RAJALAKSHMI ENGINEERING COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CS19P18- DEEP LEARNING CONCEPTS LABORATORY

LAB MANUAL

FINAL YEAR

SEVENTH SEMESTER

2025- 2026

ODD SEMESTER

LIST OF EXPERIMENTS

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

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.

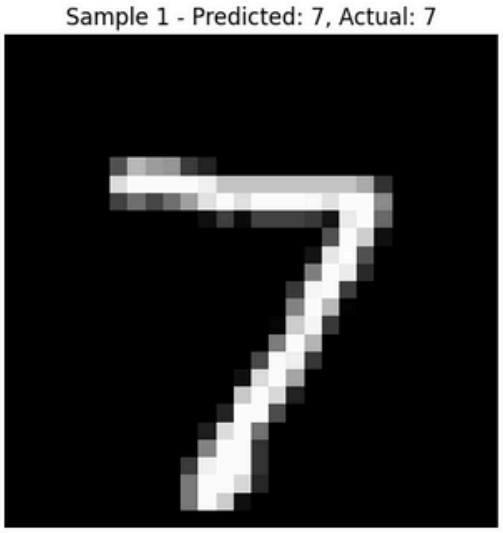
Useful Learning Resources:

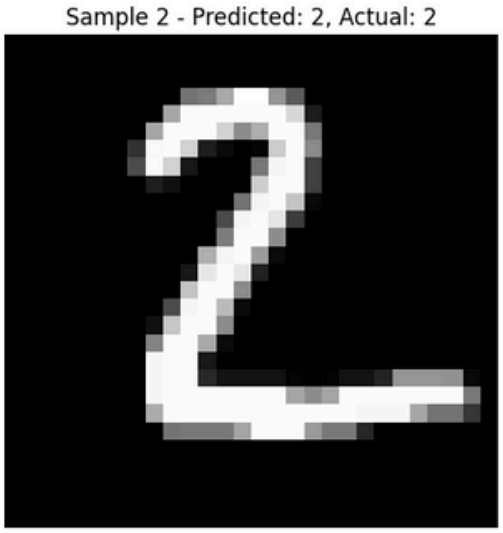
1.https://www.analyticsvidhya.com/blog/2022/07/handwritten-digit-recognition-using-tensorflow/

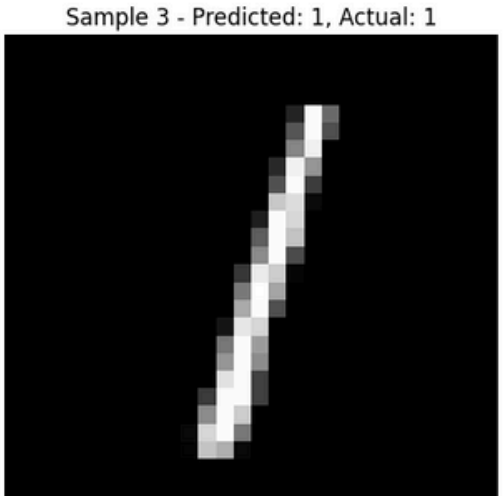
2. https://www.milindsoorya.com/blog/handwritten-digits-classification

```
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
from tensorflow.keras.callbacks import EarlyStopping


np.random.seed(42)
tf.random.set_seed(42)

feature_vector_length = 784
num_classes = 10

(X_train, Y_train), (X_test, Y_test) = mnist.load_data()

X_train = X_train.reshape(X_train.shape[0], feature_vector_length).astype('float32') / 255
X_test = X_test.reshape(X_test.shape[0], feature_vector_length).astype('float32') / 255

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

input_shape = (feature_vector_length,)

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'])

early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

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

test_loss, test_accuracy = model.evaluate(X_test, Y_test, verbose=1)
print(f'Test results - Loss: {test_loss:.4f} - Accuracy: {test_accuracy:.4f}')

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

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

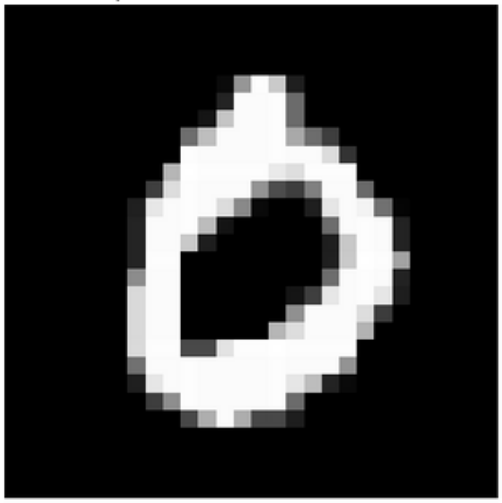
```
Epoch 1/10
192/192 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.8060 - loss: 0.7126 - val_accuracy: 0.9487 - val_loss: 0.1745
Epoch 2/10
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.9541 - loss: 0.1578 - val_accuracy: 0.9616 - val_loss: 0.1285
Epoch 3/10
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.9702 - loss: 0.1027 - val_accuracy: 0.9689 - val_loss: 0.1087
Epoch 4/10
192/192 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.9792 - loss: 0.0742 - val_accuracy: 0.9712 - val_loss: 0.0998
Epoch 5/10
192/192 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - accuracy: 0.9842 - loss: 0.0555 - val_accuracy: 0.9728 - val_loss: 0.0933
Epoch 6/10
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.9884 - loss: 0.0411 - val_accuracy: 0.9731 - val_loss: 0.0925
Epoch 7/10
192/192 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - accuracy: 0.9920 - loss: 0.0304 - val_accuracy: 0.9734 - val_loss: 0.0946
Epoch 8/10
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - accuracy: 0.9950 - loss: 0.0232 - val_accuracy: 0.9732 - val_loss: 0.0986
Epoch 9/10
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.9966 - loss: 0.0183 - val_accuracy: 0.9750 - val_loss: 0.0934
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9720 - loss: 0.0970
Test results - Loss: 0.0829 - Accuracy: 0.9760
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
```







Sample 4 - Predicted: 0, Actual: 0



Sample 5 - Predicted: 4, Actual: 4

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.

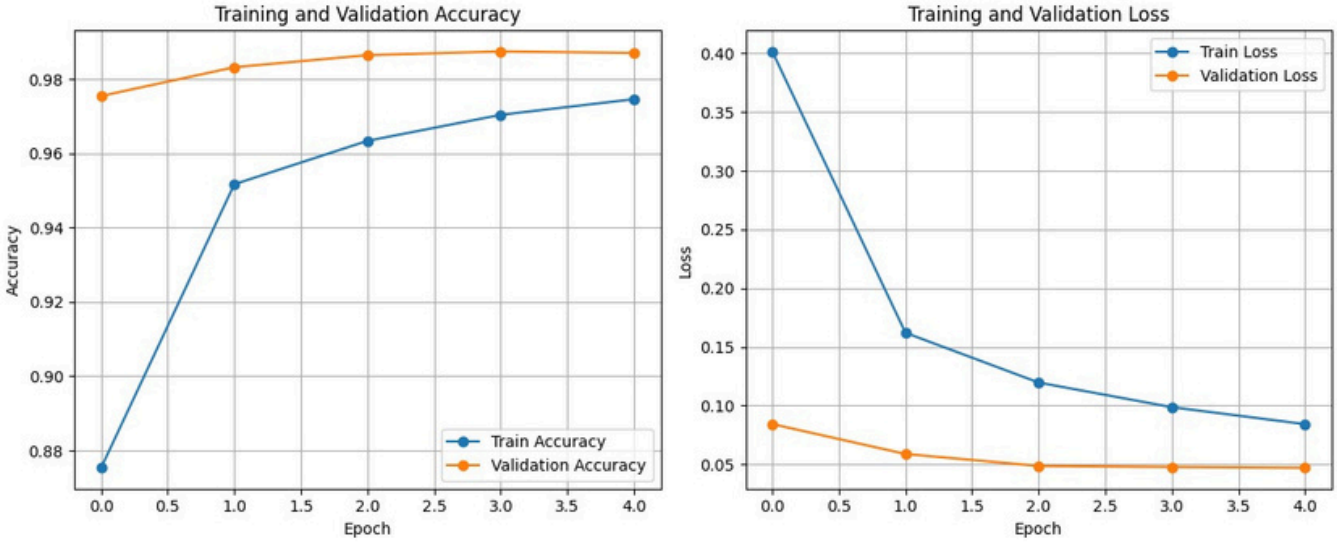
Useful Learning Resources:

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

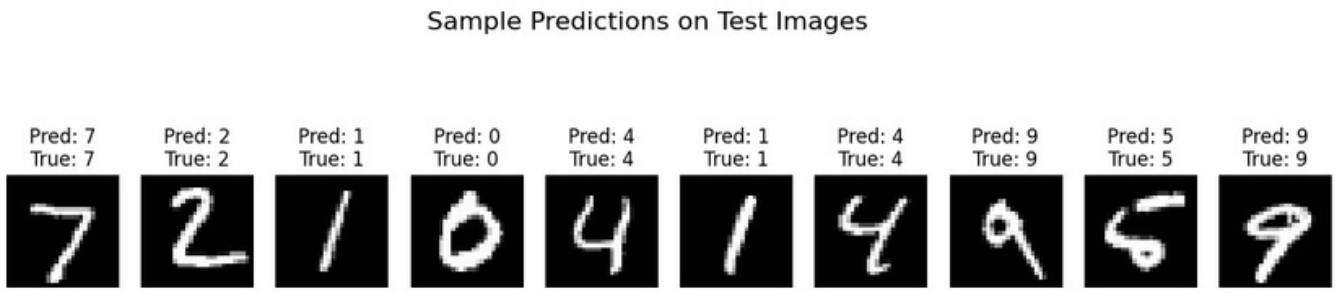
2.https://www.analyticsvidhya.com/blog/2021/06/building-a-convolutional-neural-network-using-tensorflow-keras/

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

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy', marker='o')
plt.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()
```

```
Epoch 1/5
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass an `input_shape`/`
super().__init__(activity_regularizer=activity_regularizer, **kwargs)
750/750 ━━━━━━━━━━━━━━━━━━━━ 25s 31ms/step - accuracy: 0.7633 - loss: 0.7330 - val_accuracy: 0.9754 - val_loss: 0.0843
Epoch 2/5
750/750 ━━━━━━━━━━━━━━━━━━━━ 40s 30ms/step - accuracy: 0.9479 - loss: 0.1751 - val_accuracy: 0.9832 - val_loss: 0.0588
Epoch 3/5
750/750 ━━━━━━━━━━━━━━━━━━━━ 23s 30ms/step - accuracy: 0.9610 - loss: 0.1246 - val_accuracy: 0.9864 - val_loss: 0.0486
Epoch 4/5
750/750 ━━━━━━━━━━━━━━━━━━━━ 41s 31ms/step - accuracy: 0.9688 - loss: 0.1004 - val_accuracy: 0.9874 - val_loss: 0.0477
Epoch 5/5
750/750 ━━━━━━━━━━━━━━━━━━━━ 41s 31ms/step - accuracy: 0.9734 - loss: 0.0871 - val_accuracy: 0.9870 - val_loss: 0.0470
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9849 - loss: 0.0470

Test accuracy: 0.9888
Test loss: 0.0391
```




```
WARNING:tensorflow:5 out of the last 5 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distribu
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step
```



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:

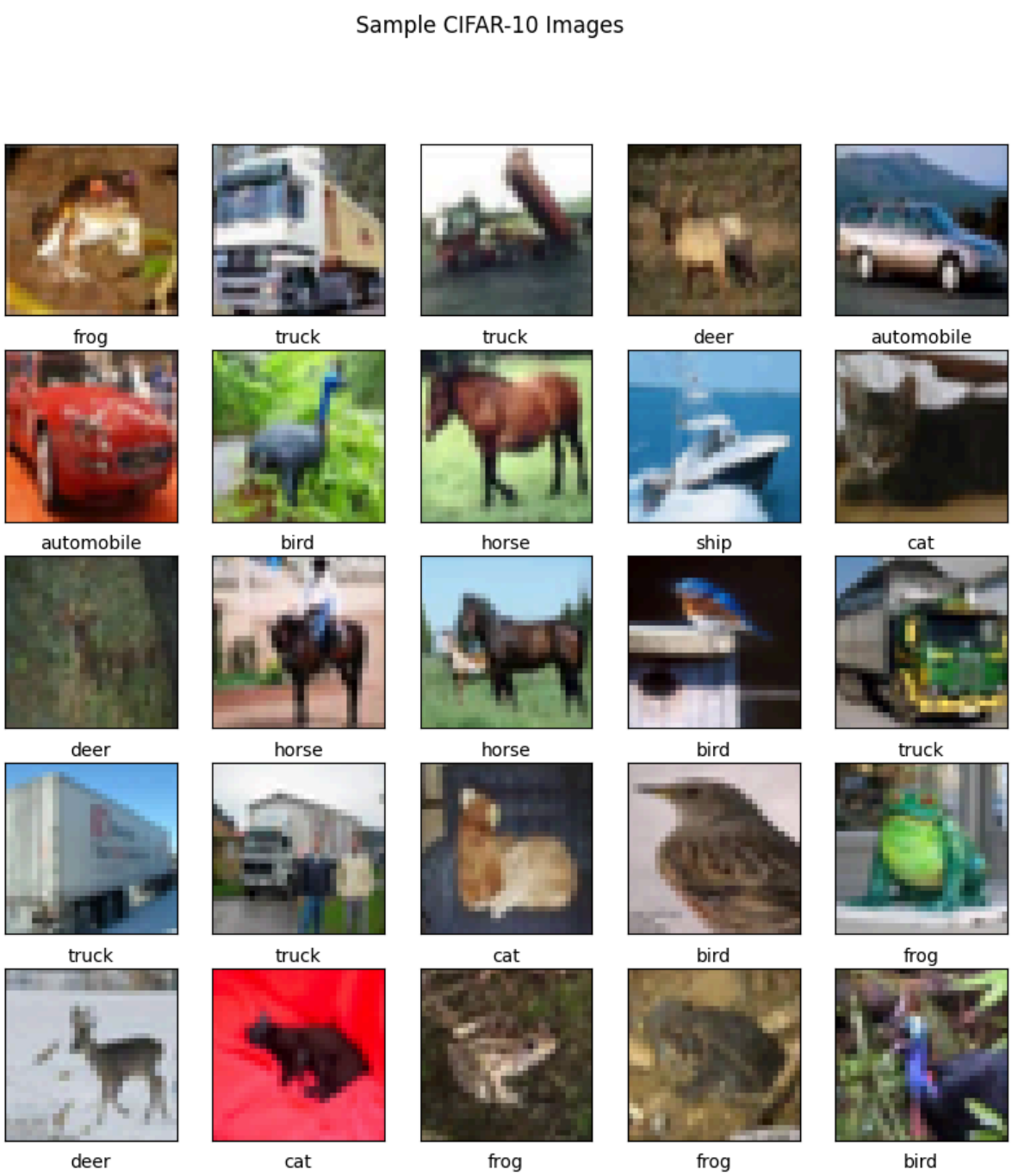
1.https://www.analyticsvidhya.com/blog/2021/01/image-classification-using-convolutional-neural-networks-a-step-by-step-guide/

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

In [30]:
```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import numpy as np

(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']
print(f"Training images shape: {train_images.shape}")
print(f"Training labels shape: {train_labels.shape}")
print(f"Test images shape: {test_images.shape}")
print(f"Test labels shape: {test_labels.shape}")
print("Dataset loaded successfully!")
```

