
2.https://www.analyticsvidhya.com/blog/2021/06/a-detailed-explanation-of-gan-with-implementation-using-tensorflow-and-keras/

```
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:



Result:

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

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

Aim:

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

Instructions:

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

Mini Project Titles;

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

LAB VIVA QUESTIONS

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

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