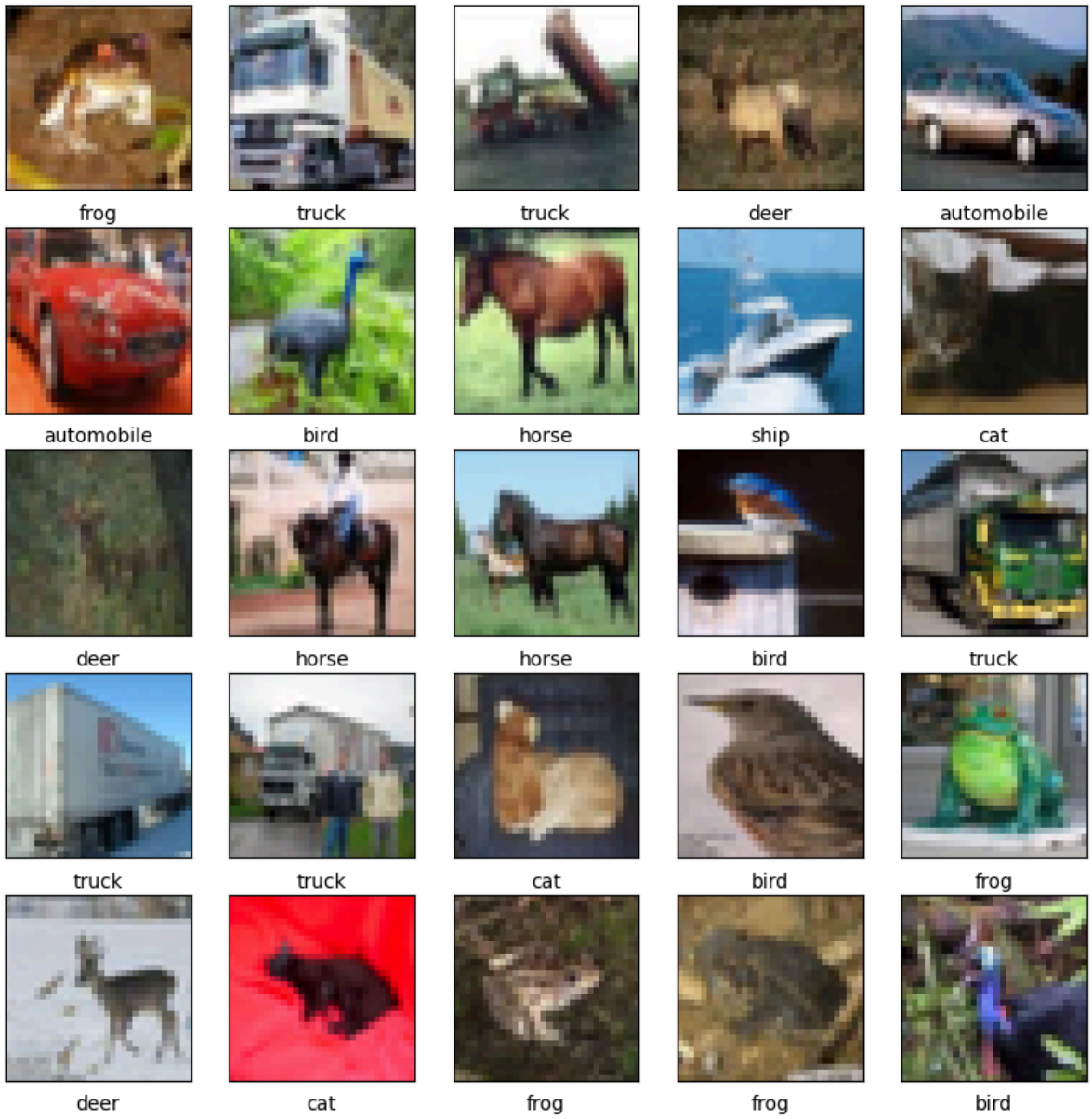
```
Training images shape: (50000, 32, 32, 3)
Training labels shape: (50000, 1)
Test images shape: (10000, 32, 32, 3)
Test labels shape: (10000, 1)
Dataset loaded successfully!
```

In [31]:
```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])
    # The labels are arrays, so we need to access the first element
    plt.xlabel(class_names[train_labels[i][0]])
plt.suptitle("Sample CIFAR-10 Images")
plt.show()
```



In [32]:
```
train_images, test_images = train_images / 255.0, test_images / 255.0
print("Pixel values normalized (divided by 255).")

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])
    # The labels are arrays, so we need to access the first element
    plt.xlabel(class_names[train_labels[i][0]])
plt.suptitle("Sample CIFAR-10 Images")
plt.show()

train_labels_one_hot = tf.keras.utils.to_categorical(train_labels, num_classes=10)
test_labels_one_hot = tf.keras.utils.to_categorical(test_labels, num_classes=10)
print("Class labels converted to one-hot encoded format.")
print(f"One-hot encoded training labels shape: {train_labels_one_hot.shape}")
```

```
Pixel values normalized (divided by 255).
```




```
Class labels converted to one-hot encoded format.
One-hot encoded training labels shape: (50000, 10)
```

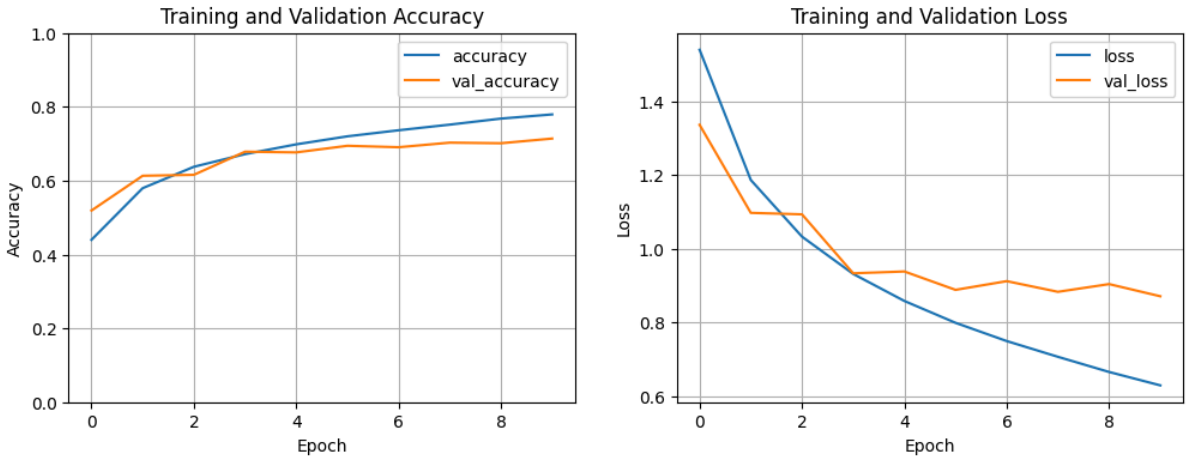
In [33]:
```python
model = models.Sequential()

# First Convolutional Block
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
# Second Convolutional Block
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
# Third Convolutional Block
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
# Flatten and Dense Layers
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
print("CNN model architecture:")
model.summary()
```

```
CNN model architecture:
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 30, 30, 32)        896

 max_pooling2d_2 (MaxPooling  (None, 15, 15, 32)       0
 2D)

 conv2d_4 (Conv2D)           (None, 13, 13, 64)        18496

 max_pooling2d_3 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 conv2d_5 (Conv2D)           (None, 4, 4, 64)          36928

 flatten_1 (Flatten)         (None, 1024)              0

 dense_2 (Dense)             (None, 64)                65600

 dense_3 (Dense)             (None, 10)                650

=================================================================
Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0
_________________________________________________________________
```

In [34]:
```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
history = model.fit(train_images, train_labels_one_hot, epochs=10,
                    validation_data=(test_images, test_labels_one_hot))
```

```
Epoch 1/10
1563/1563 [==============================] - 40s 25ms/step - loss: 1.5402 - accuracy: 0.4399
- val_loss: 1.3366 - val_accuracy: 0.5191
Epoch 2/10
1563/1563 [==============================] - 43s 28ms/step - loss: 1.1868 - accuracy: 0.5797
- val_loss: 1.0975 - val_accuracy: 0.6133
Epoch 3/10
1563/1563 [==============================] - 41s 26ms/step - loss: 1.0330 - accuracy: 0.6376
- val_loss: 1.0935 - val_accuracy: 0.6158
Epoch 4/10
1563/1563 [==============================] - 47s 30ms/step - loss: 0.9318 - accuracy: 0.6719
- val_loss: 0.9335 - val_accuracy: 0.6785
Epoch 5/10
1563/1563 [==============================] - 47s 30ms/step - loss: 0.8584 - accuracy: 0.6984
- val_loss: 0.9384 - val_accuracy: 0.6766
Epoch 6/10
1563/1563 [==============================] - 44s 28ms/step - loss: 0.7990 - accuracy: 0.7201
- val_loss: 0.8885 - val_accuracy: 0.6945
Epoch 7/10
1563/1563 [==============================] - 55s 35ms/step - loss: 0.7495 - accuracy: 0.7366
- val_loss: 0.9121 - val_accuracy: 0.6906
Epoch 8/10
1563/1563 [==============================] - 88s 56ms/step - loss: 0.7068 - accuracy: 0.7518
- val_loss: 0.8832 - val_accuracy: 0.7030
Epoch 9/10
1563/1563 [==============================] - 88s 56ms/step - loss: 0.6656 - accuracy: 0.7682
- val_loss: 0.9041 - val_accuracy: 0.7014
Epoch 10/10
1563/1563 [==============================] - 86s 55ms/step - loss: 0.6292 - accuracy: 0.7795
- val_loss: 0.8711 - val_accuracy: 0.7141
```

In [35]:
```
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend()
plt.grid(True)
plt.title('Training and Validation Accuracy')

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

test_loss, test_acc = model.evaluate(test_images, test_labels_one_hot, verbose=2)
print(f"\nTest Loss: {test_loss}")
print(f"Test Accuracy: {test_acc}")
```




```
313/313 - 5s - loss: 0.8711 - accuracy: 0.7141 - 5s/epoch - 16ms/step

Test Loss: 0.8710973858833313
Test Accuracy: 0.7141000032424927
```

In [36]:
```
sample_indices = np.random.choice(len(test_images), 10, replace=False)
sample_images = test_images[sample_indices]
sample_true_labels = test_labels[sample_indices]

predictions = model.predict(sample_images)
```

```
1/1 [==============================] - 0s 499ms/step
```

In [37]:
```
plt.figure(figsize=(15, 8))
for i in range(len(sample_images)):
    plt.subplot(2, 5, i + 1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(sample_images[i], cmap='gray')

    predicted_label_index = np.argmax(predictions[i])
    true_label_index = sample_true_labels[i][0]

    color = 'green' if predicted_label_index == true_label_index else 'red'
    plt.xlabel(f"Pred: {class_names[predicted_label_index]}\nTrue: {class_names[true_label_
```

```
plt.suptitle("Sample Predictions (Green: Correct, Red: Incorrect)", fontsize=16)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```

In [ ]:

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

In [10]:
```
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras import layers, models, regularizers
import matplotlib.pyplot as plt
import numpy as np
import os
_URL = '[https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip](https://st
path_to_zip = tf.keras.utils.get_file('cats_and_dogs.zip', origin=_URL, extract=True)

BASE_DIR = os.path.join(os.path.dirname(path_to_zip), 'cats_and_dogs_filtered')

train_dir = os.path.join(BASE_DIR, 'train')
validation_dir = os.path.join(BASE_DIR, 'validation')

print(f"Dataset extracted to: {BASE_DIR}")
print(f"Training images are in: {train_dir}")
print(f"Validation images are in: {validation_dir}")
```

```
Dataset extracted to: C:\Users\shank\.keras\datasets\cats_and_dogs_filtered
Training images are in: C:\Users\shank\.keras\datasets\cats_and_dogs_filtered\train
Validation images are in: C:\Users\shank\.keras\datasets\cats_and_dogs_filtered\validation
```

In [11]:
```
IMG_SIZE = (224, 224)
BATCH_SIZE = 32
BUFFER_SIZE = tf.data.AUTOTUNE
train_dataset = tf.keras.utils.image_dataset_from_directory(
    train_dir,
    shuffle=True,
    batch_size=BATCH_SIZE,
    image_size=IMG_SIZE
)

validation_dataset = tf.keras.utils.image_dataset_from_directory(
    validation_dir,
    shuffle=True,
    batch_size=BATCH_SIZE,
    image_size=IMG_SIZE
)

class_names = train_dataset.class_names
print(f"\nClass names: {class_names}")

plt.figure(figsize=(10, 10))
for images, labels in train_dataset.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(class_names[labels[i]])
        plt.axis("off")
plt.suptitle("Sample Images from the Dataset", fontsize=16)
plt.show()

preprocess_input = tf.keras.applications.vgg16.preprocess_input

train_dataset = train_dataset.cache().prefetch(buffer_size=BUFFER_SIZE)
validation_dataset = validation_dataset.cache().prefetch(buffer_size=BUFFER_SIZE)
```

```
Found 2000 files belonging to 2 classes.
Found 1000 files belonging to 2 classes.

Class names: ['cats', 'dogs']
```

In [12]:
```
base_model = VGG16(input_shape=IMG_SIZE + (3,),
                   include_top=False,
                   weights='imagenet')

base_model.trainable = False
global_average_layer = layers.GlobalAveragePooling2D()
prediction_layer = layers.Dense(1, activation='sigmoid')

inputs = tf.keras.Input(shape=(224, 224, 3))
x = preprocess_input(inputs)
x = base_model(x, training=False)
x = global_average_layer(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)

print("VGG16-based Transfer Learning Model Summary:")
model.summary()

#
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16
_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 43s 1us/step
VGG16-based Transfer Learning Model Summary:
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]     0

 tf.__operators__.getitem (S  (None, 224, 224, 3)      0
 licingOpLambda)

 tf.nn.bias_add (TFOpLambda)  (None, 224, 224, 3)      0

 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 global_average_pooling2d (G  (None, 512)              0
 lobalAveragePooling2D)

 dense (Dense)               (None, 1)                 513

=================================================================
Total params: 14,715,201
Trainable params: 513
Non-trainable params: 14,714,688
_________________________________________________________________
```

In [13]:

```python
base_learning_rate = 0.0001

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=base_learning_rate),
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=['accuracy'])

initial_epochs = 10
print(f"\nStarting model training for {initial_epochs} epochs...")
history = model.fit(
    train_dataset,
    epochs=initial_epochs,
    validation_data=validation_dataset
)
print("Model training complete!")
```

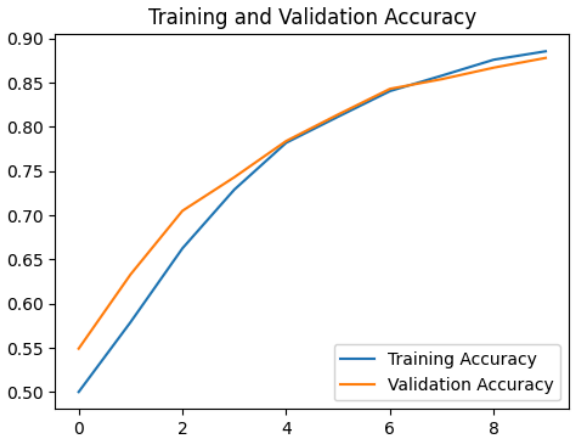
```
Starting model training for 10 epochs...
Epoch 1/10
63/63 [==============================] - 563s 9s/step - loss: 2.5582 - accuracy: 0.5000 - val_
loss: 2.1008 - val_accuracy: 0.5490
Epoch 2/10
63/63 [==============================] - 438s 7s/step - loss: 1.7738 - accuracy: 0.5790 - val_
loss: 1.5291 - val_accuracy: 0.6330
Epoch 3/10
63/63 [==============================] - 396s 6s/step - loss: 1.3127 - accuracy: 0.6625 - val_
loss: 1.1527 - val_accuracy: 0.7050
Epoch 4/10
63/63 [==============================] - 402s 6s/step - loss: 1.0072 - accuracy: 0.7290 - val_
loss: 0.8974 - val_accuracy: 0.7430
Epoch 5/10
63/63 [==============================] - 374s 6s/step - loss: 0.8007 - accuracy: 0.7820 - val_
loss: 0.7199 - val_accuracy: 0.7840
Epoch 6/10
63/63 [==============================] - 366s 6s/step - loss: 0.6575 - accuracy: 0.8115 - val_
loss: 0.5939 - val_accuracy: 0.8140
Epoch 7/10
63/63 [==============================] - 393s 6s/step - loss: 0.5555 - accuracy: 0.8405 - val_
loss: 0.5022 - val_accuracy: 0.8430
Epoch 8/10
63/63 [==============================] - 371s 6s/step - loss: 0.4808 - accuracy: 0.8580 - val_
loss: 0.4327 - val_accuracy: 0.8540
Epoch 9/10
63/63 [==============================] - 355s 6s/step - loss: 0.4236 - accuracy: 0.8760 - val_
loss: 0.3784 - val_accuracy: 0.8670
Epoch 10/10
63/63 [==============================] - 351s 6s/step - loss: 0.3782 - accuracy: 0.8855 - val_
loss: 0.3351 - val_accuracy: 0.8780
Model training complete!
```

In [15]:

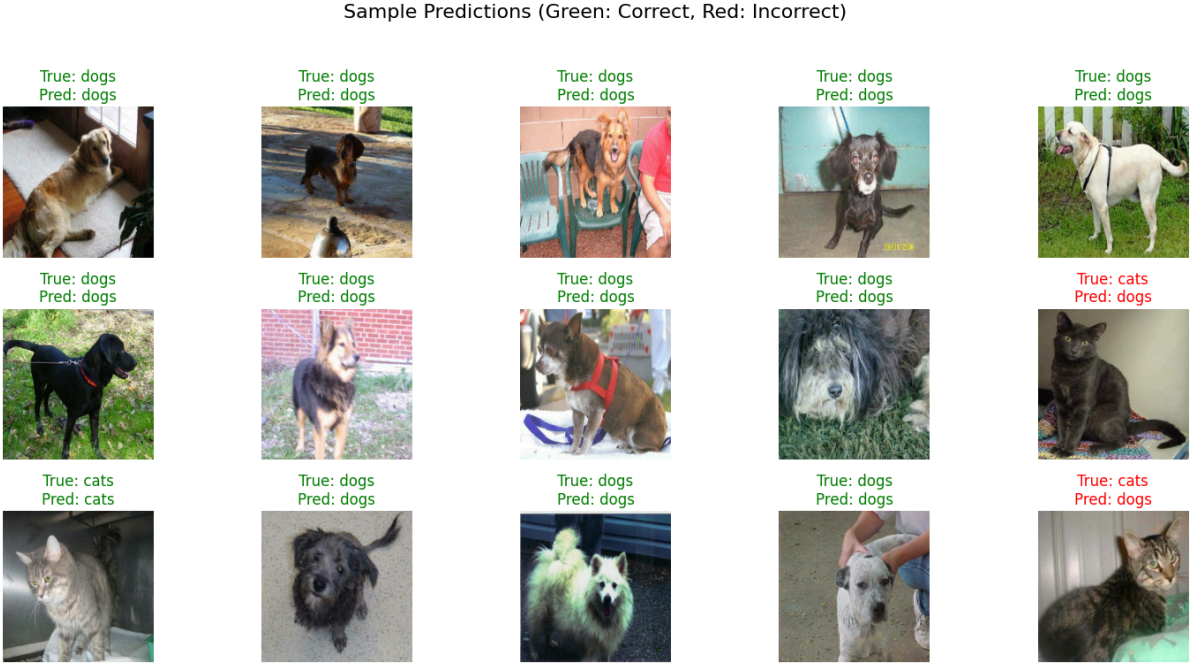
```python
loss, accuracy = model.evaluate(validation_dataset)
print(f"\nModel Test Accuracy: {accuracy*100:.2f}%")
print(f"Model Test Loss: {loss:.4f}")

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss_history = history.history['loss']
val_loss_history = history.history['val_loss']
epochs_range = range(initial_epochs)

plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss_history, label='Training Loss')
plt.plot(epochs_range, val_loss_history, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()

test_images, test_labels = next(iter(validation_dataset))

predictions = model.predict(test_images)

predicted_labels = (predictions > 0.5).astype("int32")

true_labels_names = [class_names[i] for i in test_labels.numpy()]
predicted_labels_names = [class_names[i] for i in predicted_labels.flatten()]

plt.figure(figsize=(15, 8))
num_images_to_show = min(len(test_images), 15)
```

```
for i in range(num_images_to_show):
    plt.subplot(3, 5, i + 1)
    plt.imshow(test_images[i].numpy().astype("uint8"))
    plt.title(f"True: {true_labels_names[i]}\nPred: {predicted_labels_names[i]}",
              color='green' if true_labels_names[i] == predicted_labels_names[i] else 'red')
    plt.axis("off")
plt.suptitle("Sample Predictions (Green: Correct, Red: Incorrect)", fontsize=16)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```

```
32/32 [==============================] - 109s 3s/step - loss: 0.3351 - accuracy: 0.8780

Model Test Accuracy: 87.80%
Model Test Loss: 0.3351
```




```
1/1 [==============================] - 4s 4s/step
```



In [ ]:

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:

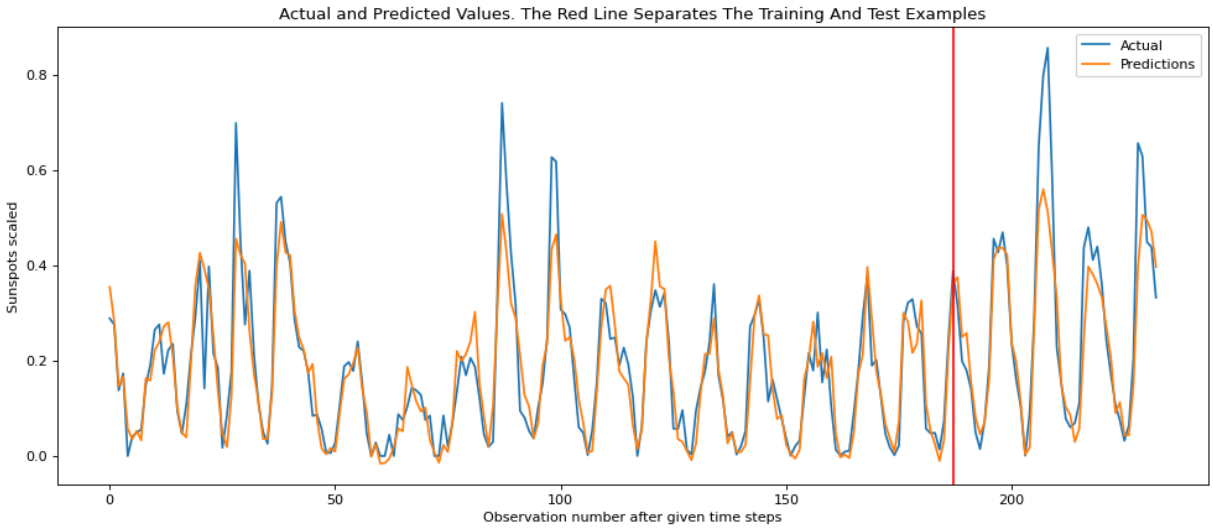
1. https://machinelearningmastery.com/understanding-simple-recurrent-neural-networks-in-keras/
2. https://victorzhou.com/blog/keras-rnn-tutorial/

In [1]:
```
from pandas import read_csv
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, SimpleRNN
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
import math
import matplotlib.pyplot as plt
```

In [4]:
```
# Parameter split_percent defines the ratio of training examples
def get_train_test(url, split_percent=0.8):
    df = read_csv(url, usecols=[1], engine='python')
    data = np.array(df.values.astype('float32'))
    scaler = MinMaxScaler(feature_range=(0, 1))
    data = scaler.fit_transform(data).flatten()
    n = len(data)
    # Point for splitting data into train and test
    split = int(n*split_percent)
    train_data = data[range(split)]
    test_data = data[split:]
    return train_data, test_data, data

# Prepare the input X and target Y
def get_XY(dat, time_steps):
    Y_ind = np.arange(time_steps, len(dat), time_steps)
    Y = dat[Y_ind]
    rows_x = len(Y)
    X = dat[range(time_steps*rows_x)]
    X = np.reshape(X, (rows_x, time_steps, 1))
    return X, Y

def create_RNN(hidden_units, dense_units, input_shape, activation):
    model = Sequential()
    model.add(SimpleRNN(hidden_units, input_shape=input_shape, activation=activation[0]))
    model.add(Dense(units=dense_units, activation=activation[1]))
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model

def print_error(trainY, testY, train_predict, test_predict):
    # Error of predictions
    train_rmse = math.sqrt(mean_squared_error(trainY, train_predict))
    test_rmse = math.sqrt(mean_squared_error(testY, test_predict))
    # Print RMSE
    print('Train RMSE: %.3f RMSE' % (train_rmse))
    print('Test RMSE: %.3f RMSE' % (test_rmse))

# Plot the result
def plot_result(trainY, testY, train_predict, test_predict):
    actual = np.append(trainY, testY)
    predictions = np.append(train_predict, test_predict)
    rows = len(actual)
    plt.figure(figsize=(15, 6), dpi=80)
    plt.plot(range(rows), actual)
    plt.plot(range(rows), predictions)
    plt.axvline(x=len(trainY), color='r')
    plt.legend(['Actual', 'Predictions'])
    plt.xlabel('Observation number after given time steps')
    plt.ylabel('Sunspots scaled')
    plt.title('Actual and Predicted Values. The Red Line Separates The Training And Test Examples')

sunspots_url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/monthly-sunspots.csv'
time_steps = 12
train_data, test_data, data = get_train_test(sunspots_url)
trainX, trainY = get_XY(train_data, time_steps)
testX, testY = get_XY(test_data, time_steps)

# Create model and train
model = create_RNN(hidden_units=3, dense_units=1, input_shape=(time_steps,1),
                   activation=['tanh', 'tanh'])
model.fit(trainX, trainY, epochs=20, batch_size=1, verbose=2)

# make predictions
```

```
train_predict = model.predict(trainX)
test_predict = model.predict(testX)

# Print error
print_error(trainY, testY, train_predict, test_predict)

#Plot result
plot_result(trainY, testY, train_predict, test_predict)
```

```
Epoch 1/20
187/187 - 1s - loss: 0.0152 - 1s/epoch - 6ms/step
Epoch 2/20
187/187 - 0s - loss: 0.0113 - 238ms/epoch - 1ms/step
Epoch 3/20
187/187 - 0s - loss: 0.0091 - 245ms/epoch - 1ms/step
Epoch 4/20
187/187 - 0s - loss: 0.0078 - 322ms/epoch - 2ms/step
Epoch 5/20
187/187 - 0s - loss: 0.0068 - 249ms/epoch - 1ms/step
Epoch 6/20
187/187 - 0s - loss: 0.0062 - 234ms/epoch - 1ms/step
Epoch 7/20
187/187 - 0s - loss: 0.0057 - 238ms/epoch - 1ms/step
Epoch 8/20
187/187 - 0s - loss: 0.0053 - 247ms/epoch - 1ms/step
Epoch 9/20
187/187 - 0s - loss: 0.0050 - 256ms/epoch - 1ms/step
Epoch 10/20
187/187 - 0s - loss: 0.0048 - 239ms/epoch - 1ms/step
Epoch 11/20
187/187 - 0s - loss: 0.0046 - 261ms/epoch - 1ms/step
Epoch 12/20
187/187 - 0s - loss: 0.0045 - 249ms/epoch - 1ms/step
Epoch 13/20
187/187 - 0s - loss: 0.0043 - 248ms/epoch - 1ms/step
Epoch 14/20
187/187 - 0s - loss: 0.0042 - 238ms/epoch - 1ms/step
Epoch 15/20
187/187 - 0s - loss: 0.0041 - 258ms/epoch - 1ms/step
Epoch 16/20
187/187 - 0s - loss: 0.0040 - 267ms/epoch - 1ms/step
Epoch 17/20
187/187 - 0s - loss: 0.0040 - 239ms/epoch - 1ms/step
Epoch 18/20
187/187 - 0s - loss: 0.0038 - 246ms/epoch - 1ms/step
Epoch 19/20
187/187 - 0s - loss: 0.0039 - 241ms/epoch - 1ms/step
Epoch 20/20
187/187 - 0s - loss: 0.0038 - 242ms/epoch - 1ms/step
6/6 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 3ms/step
Train RMSE: 0.060 RMSE
Test RMSE: 0.094 RMSE
```



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

In [ ]:
```
import re
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
```

In [ ]:
```
print("TensorFlow:", tf.__version__)
```

```
TensorFlow: 2.11.0
```

In [ ]:
```
vocab_size = 10000
maxlen = 200  # sequence length (pad/cut reviews to this length)
```

In [ ]:
```
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=vocab_size)
```

In [ ]:
```
x_train = pad_sequences(x_train, maxlen=maxlen)
x_test  = pad_sequences(x_test,  maxlen=maxlen)
```

In [ ]:
```
print("Train shape:", x_train.shape, "Test shape:", x_test.shape)
```

```
Train shape: (25000, 200) Test shape: (25000, 200)
```

In [ ]:
```
model = keras.Sequential([
    layers.Embedding(input_dim=vocab_size, output_dim=64, input_length=maxl
    layers.SimpleRNN(64, activation="tanh"),
    layers.Dense(1, activation="sigmoid")
])
```

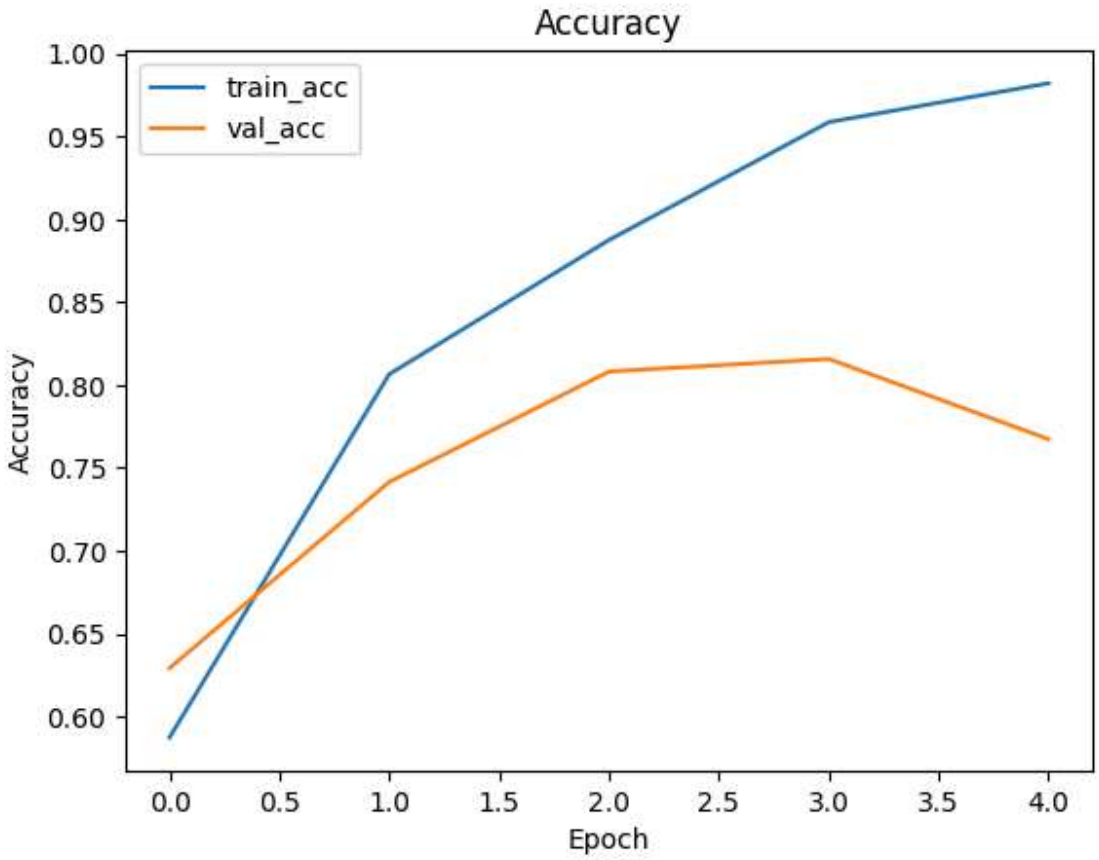
In [ ]:
```
model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accur
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 200, 64)           640000

 simple_rnn (SimpleRNN)      (None, 64)                8256

 dense (Dense)               (None, 1)                 65

=================================================================
Total params: 648,321
Trainable params: 648,321
Non-trainable params: 0
```

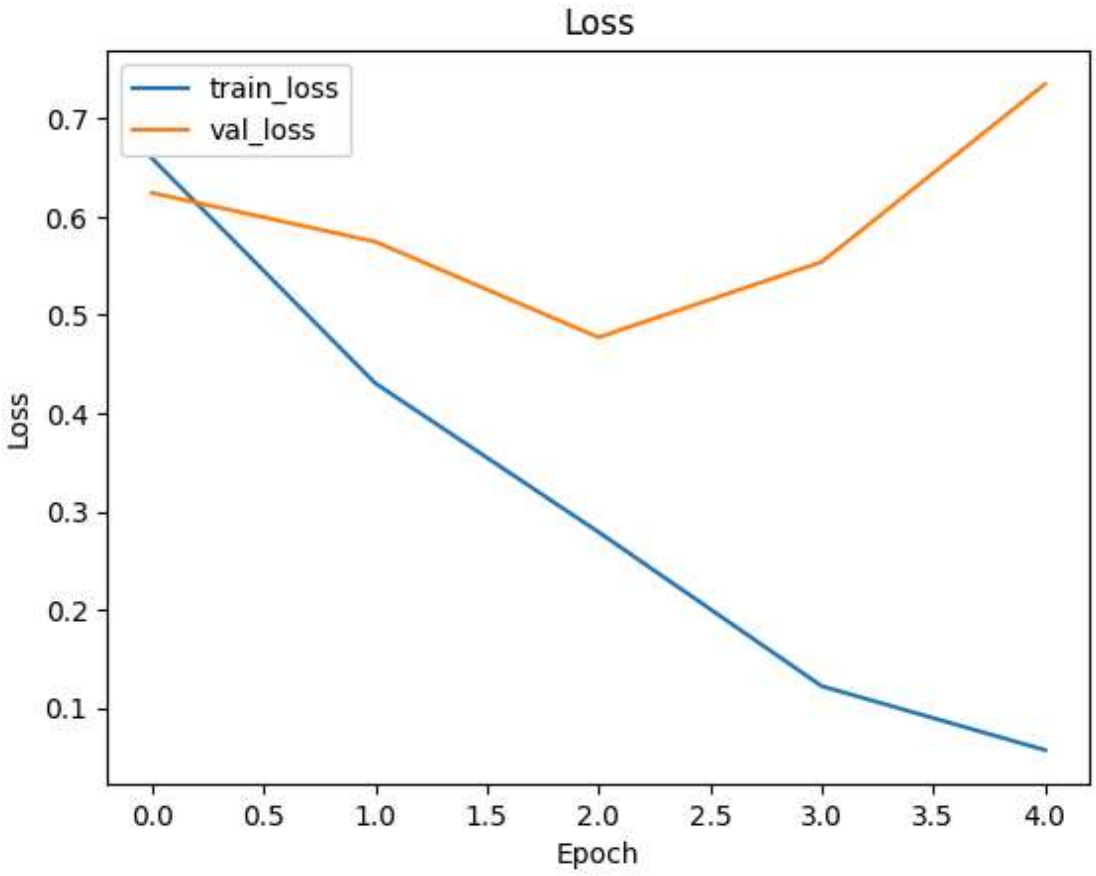
In [ ]:
```
history = model.fit(
    x_train, y_train,
    epochs=5,
    batch_size=64,
    validation_split=0.2,
    verbose=1
)
```

```
Epoch 1/5
313/313 [==============================] - 33s 97ms/step - loss: 0.6595 - acc
uracy: 0.5874 - val_loss: 0.6243 - val_accuracy: 0.6292
Epoch 2/5
313/313 [==============================] - 30s 95ms/step - loss: 0.4306 - acc
uracy: 0.8066 - val_loss: 0.5746 - val_accuracy: 0.7416
Epoch 3/5
313/313 [==============================] - 30s 95ms/step - loss: 0.2791 - acc
uracy: 0.8876 - val_loss: 0.4773 - val_accuracy: 0.8082
Epoch 4/5
313/313 [==============================] - 31s 99ms/step - loss: 0.1222 - acc
uracy: 0.9587 - val_loss: 0.5540 - val_accuracy: 0.8158
Epoch 5/5
313/313 [==============================] - 30s 97ms/step - loss: 0.0571 - acc
uracy: 0.9822 - val_loss: 0.7352 - val_accuracy: 0.7674
```

In [ ]:
```
plt.figure()
plt.plot(history.history["accuracy"], label="train_acc")
plt.plot(history.history["val_accuracy"], label="val_acc")
plt.title("Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



In [ ]:
```
plt.figure()
plt.plot(history.history["loss"], label="train_loss")
plt.plot(history.history["val_loss"], label="val_loss")
plt.title("Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



In [ ]:
```
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)
print(f"\nTest Accuracy: {test_acc:.4f}")
print(f"Test Loss    : {test_loss:.4f}")
```

```
Test Accuracy: 0.7641
Test Loss    : 0.7513
```

In [ ]:
```
y_prob = model.predict(x_test, verbose=0).reshape(-1)
y_pred = (y_prob >= 0.5).astype(int)
```

Negative (0)	0.76	0.78	0.77	12500
Positive (1)	0.77	0.75	0.76	12500
accuracy			0.76	25000
macro avg	0.76	0.76	0.76	25000
weighted avg	0.76	0.76	0.76	25000

In [ ]:

```
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)
```

```
Confusion Matrix:
 [[9691 2809]
 [3089 9411]]
```

In [ ]:

```
word_index = imdb.get_word_index()

# Keras IMDB uses reserved indices:
# 0: padding, 1: start_of_sequence, 2: unknown, 3: unused
INDEX_FROM = 3
```

In [ ]:

```
def encode_review(raw_text, maxlen=maxlen):
    # Basic clean: lowercase & keep words
    text = raw_text.lower()
    # Split on non-letters to get tokens
    tokens = re.findall(r"[a-z']+", text)

    # Map tokens to indices (with +INDEX_FROM); unknown->2
    seq = []
    for w in tokens:
        idx = word_index.get(w, None)
        if idx is not None and (idx + INDEX_FROM) < vocab_size:
            seq.append(idx + INDEX_FROM)
        else:
            seq.append(2)  # unknown token

    # Optionally prepend start-of-sequence token 1
    seq = [1] + seq
    # Pad to required length
    return pad_sequences([seq], maxlen=maxlen)[0]

def predict_sentiment(raw_text):
    seq = encode_review(raw_text)
    prob = float(model.predict(np.array([seq]), verbose=0)[0][0])
    label = 1 if prob >= 0.5 else 0
    return label, prob
```

In [ ]:

```
samples = [
    "The movie was amazing and I loved it!",
    "This was a terrible film. I hated every minute.",
    "Not bad, but could have been better.",
]

for s in samples:
    label, prob = predict_sentiment(s)
    print(f"\nText: {s}\nPredicted: {'Positive (1)' if label==1 else 'Negat

# ---- Optional: Inspect a few test predictions vs actual ----
print("\nSample Predictions vs Actual (first 10):")
sample_probs = model.predict(x_test[:10], verbose=0).reshape(-1)
sample_pred = (sample_probs >= 0.5).astype(int)
print("Pred :", sample_pred.tolist())
print("Actual:", y_test[:10].tolist())
```

```
Text: The movie was amazing and I loved it!
Predicted: Positive (1) | Probability=1.000
```

```
Text: This was a terrible film. I hated every minute.
Predicted: Positive (1) | Probability=0.686

Text: Not bad, but could have been better.
Predicted: Negative (0) | Probability=0.002

Sample Predictions vs Actual (first 10):
Pred : [0, 1, 0, 1, 1, 0, 0, 0, 1, 0]
Actual: [0, 1, 1, 0, 1, 1, 1, 0, 0, 1]
```

In [ ]:

```
predict_sentiment("i hate the movie")
```

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

In [1]:
```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt
```

In [2]:
```
(x_train, _), (x_test, _) = keras.datasets.mnist.load_data()

print(f"Original x_train shape: {x_train.shape}")
print(f"Original x_test shape: {x_test.shape}")
```

```
Original x_train shape: (60000, 28, 28)
Original x_test shape: (10000, 28, 28)
```

In [3]:
```
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))

print(f"Processed x_train shape: {x_train.shape}")
print(f"Processed x_test shape: {x_test.shape}")

input_dim = x_train.shape[1] # 784

encoding_dim = 32
```

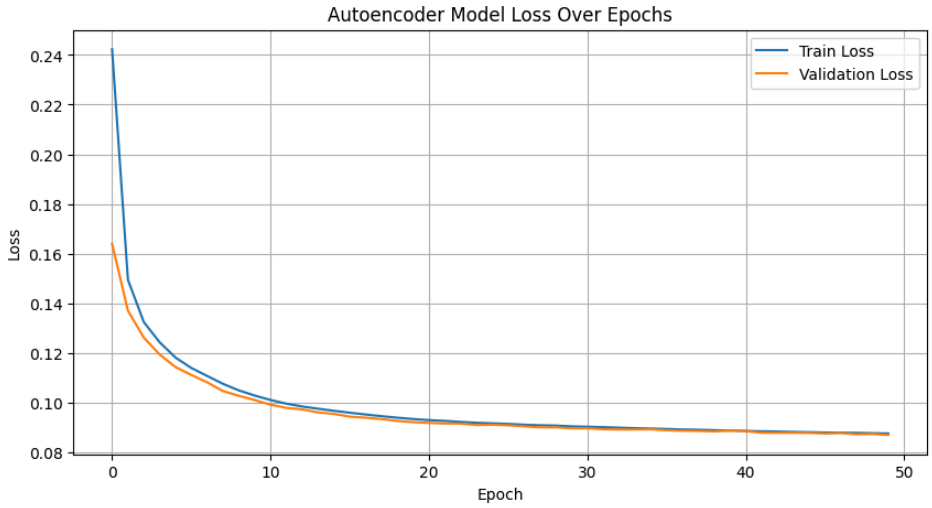
```
Processed x_train shape: (60000, 784)
Processed x_test shape: (10000, 784)
```

In [4]:
```
input_img = keras.Input(shape=(input_dim,))
encoded = layers.Dense(128, activation='relu')(input_img)
encoded = layers.Dense(64, activation='relu')(encoded)
encoded = layers.Dense(encoding_dim, activation='relu')(encoded) # Bottleneck layer

decoded = layers.Dense(64, activation='relu')(encoded)
decoded = layers.Dense(128, activation='relu')(decoded)
decoded = layers.Dense(input_dim, activation='sigmoid')(decoded)

autoencoder = keras.Model(input_img, decoded)

encoder = keras.Model(input_img, encoded)

autoencoder.summary()
encoder.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 784)]             0

 dense (Dense)               (None, 128)               100480

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 32)                2080

 dense_3 (Dense)             (None, 64)                2112

 dense_4 (Dense)             (None, 128)               8320

 dense_5 (Dense)             (None, 784)               101136

=================================================================
Total params: 222,384
Trainable params: 222,384
Non-trainable params: 0
_________________________________________________________________
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 784)]             0

 dense (Dense)               (None, 128)               100480

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 32)                2080

=================================================================
Total params: 110,816
Trainable params: 110,816
Non-trainable params: 0
_________________________________________________________________
```

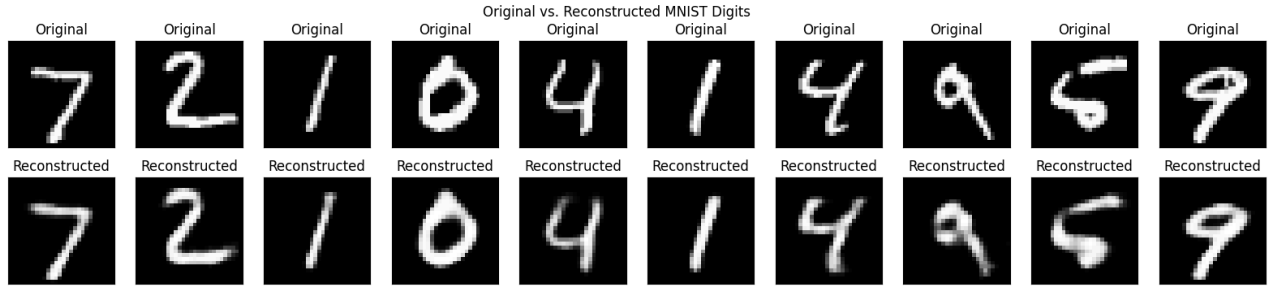
In [5]:
```
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
history = autoencoder.fit(x_train, x_train,
                          epochs=50,
                          batch_size=256,
                          shuffle=True,
                          validation_data=(x_test, x_test),
                          verbose=1)

plt.figure(figsize=(10, 5))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Autoencoder Model Loss Over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()
```

```
Epoch 1/50
235/235 [==============================] - 3s 7ms/step - loss: 0.2423 - val_loss: 0.1639
Epoch 2/50
235/235 [==============================] - 1s 6ms/step - loss: 0.1494 - val_loss: 0.1370
Epoch 3/50
235/235 [==============================] - 1s 6ms/step - loss: 0.1324 - val_loss: 0.1261
Epoch 4/50
235/235 [==============================] - 2s 6ms/step - loss: 0.1242 - val_loss: 0.1193
Epoch 5/50
235/235 [==============================] - 2s 7ms/step - loss: 0.1181 - val_loss: 0.1143
Epoch 6/50
235/235 [==============================] - 1s 6ms/step - loss: 0.1139 - val_loss: 0.1111
Epoch 7/50
235/235 [==============================] - 2s 6ms/step - loss: 0.1107 - val_loss: 0.1081
Epoch 8/50
235/235 [==============================] - 2s 6ms/step - loss: 0.1075 - val_loss: 0.1046
Epoch 9/50
235/235 [==============================] - 2s 6ms/step - loss: 0.1048 - val_loss: 0.1027
Epoch 10/50
235/235 [==============================] - 1s 6ms/step - loss: 0.1028 - val_loss: 0.1009
Epoch 11/50
235/235 [==============================] - 1s 6ms/step - loss: 0.1010 - val_loss: 0.0991
Epoch 12/50
235/235 [==============================] - 2s 7ms/step - loss: 0.0995 - val_loss: 0.0978
Epoch 13/50
235/235 [==============================] - 2s 6ms/step - loss: 0.0984 - val_loss: 0.0971
Epoch 14/50
235/235 [==============================] - 2s 7ms/step - loss: 0.0974 - val_loss: 0.0960
Epoch 15/50
235/235 [==============================] - 2s 6ms/step - loss: 0.0966 - val_loss: 0.0953
Epoch 16/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0958 - val_loss: 0.0943
Epoch 17/50
235/235 [==============================] - 2s 7ms/step - loss: 0.0951 - val_loss: 0.0938
Epoch 18/50
235/235 [==============================] - 2s 6ms/step - loss: 0.0944 - val_loss: 0.0933
Epoch 19/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0938 - val_loss: 0.0925
Epoch 20/50
235/235 [==============================] - 2s 7ms/step - loss: 0.0933 - val_loss: 0.0920
Epoch 21/50
235/235 [==============================] - 2s 6ms/step - loss: 0.0929 - val_loss: 0.0917
Epoch 22/50
235/235 [==============================] - 2s 7ms/step - loss: 0.0925 - val_loss: 0.0915
Epoch 23/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0921 - val_loss: 0.0915
Epoch 24/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0918 - val_loss: 0.0909
Epoch 25/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0916 - val_loss: 0.0910
Epoch 26/50
235/235 [==============================] - 2s 7ms/step - loss: 0.0913 - val_loss: 0.0908
Epoch 27/50
235/235 [==============================] - 2s 6ms/step - loss: 0.0910 - val_loss: 0.0904
Epoch 28/50
235/235 [==============================] - 2s 7ms/step - loss: 0.0908 - val_loss: 0.0899
Epoch 29/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0907 - val_loss: 0.0900
Epoch 30/50
235/235 [==============================] - 2s 6ms/step - loss: 0.0903 - val_loss: 0.0895
Epoch 31/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0902 - val_loss: 0.0896
Epoch 32/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0900 - val_loss: 0.0892
Epoch 33/50
235/235 [==============================] - 2s 6ms/step - loss: 0.0898 - val_loss: 0.0891
Epoch 34/50
235/235 [==============================] - 2s 6ms/step - loss: 0.0896 - val_loss: 0.0891
Epoch 35/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0895 - val_loss: 0.0892
Epoch 36/50
235/235 [==============================] - 2s 7ms/step - loss: 0.0893 - val_loss: 0.0888
Epoch 37/50
235/235 [==============================] - 2s 7ms/step - loss: 0.0891 - val_loss: 0.0886
Epoch 38/50
235/235 [==============================] - 2s 7ms/step - loss: 0.0890 - val_loss: 0.0885
Epoch 39/50
235/235 [==============================] - 2s 6ms/step - loss: 0.0888 - val_loss: 0.0883
Epoch 40/50
235/235 [==============================] - 2s 7ms/step - loss: 0.0887 - val_loss: 0.0886
Epoch 41/50
235/235 [==============================] - 1s 6ms/step - loss: 0.0885 - val_loss: 0.0884
Epoch 42/50
235/235 [==============================] - 2s 6ms/step - loss: 0.0884 - val_loss: 0.0878
Epoch 43/50
235/235 [==============================] - 2s 7ms/step - loss: 0.0883 - val_loss: 0.0877
Epoch 44/50
235/235 [==============================] - 2s 7ms/step - loss: 0.0881 - val_loss: 0.0877
Epoch 45/50
235/235 [==============================] - 2s 7ms/step - loss: 0.0880 - val_loss: 0.0877
Epoch 46/50
235/235 [==============================] - 2s 7ms/step - loss: 0.0879 - val_loss: 0.0874
Epoch 47/50
235/235 [==============================] - 2s 7ms/step - loss: 0.0878 - val_loss: 0.0877
Epoch 48/50
235/235 [==============================] - 2s 7ms/step - loss: 0.0877 - val_loss: 0.0872
Epoch 49/50
235/235 [==============================] - 2s 7ms/step - loss: 0.0876 - val_loss: 0.0873
Epoch 50/50
235/235 [==============================] - 2s 7ms/step - loss: 0.0875 - val_loss: 0.0869
```



In [7]:
```
encoded_imgs = encoder.predict(x_test)
decoded_imgs = autoencoder.predict(x_test)

n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
    ax.set_title("Original")

    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
    ax.set_title("Reconstructed")
plt.suptitle('Original vs. Reconstructed MNIST Digits')
plt.show()
```

```
313/313 [==============================] - 0s 761us/step
313/313 [==============================] - 0s 1ms/step
```



In [8]:
```
test_loss = autoencoder.evaluate(x_test, x_test, verbose=0)
print(f"Final Test Reconstruction Loss (Binary Cross-entropy): {test_loss:.4f}")

mse = np.mean(np.power(x_test - decoded_imgs, 2))
print(f"Mean Squared Error (MSE) on Test Set: {mse:.4f}")
```

```
Final Test Reconstruction Loss (Binary Cross-entropy): 0.0869
Mean Squared Error (MSE) on Test Set: 0.0084
```

In [ ]:

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

In [2]:
```
import struct
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import LeakyReLU
from tensorflow.keras.layers import ZeroPadding2D
from tensorflow.keras.layers import UpSampling2D
from tensorflow.keras.layers import add, concatenate
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.image import load_img
import matplotlib.pyplot as plt
from matplotlib import pyplot
from matplotlib.patches import Rectangle
from numpy import expand_dims
```

In [4]:
```
def _conv_block(inp, convs, skip=True):
    x = inp
    count = 0
    for conv in convs:
        if count == (len(convs) - 2) and skip:
            skip_connection = x
        count += 1
        if conv['stride'] > 1: x = ZeroPadding2D(((1, 0), (1, 0)))(x)  # peculiar padding as darknet prefer left and top
        x = Conv2D(conv['filter'],
                   conv['kernel'],
                   strides=conv['stride'],
                   padding='valid' if conv['stride'] > 1 else 'same',  # peculiar padding as darknet prefer left and top
                   name='conv_' + str(conv['layer_idx']),
                   use_bias=False if conv['bnorm'] else True)(x)
        if conv['bnorm']: x = BatchNormalization(epsilon=0.001, name='bnorm_' + str(conv['layer_idx']))(x)
        if conv['leaky']: x = LeakyReLU(alpha=0.1, name='leaky_' + str(conv['layer_idx']))(x)
    return add([skip_connection, x]) if skip else x


def make_yolov3_model():
    input_image = Input(shape=(None, None, 3))
    # Layer  0 => 4
    x = _conv_block(input_image,
                    [{'filter': 32, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 0},
                     {'filter': 64, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 1},
                     {'filter': 32, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 2},
                     {'filter': 64, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 3}])
    # Layer  5 => 8
    x = _conv_block(x, [{'filter': 128, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 5},
                        {'filter': 64, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 6},
                        {'filter': 128, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 7}])
    # Layer  9 => 11
    x = _conv_block(x, [{'filter': 64, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 9},
                        {'filter': 128, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 10}])
    # Layer 12 => 15
    x = _conv_block(x, [{'filter': 256, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 12},
                        {'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 13},
                        {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 14}])
    # Layer 16 => 36
    for i in range(7):
        x = _conv_block(x, [
            {'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 16 + i * 3},
            {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 17 + i * 3}])
    skip_36 = x
    # Layer 37 => 40
    x = _conv_block(x, [{'filter': 512, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 37},
                        {'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 38},
                        {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 39}])
    # Layer 41 => 61
    for i in range(7):
        x = _conv_block(x, [
            {'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 41 + i * 3},
            {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 42 + i * 3}])
    skip_61 = x
    # Layer 62 => 65
    x = _conv_block(x, [{'filter': 1024, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 62},
                        {'filter': 512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 63},
                        {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 64}])
    # Layer 66 => 74
    for i in range(3):
        x = _conv_block(x, [
            {'filter': 512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 66 + i * 3},
            {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 67 + i * 3}])
    # Layer 75 => 79
    x = _conv_block(x, [{'filter': 512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 75},
                        {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 76},
                        {'filter': 512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 77},
                        {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 78},
                        {'filter': 512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 79}],
                    skip=False)
    # Layer 80 => 82
    yolo_82 = _conv_block(x, [{'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 80},
                              {'filter': 255, 'kernel': 1, 'stride': 1, 'bnorm': False, 'leaky': False,
                               'layer_idx': 81}], skip=False)
    # Layer 83 => 86
    x = _conv_block(x, [{'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 84}],
                    skip=False)
    x = UpSampling2D(2)(x)
    x = concatenate([x, skip_61])
    # Layer 87 => 91
```

```
    x = _conv_block(x, [{'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 87},
                        {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 88},
                        {'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 89},
                        {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 90},
                        {'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 91}],
                    skip=False)
    # Layer 92 => 94
    yolo_94 = _conv_block(x, [{'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 92},
                              {'filter': 255, 'kernel': 1, 'stride': 1, 'bnorm': False, 'leaky': False,
                               'layer_idx': 93}], skip=False)
    # Layer 95 => 98
    x = _conv_block(x, [{'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 96}],
                    skip=False)
    x = UpSampling2D(2)(x)
    x = concatenate([x, skip_36])
    # Layer 99 => 106
    yolo_106 = _conv_block(x, [{'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 99},
                               {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True,
                                'layer_idx': 100},
                               {'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
                                'layer_idx': 101},
                               {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True,
                                'layer_idx': 102},
                               {'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
                                'layer_idx': 103},
                               {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True,
                                'layer_idx': 104},
                               {'filter': 255, 'kernel': 1, 'stride': 1, 'bnorm': False, 'leaky': False,
                                'layer_idx': 105}], skip=False)
    model = Model(input_image, [yolo_82, yolo_94, yolo_106])
    return model


class WeightReader:
    def __init__(self, weight_file):
        with open(weight_file, 'rb') as w_f:
            major, = struct.unpack('i', w_f.read(4))
            minor, = struct.unpack('i', w_f.read(4))
            revision, = struct.unpack('i', w_f.read(4))
            if (major * 10 + minor) >= 2 and major < 1000 and minor < 1000:
                w_f.read(8)
            else:
                w_f.read(4)
            transpose = (major > 1000) or (minor > 1000)
            binary = w_f.read()
        self.offset = 0
        self.all_weights = np.frombuffer(binary, dtype='float32')

    def read_bytes(self, size):
        self.offset = self.offset + size
        return self.all_weights[self.offset - size:self.offset]

    def load_weights(self, model):
        for i in range(106):
            try:
                conv_layer = model.get_layer('conv_' + str(i))
                print("loading weights of convolution #" + str(i))
                if i not in [81, 93, 105]:
                    norm_layer = model.get_layer('bnorm_' + str(i))
                    size = np.prod(norm_layer.get_weights()[0].shape)
                    beta = self.read_bytes(size)  # bias
                    gamma = self.read_bytes(size)  # scale
                    mean = self.read_bytes(size)  # mean
                    var = self.read_bytes(size)  # variance
                    weights = norm_layer.set_weights([gamma, beta, mean, var])
                if len(conv_layer.get_weights()) > 1:
                    bias = self.read_bytes(np.prod(conv_layer.get_weights()[1].shape))
                    kernel = self.read_bytes(np.prod(conv_layer.get_weights()[0].shape))
                    kernel = kernel.reshape(list(reversed(conv_layer.get_weights()[0].shape)))
                    kernel = kernel.transpose([2, 3, 1, 0])
                    conv_layer.set_weights([kernel, bias])
                else:
                    kernel = self.read_bytes(np.prod(conv_layer.get_weights()[0].shape))
                    kernel = kernel.reshape(list(reversed(conv_layer.get_weights()[0].shape)))
                    kernel = kernel.transpose([2, 3, 1, 0])
                    conv_layer.set_weights([kernel])
            except ValueError:
                print("no convolution #" + str(i))

    def reset(self):
        self.offset = 0
```

In [24]:

```
# define the model
model = make_yolov3_model()
# load the model weights
weight_reader = WeightReader('./yolov3.weights')
# set the model weights into the model
weight_reader.load_weights(model)
# # save the model to file
# model.save('/content/drive/My Drive/DPprojects/Object Detection - Yolo/model/model.h5')


# model.summary()
```

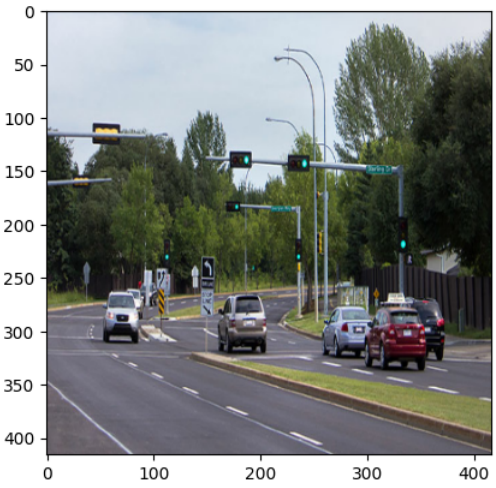
```
loading weights of convolution #0
loading weights of convolution #1
loading weights of convolution #2
loading weights of convolution #3
no convolution #4
loading weights of convolution #5
loading weights of convolution #6
loading weights of convolution #7
no convolution #8
loading weights of convolution #9
loading weights of convolution #10
no convolution #11
loading weights of convolution #12
loading weights of convolution #13
loading weights of convolution #14
no convolution #15
loading weights of convolution #16
loading weights of convolution #17
no convolution #18
loading weights of convolution #19
loading weights of convolution #20
no convolution #21
loading weights of convolution #22
loading weights of convolution #23
no convolution #24
loading weights of convolution #25
loading weights of convolution #26
no convolution #27
loading weights of convolution #28
loading weights of convolution #29
no convolution #30
loading weights of convolution #31
loading weights of convolution #32
no convolution #33
loading weights of convolution #34
loading weights of convolution #35
no convolution #36
loading weights of convolution #37
loading weights of convolution #38
loading weights of convolution #39
no convolution #40
loading weights of convolution #41
loading weights of convolution #42
no convolution #43
loading weights of convolution #44
loading weights of convolution #45
no convolution #46
loading weights of convolution #47
loading weights of convolution #48
no convolution #49
loading weights of convolution #50
loading weights of convolution #51
no convolution #52
loading weights of convolution #53
loading weights of convolution #54
no convolution #55
loading weights of convolution #56
loading weights of convolution #57
no convolution #58
loading weights of convolution #59
loading weights of convolution #60
no convolution #61
loading weights of convolution #62
loading weights of convolution #63
loading weights of convolution #64
no convolution #65
loading weights of convolution #66
loading weights of convolution #67
no convolution #68
loading weights of convolution #69
loading weights of convolution #70
no convolution #71
loading weights of convolution #72
loading weights of convolution #73
no convolution #74
loading weights of convolution #75
loading weights of convolution #76
loading weights of convolution #77
loading weights of convolution #78
loading weights of convolution #79
loading weights of convolution #80
loading weights of convolution #81
no convolution #82
no convolution #83
loading weights of convolution #84
no convolution #85
no convolution #86
loading weights of convolution #87
loading weights of convolution #88
loading weights of convolution #89
loading weights of convolution #90
loading weights of convolution #91
loading weights of convolution #92
loading weights of convolution #93
no convolution #94
no convolution #95
loading weights of convolution #96
no convolution #97
no convolution #98
loading weights of convolution #99
```

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

In [8]:
```
labels = ["person", "bicycle", "car", "motorbike", "aeroplane", "bus", "train", "truck",
        "boat", "traffic light", "fire hydrant", "stop sign", "parking meter", "bench",
        "bird", "cat", "dog", "horse", "sheep", "cow", "elephant", "bear", "zebra", "giraffe",
        "backpack", "umbrella", "handbag", "tie", "suitcase", "frisbee", "skis", "snowboard",
        "sports ball", "kite", "baseball bat", "baseball glove", "skateboard", "surfboard",
        "tennis racket", "bottle", "wine glass", "cup", "fork", "knife", "spoon", "bowl", "banana",
        "apple", "sandwich", "orange", "broccoli", "carrot", "hot dog", "pizza", "donut", "cake",
        "chair", "sofa", "pottedplant", "bed", "diningtable", "toilet", "tvmonitor", "laptop", "mouse",
        "remote", "keyboard", "cell phone", "microwave", "oven", "toaster", "sink", "refrigerator",
        "book", "clock", "vase", "scissors", "teddy bear", "hair drier", "toothbrush"]
IMAGE_WIDTH=416
IMAGE_HEIGHT=416
def load_and_preprocess_image(path,shape):
  image=tf.io.read_file(path)
  width,height=load_img(path).size
  image=tf.image.decode_jpeg(image,channels=3)
  image=tf.image.resize(image, shape)
  image/=255
  return image,width,height
```

In [9]:
```
photo_filename='./traffic.jpg'
_image, image_w, image_h=load_and_preprocess_image(photo_filename,[IMAGE_WIDTH,IMAGE_HEIGHT])
plt.imshow(_image)
```

Out[9]: <matplotlib.image.AxesImage at 0x1f3e1dc9d88>



In [10]:
```
image = expand_dims(_image, 0)
yhat = model.predict(image)
print([a.shape for a in yhat])
```

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

In [14]:
```
class BoundBox:
        '''
        Objects of boxes. (xmin,ymin) represents the upleft coordinate of the box while (xmax,ymax) means downright one.
        '''
        def __init__(self, xmin, ymin, xmax, ymax, objness = None, classes = None):
                self.xmin = xmin
                self.ymin = ymin
                self.xmax = xmax
                self.ymax = ymax
                self.objness = objness
                self.classes = classes
                self.label = -1
                self.score = -1

        def get_label(self):
                if self.label == -1:
                        self.label = np.argmax(self.classes)

                return self.label

        def get_score(self):
                if self.score == -1:
                        self.score = self.classes[self.get_label()]

                return self.score

def _sigmoid(x):
        return 1. / (1. + np.exp(-x))

def decode_netout(netout, anchors, net_h, net_w):
        grid_h, grid_w = netout.shape[:2]
        nb_box = 3
        netout = netout.reshape((grid_h, grid_w, nb_box, -1))
        nb_class = netout.shape[-1] - 5
        boxes = []
```

```
        netout[..., :2]  = _sigmoid(netout[..., :2])
        netout[..., 4:]  = _sigmoid(netout[..., 4:])
        netout[..., 5:]  = netout[..., 4][..., np.newaxis] * netout[..., 5:]

        for i in range(grid_h*grid_w):
                row = i / grid_w
                col = i % grid_w
                for b in range(nb_box):
                        # 4th element is objectness score
                        objectness = netout[int(row)][int(col)][b][4]
                        # if(objectness.all() <= obj_thresh): continue
                        # first 4 elements are x, y, w, and h
                        x, y, w, h = netout[int(row)][int(col)][b][:4]
                        x = (col + x) / grid_w # center position, unit: image width
                        y = (row + y) / grid_h # center position, unit: image height
                        w = anchors[2 * b + 0] * np.exp(w) / net_w # unit: image width
                        h = anchors[2 * b + 1] * np.exp(h) / net_h # unit: image height
                        # last elements are class probabilities
                        classes = netout[int(row)][col][b][5:]
                        box = BoundBox(x-w/2, y-h/2, x+w/2, y+h/2, objectness, classes)
                        boxes.append(box)
        return boxes
```

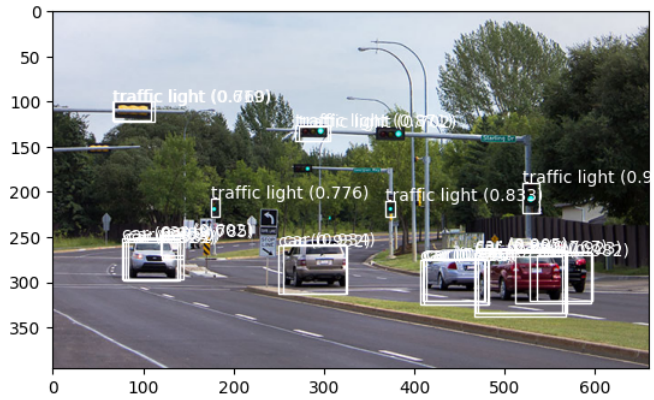
In [15]:
```
anchors = [[116,90, 156,198, 373,326], [30,61, 62,45, 59,119], [10,13, 16,30, 33,23]]
boxes = list()
for i in range(len(yhat)):
        boxes += decode_netout(yhat[i][0], anchors[i], net_h=IMAGE_HEIGHT, net_w=IMAGE_WIDTH)

for i in range(len(boxes)):
        x_offset, x_scale = (IMAGE_WIDTH - IMAGE_WIDTH)/2./IMAGE_HEIGHT, float(IMAGE_WIDTH)/IMAGE_WIDTH
        y_offset, y_scale = (IMAGE_HEIGHT - IMAGE_HEIGHT)/2./IMAGE_HEIGHT, float(IMAGE_HEIGHT)/IMAGE_HEIGHT
        boxes[i].xmin = int((boxes[i].xmin - x_offset) / x_scale * image_w)
        boxes[i].xmax = int((boxes[i].xmax - x_offset) / x_scale * image_w)
        boxes[i].ymin = int((boxes[i].ymin - y_offset) / y_scale * image_h)
        boxes[i].ymax = int((boxes[i].ymax - y_offset) / y_scale * image_h)
len(boxes)
```

Out[15]:
```
10647
```

In [16]:
```
def box_filter(boxes,labels,threshold_socre):
        valid_boxes=[]
        valid_labels=[]
        valid_scores=[]
        for box in boxes:
                for i in range(len(labels)):
                        if box.classes[i] > threshold_socre:
                                valid_boxes.append(box)
                                valid_labels.append(labels[i])
                                valid_scores.append(box.classes[i])

        return (valid_boxes,valid_labels,valid_scores)
valid_data= box_filter(boxes, labels, threshold_socre=0.6)
```

In [17]:
```
def draw_boxes(filename, valid_data):

        data = pyplot.imread(filename)
        pyplot.imshow(data)
        ax = pyplot.gca()
        for i in range(len(valid_data[0])):
                box = valid_data[0][i]
                y1, x1, y2, x2 = box.ymin, box.xmin, box.ymax, box.xmax
                width, height = x2 - x1, y2 - y1
                rect = Rectangle((x1, y1), width, height, fill=False, color='white')
                ax.add_patch(rect)
                print(valid_data[1][i], valid_data[2][i])
                label = "%s (%.3f)" % (valid_data[1][i], valid_data[2][i])
                pyplot.text(x1, y1, label, color='white')
        pyplot.show()
draw_boxes(photo_filename,valid_data)
```

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



In [18]:

```
def encoder_dic(valid_data):
  data_dic={}
  (valid_boxes,valid_labels,valid_scores)=valid_data
  for box, label,score in zip(valid_boxes,valid_labels,valid_scores):
    if label not in data_dic:
      data_dic[label]=[[score,box,'kept']]
    else:
      data_dic[label].append([score,box,'kept'])

  return data_dic
dic=encoder_dic(valid_data)


def decode_box_coor(box):
  return (box.xmin, box.ymin,box.xmax, box.ymax )

def iou(box1, box2):
  (box1_x1, box1_y1, box1_x2, box1_y2) = decode_box_coor(box1)
  (box2_x1, box2_y1, box2_x2, box2_y2) = decode_box_coor(box2)

  xi1 = max(box1_x1,box2_x1)
  yi1 = max(box1_y1,box2_y1)
  xi2 = min(box1_x2,box2_x2)
  yi2 = min(box1_y2,box2_y2)
  inter_width = xi2-xi1
  inter_height = yi2-yi1
  inter_area = max(inter_height,0)*max(inter_width,0)

  box1_area = (box1_x2-box1_x1)*(box1_y2-box1_y1)
  box2_area = (box2_x2-box2_x1)*(box2_y2-box2_y1)
  union_area = box1_area+box2_area-inter_area

  iou = inter_area/union_area

  return iou

def do_nms(data_dic, nms_thresh):
  final_boxes,final_scores,final_labels=list(),list(),list()
  for label in data_dic:
    scores_boxes=sorted(data_dic[label],reverse=True)
    for i in range(len(scores_boxes)):
      if scores_boxes[i][2]=='removed': continue
      for j in range(i+1,len(scores_boxes)):
        if iou(scores_boxes[i][1],scores_boxes[j][1]) >= nms_thresh:
          scores_boxes[j][2]="removed"

    for e in scores_boxes:
      print(label+' '+str(e[0]) + " status: "+ e[2])
      if e[2]=='kept':
        final_boxes.append(e[1])
        final_labels.append(label)
        final_scores.append(e[0])


  return (final_boxes,final_labels,final_scores)
final_data=do_nms(dic, 0.7)
```

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

In [19]:
```
draw_boxes(photo_filename,final_data)
```

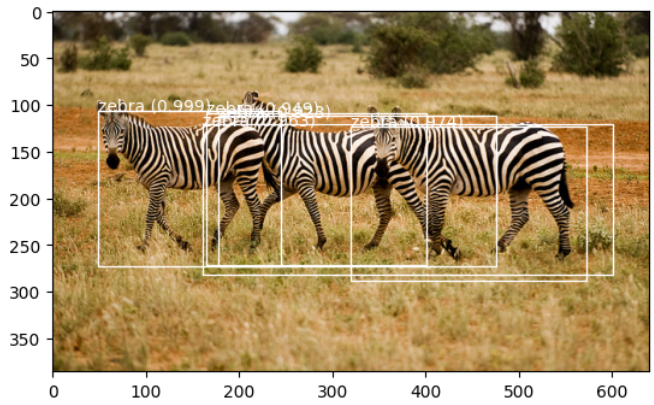
```
car 0.9982241
car 0.98843783
car 0.9822434
car 0.98158294
car 0.9342798
car 0.78529125
car 0.76686305
traffic light 0.964606
traffic light 0.87137073
traffic light 0.8328069
traffic light 0.77623147
traffic light 0.75971967
```

In [20]:
```
def yolo_non_max_suppression(scores, boxes, classes, max_boxes = 10, iou_threshold = 0.5):

    max_boxes_tensor = K.variable(max_boxes, dtype='int32')     # tensor to be used in tf.image.non_max_suppression()
    K.get_session().run(tf.variables_initializer([max_boxes_tensor])) # initialize variable max_boxes_tensor
    nms_indices = tf.image.non_max_suppression(scores=scores,boxes=boxes,max_output_size=max_boxes,iou_threshold=iou_threshold)

    scores = K.gather(scores,nms_indices)
    boxes = K.gather(boxes,nms_indices)
    classes = K.gather(classes,nms_indices)

    return scores, boxes, classes
```

In [21]:
```
def showresults(path):
  _image,width,height=load_and_preprocess_image(path,[IMAGE_WIDTH,IMAGE_HEIGHT])
  image = expand_dims(_image, 0)
  yhat = model.predict(image)
  boxes = list()
  for i in range(len(yhat)):
    boxes += decode_netout(yhat[i][0], anchors[i], net_h=IMAGE_HEIGHT, net_w=IMAGE_WIDTH)
  for i in range(len(boxes)):
    x_offset, x_scale = (IMAGE_WIDTH - IMAGE_WIDTH)/2./IMAGE_HEIGHT, float(IMAGE_WIDTH)/IMAGE_WIDTH
    y_offset, y_scale = (IMAGE_HEIGHT - IMAGE_HEIGHT)/2./IMAGE_HEIGHT, float(IMAGE_HEIGHT)/IMAGE_HEIGHT
    boxes[i].xmin = int((boxes[i].xmin - x_offset) / x_scale * image_w)
    boxes[i].xmax = int((boxes[i].xmax - x_offset) / x_scale * image_w)
    boxes[i].ymin = int((boxes[i].ymin - y_offset) / y_scale * image_h)
    boxes[i].ymax = int((boxes[i].ymax - y_offset) / y_scale * image_h)
  valid_data= box_filter(boxes, labels, threshold_socre=0.6)
  dic=encoder_dic(valid_data)
  final_data=do_nms(dic, 0.7)
  draw_boxes(path,final_data)
```

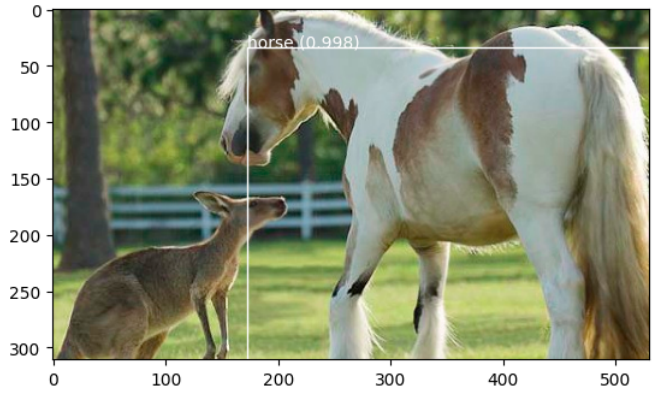
In [22]:
```
showresults('./zebra.jpg')
```

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



In [23]:
```
showresults('./kangaroo.png')
```

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



In [ ]:
```
```

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/

In [1]:
```
import tensorflow as tf
```

In [2]:
```
tf.__version__
```

Out[2]:
```
'2.19.0'
```

In [3]:
```
# To generate GIFs
!pip install imageio
!pip install git+https://github.com/tensorflow/docs
```

```
Requirement already satisfied: imageio in /usr/local/lib/python3.12/dist-pack
ages (2.37.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packag
es (from imageio) (2.0.2)
Requirement already satisfied: pillow>=8.3.2 in /usr/local/lib/python3.12/dis
t-packages (from imageio) (11.3.0)
Collecting git+https://github.com/tensorflow/docs
  Cloning https://github.com/tensorflow/docs to /tmp/pip-req-build-k46btmga
  Running command git clone --filter=blob:none --quiet https://github.com/ten
sorflow/docs /tmp/pip-req-build-k46btmga
  Resolved https://github.com/tensorflow/docs to commit e21d085d5ed82504ffcec
11aa82ebc78f1f2302e
  Preparing metadata (setup.py) ... done
Collecting astor (from tensorflow-docs==2025.3.6.10029)
  Downloading astor-0.8.1-py2.py3-none-any.whl.metadata (4.2 kB)
Requirement already satisfied: absl-py in /usr/local/lib/python3.12/dist-pack
ages (from tensorflow-docs==2025.3.6.10029) (1.4.0)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-packa
ges (from tensorflow-docs==2025.3.6.10029) (3.1.6)
Requirement already satisfied: nbformat in /usr/local/lib/python3.12/dist-pac
kages (from tensorflow-docs==2025.3.6.10029) (5.10.4)
Requirement already satisfied: protobuf>=3.12 in /usr/local/lib/python3.12/di
st-packages (from tensorflow-docs==2025.3.6.10029) (5.29.5)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.12/dist-packa
ges (from tensorflow-docs==2025.3.6.10029) (6.0.2)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/d
ist-packages (from jinja2->tensorflow-docs==2025.3.6.10029) (3.0.2)
Requirement already satisfied: fastjsonschema>=2.15 in /usr/local/lib/python
3.12/dist-packages (from nbformat->tensorflow-docs==2025.3.6.10029) (2.21.2)
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.12/d
ist-packages (from nbformat->tensorflow-docs==2025.3.6.10029) (4.25.1)
Requirement already satisfied: jupyter-core!=5.0.*,>=4.12 in /usr/local/lib/p
ython3.12/dist-packages (from nbformat->tensorflow-docs==2025.3.6.10029) (5.
8.1)
Requirement already satisfied: traitlets>=5.1 in /usr/local/lib/python3.12/di
st-packages (from nbformat->tensorflow-docs==2025.3.6.10029) (5.7.1)
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.12/dis
t-packages (from jsonschema>=2.6->nbformat->tensorflow-docs==2025.3.6.10029)
(25.3.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/l
ocal/lib/python3.12/dist-packages (from jsonschema>=2.6->nbformat->tensorflow
-docs==2025.3.6.10029) (2025.9.1)
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.
12/dist-packages (from jsonschema>=2.6->nbformat->tensorflow-docs==2025.3.6.1
0029) (0.36.2)
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.12/di
st-packages (from jsonschema>=2.6->nbformat->tensorflow-docs==2025.3.6.10029)
```

```
(0.27.1)
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.1
2/dist-packages (from jupyter-core!=5.0.*,>=4.12->nbformat->tensorflow-docs==
2025.3.6.10029) (4.4.0)
Requirement already satisfied: typing-extensions>=4.4.0 in /usr/local/lib/pyt
hon3.12/dist-packages (from referencing>=0.28.4->jsonschema>=2.6->nbformat->t
ensorflow-docs==2025.3.6.10029) (4.15.0)
Downloading astor-0.8.1-py2.py3-none-any.whl (27 kB)
Building wheels for collected packages: tensorflow-docs
  Building wheel for tensorflow-docs (setup.py) ... done
  Created wheel for tensorflow-docs: filename=tensorflow_docs-2025.3.6.10029-
py3-none-any.whl size=186351 sha256=0c34c2c2ea736cf4fe5510ae462b0c7436428b2bd
e43b5a43a1b442dbda4c6c3
  Stored in directory: /tmp/pip-ephem-wheel-cache-uvuh7dmu/wheels/3e/88/34/48
d2789bc9d37b33ddce06bccc454fae0285e5396d0a5be9d9
Successfully built tensorflow-docs
Installing collected packages: astor, tensorflow-docs
Successfully installed astor-0.8.1 tensorflow-docs-2025.3.6.10029
```

In [4]:
```
import glob
import imageio
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
from tensorflow.keras import layers
import time

from IPython import display
```

In [5]:
```
(train_images, train_labels), (_, _) = tf.keras.datasets.mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-data
sets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

In [6]:
```
train_images = train_images.reshape(train_images.shape[0], 28, 28, 1).astyp
train_images = (train_images - 127.5) / 127.5  # Normalize the images to [-
```

In [7]:
```
BUFFER_SIZE = 60000
BATCH_SIZE = 256
```

In [8]:
```
# Batch and shuffle the data
train_dataset = tf.data.Dataset.from_tensor_slices(train_images).shuffle(BU
```

In [9]:
```
def make_generator_model():
    model = tf.keras.Sequential()
    model.add(layers.Dense(7*7*256, use_bias=False, input_shape=(100,)))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Reshape((7, 7, 256)))
    assert model.output_shape == (None, 7, 7, 256)  # Note: None is the bat

    model.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='
    assert model.output_shape == (None, 7, 7, 128)
```

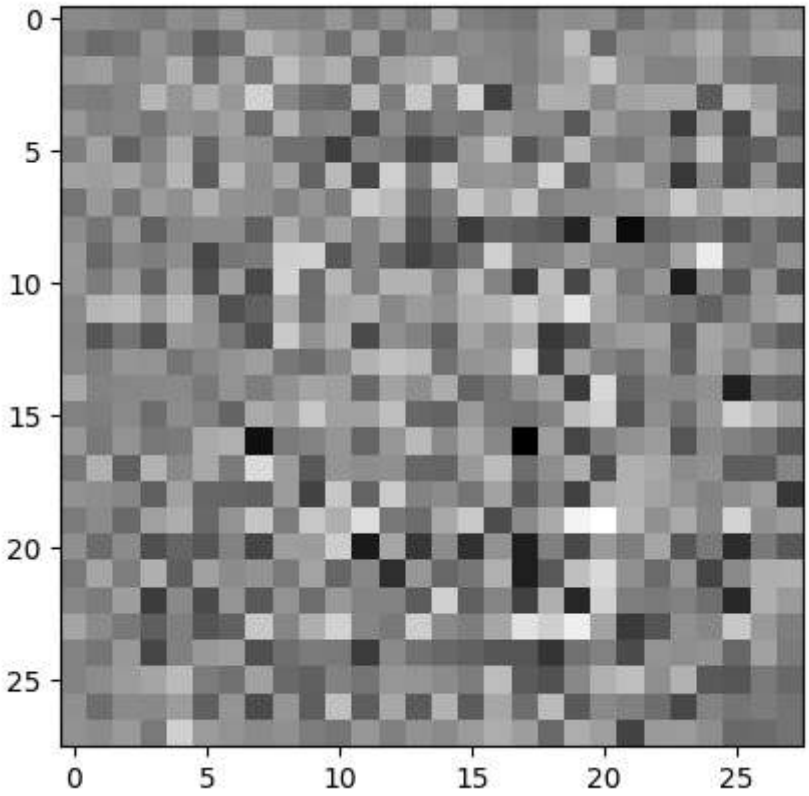
```
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='s
    assert model.output_shape == (None, 14, 14, 64)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(1, (5, 5), strides=(2, 2), padding='sa
    assert model.output_shape == (None, 28, 28, 1)

    return model
```

In [10]:

```
generator = make_generator_model()

noise = tf.random.normal([1, 100])
generated_image = generator(noise, training=False)

plt.imshow(generated_image[0, :, :, 0], cmap='gray')
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: Us
erWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first l
ayer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Out[10]: <matplotlib.image.AxesImage at 0x7caeb5703f50>



In [11]:

```
def make_discriminator_model():
    model = tf.keras.Sequential()
    model.add(layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same',
                                     input_shape=[28, 28, 1]))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
```

```
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Flatten())
    model.add(layers.Dense(1))

    return model
```

In [12]:
```
discriminator = make_discriminator_model()
decision = discriminator(generated_image)
print (decision)
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
tf.Tensor([[-0.00331897]], shape=(1, 1), dtype=float32)
```

In [13]:
```
# This method returns a helper function to compute cross entropy loss
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
```

In [14]:
```
def discriminator_loss(real_output, fake_output):
    real_loss = cross_entropy(tf.ones_like(real_output), real_output)
    fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
    total_loss = real_loss + fake_loss
    return total_loss
```

In [15]:
```
def generator_loss(fake_output):
    return cross_entropy(tf.ones_like(fake_output), fake_output)
```

In [16]:
```
generator_optimizer = tf.keras.optimizers.Adam(1e-4)
discriminator_optimizer = tf.keras.optimizers.Adam(1e-4)
```

In [17]:
```
checkpoint_dir = './training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt")
checkpoint = tf.train.Checkpoint(generator_optimizer=generator_optimizer,
                                 discriminator_optimizer=discriminator_opti
                                 generator=generator,
                                 discriminator=discriminator)
```

In [18]:
```
EPOCHS = 50
noise_dim = 100
num_examples_to_generate = 16

# You will reuse this seed overtime (so it's easier)
# to visualize progress in the animated GIF)
seed = tf.random.normal([num_examples_to_generate, noise_dim])
```

In [19]:
```
# Notice the use of `tf.function`
# This annotation causes the function to be "compiled".
@tf.function
def train_step(images):
```

```
    noise = tf.random.normal([BATCH_SIZE, noise_dim])

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
      generated_images = generator(noise, training=True)

      real_output = discriminator(images, training=True)
      fake_output = discriminator(generated_images, training=True)

      gen_loss = generator_loss(fake_output)
      disc_loss = discriminator_loss(real_output, fake_output)

    gradients_of_generator = gen_tape.gradient(gen_loss, generator.trainabl
    gradients_of_discriminator = disc_tape.gradient(disc_loss, discriminato

    generator_optimizer.apply_gradients(zip(gradients_of_generator, generat
    discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator,
```

In [20]:

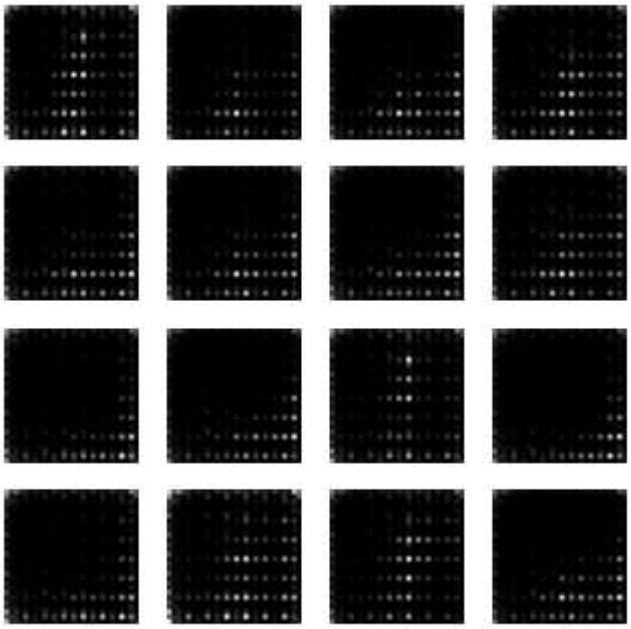
```
def train(dataset, epochs):
  for epoch in range(epochs):
    start = time.time()

    for image_batch in dataset:
      train_step(image_batch)

    # Produce images for the GIF as you go
    # display.clear_output(wait=True)
    generate_and_save_images(generator,
                             epoch + 1,
                             seed)

    # Save the model every 15 epochs
    if (epoch + 1) % 15 == 0:
      checkpoint.save(file_prefix = checkpoint_prefix)

    print ('Time for epoch {} is {} sec'.format(epoch + 1, time.time()-star

  # Generate after the final epoch
  # display.clear_output(wait=True)
  generate_and_save_images(generator,
                           epochs,
                           seed)
```

In [21]:

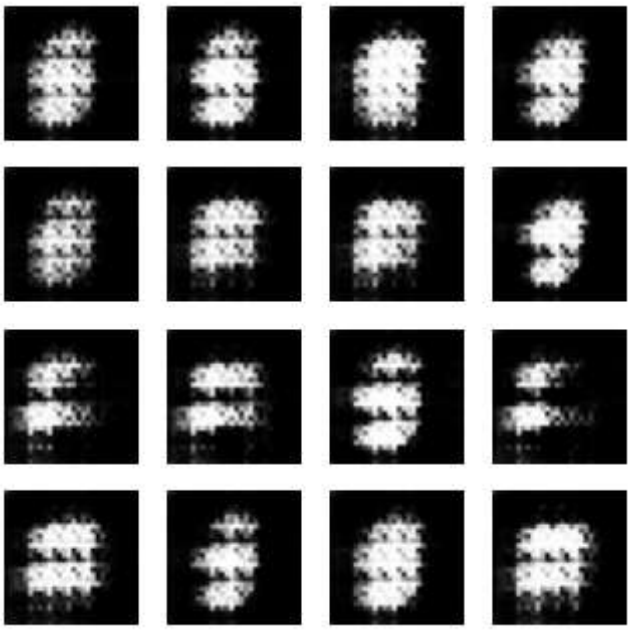
```
def generate_and_save_images(model, epoch, test_input):
  # Notice `training` is set to False.
  # This is so all layers run in inference mode (batchnorm).
  predictions = model(test_input, training=False)

  fig = plt.figure(figsize=(4, 4))

  for i in range(predictions.shape[0]):
      plt.subplot(4, 4, i+1)
      plt.imshow(predictions[i, :, :, 0] * 127.5 + 127.5, cmap='gray')
      plt.axis('off')

  plt.savefig('image_at_epoch_{:04d}.png'.format(epoch))
  plt.show()
```

In [22]:

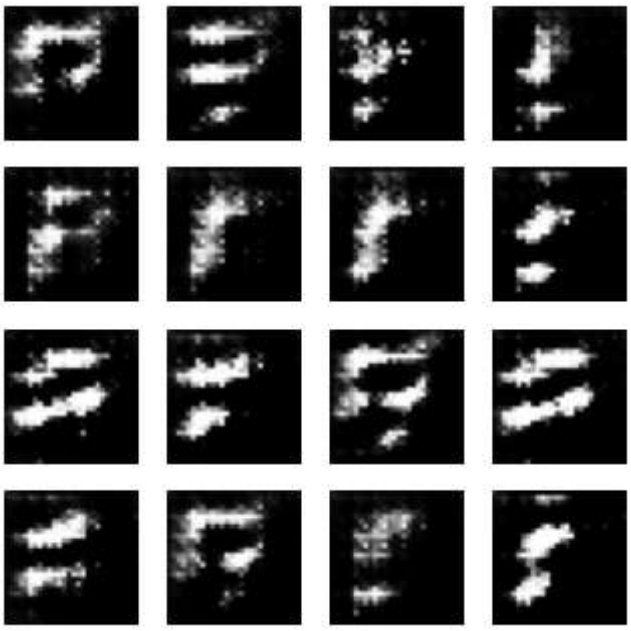
```
train(train_dataset, EPOCHS)
```



```
Time for epoch 1 is 41.512364625930786 sec
```



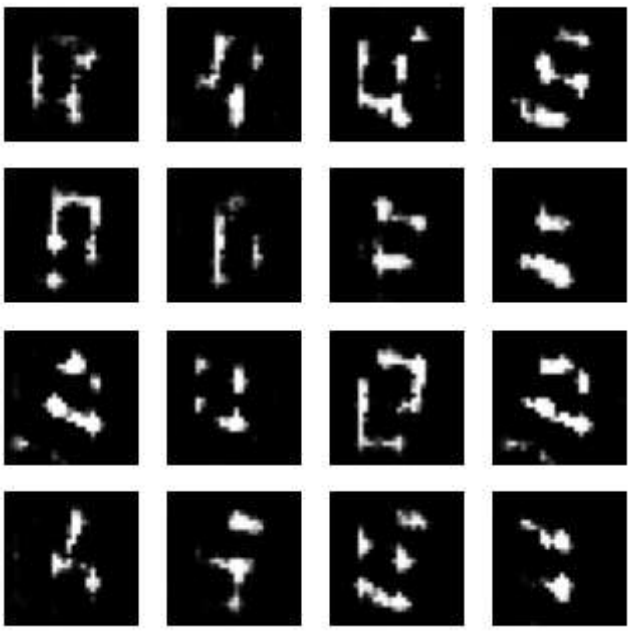
```
Time for epoch 2 is 12.868218898773193 sec
```



```
Time for epoch 3 is 12.971028566360474 sec
```

Time for epoch 3 is 12.971028566360474 sec

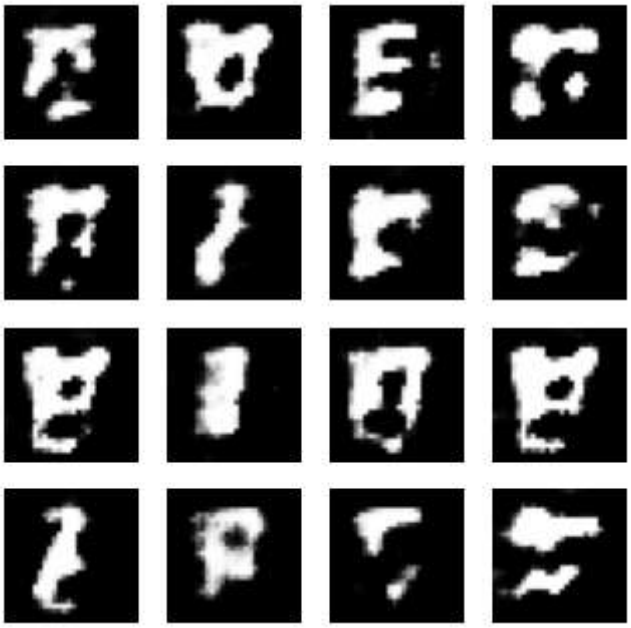
Time for epoch 4 is 13.403157949447632 sec



Time for epoch 5 is 13.294790506362915 sec

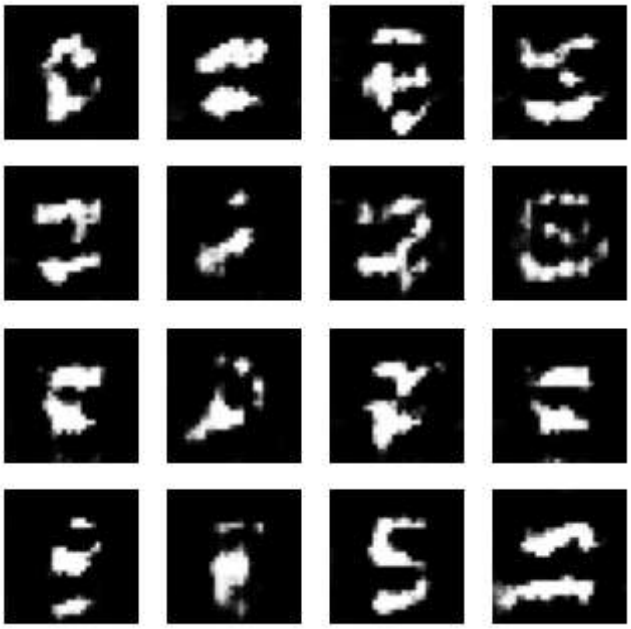
Time for epoch 6 is 13.452869176864624 sec

Time for epoch 7 is 13.467377662658691 sec



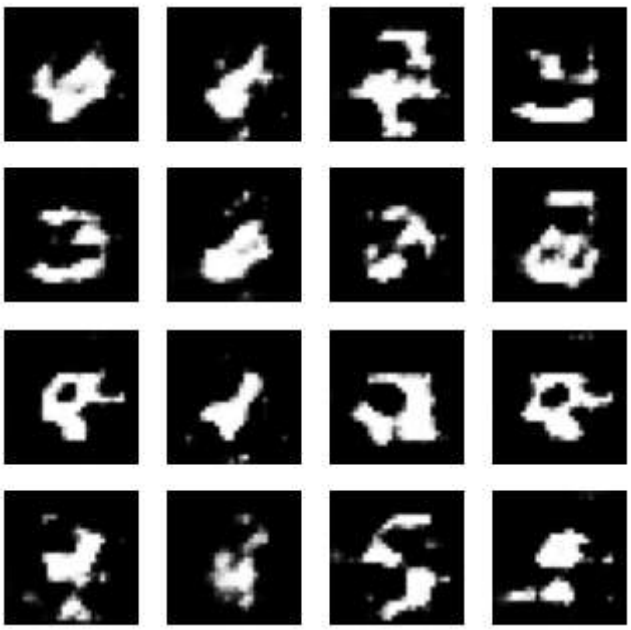
Time for epoch 8 is 13.146721839904785 sec

Time for epoch 9 is 13.12352705001831 sec



Time for epoch 10 is 13.1309072971344 sec

Time for epoch 11 is 13.476233720779419 sec

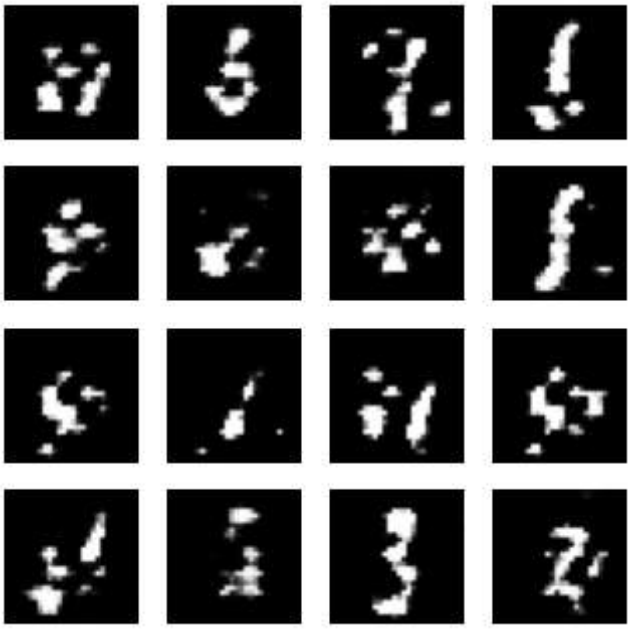


Time for epoch 12 is 13.232011079788208 sec

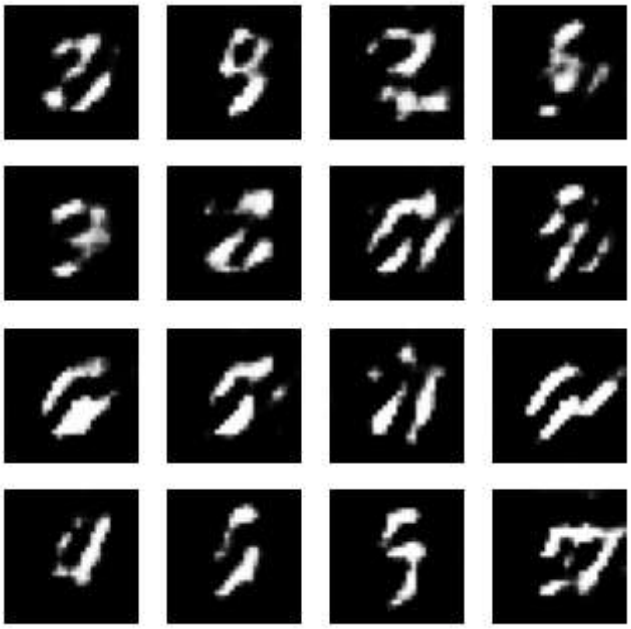
Time for epoch 13 is 13.205699443817139 sec

Time for epoch 14 is 13.20341157913208 sec

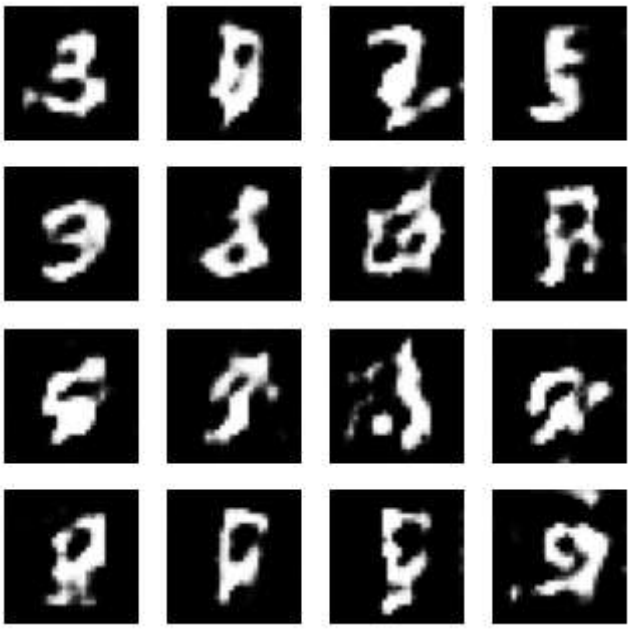
Time for epoch 15 is 13.50782585144043 sec



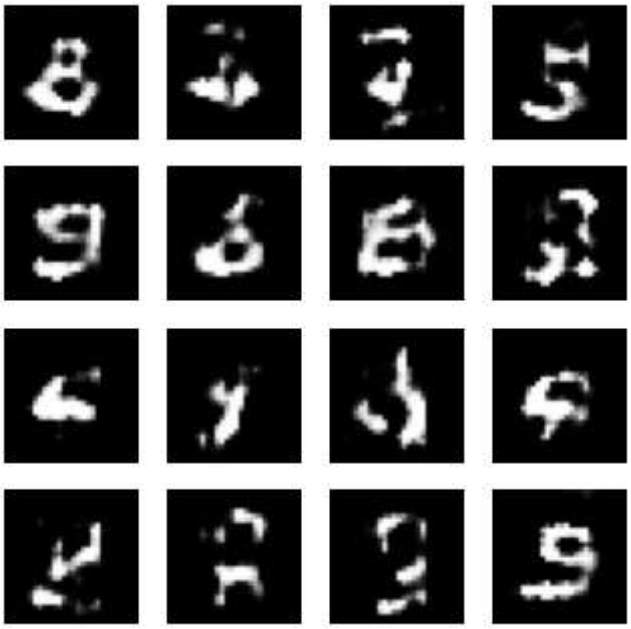
Time for epoch 16 is 13.190398693084717 sec



Time for epoch 17 is 13.499018907546997 sec



Time for epoch 18 is 13.22204875946045 sec

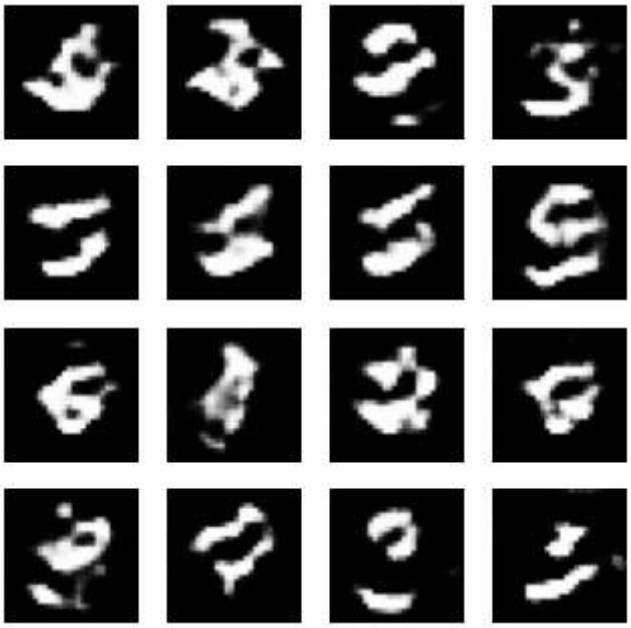


Time for epoch 19 is 13.225222110748291 sec

Time for epoch 20 is 13.21445369720459 sec

Time for epoch 21 is 13.206793546676636 sec

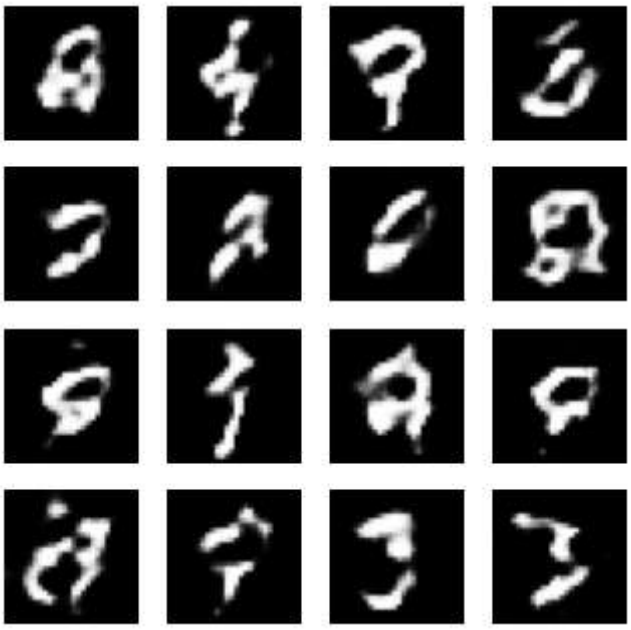
Time for epoch 22 is 13.174060344696045 sec



Time for epoch 23 is 13.519931554794312 sec

Time for epoch 24 is 13.3197603225708 sec

Time for epoch 25 is 13.267343521118164 sec



Time for epoch 26 is 13.207594156265259 sec

Time for epoch 27 is 13.218523502349854 sec

Time for epoch 28 is 13.225352048873901 sec

Time for epoch 29 is 13.5050630569458 sec

Time for epoch 30 is 13.336615085601807 sec

Time for epoch 31 is 13.199566364288333 sec

Time for epoch 32 is 13.311694622039795 sec

Time for epoch 33 is 13.33041787147522 sec

Time for epoch 34 is 13.225486755371094 sec

Time for epoch 35 is 13.194588899612427 sec

Time for epoch 36 is 13.460001945495605 sec

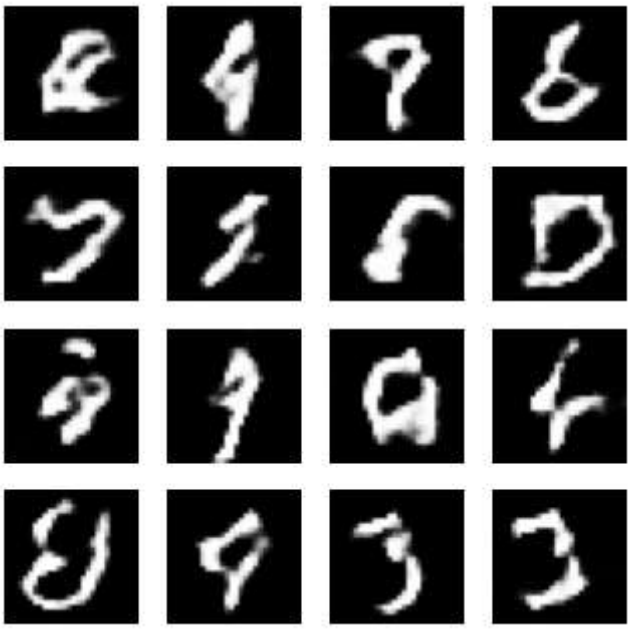
Time for epoch 37 is 13.209766387939453 sec

Time for epoch 38 is 13.211616039276123 sec

Time for epoch 39 is 13.226772785186768 sec

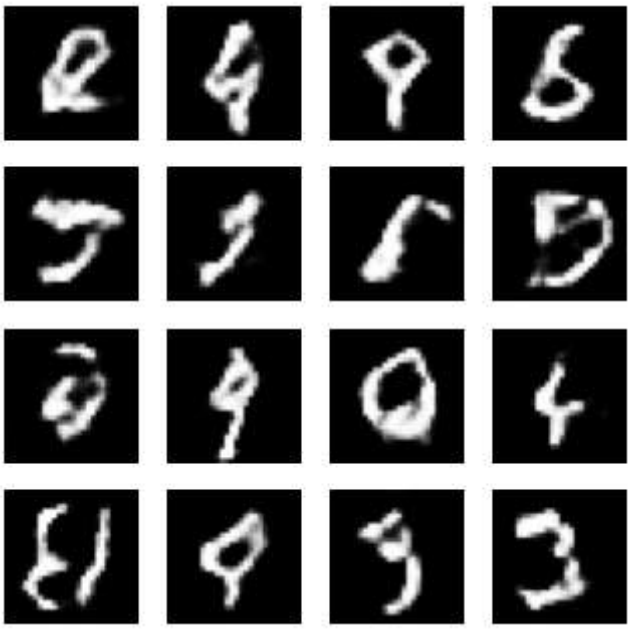
Time for epoch 40 is 13.22469186782837 sec

Time for epoch 41 is 13.33402156829834 sec

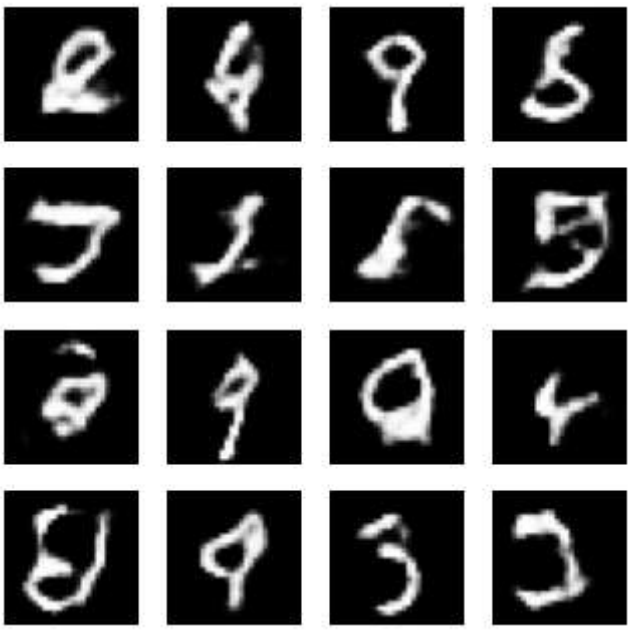


Time for epoch 42 is 13.565944194793701 sec

Time for epoch 43 is 13.210325956344604 sec



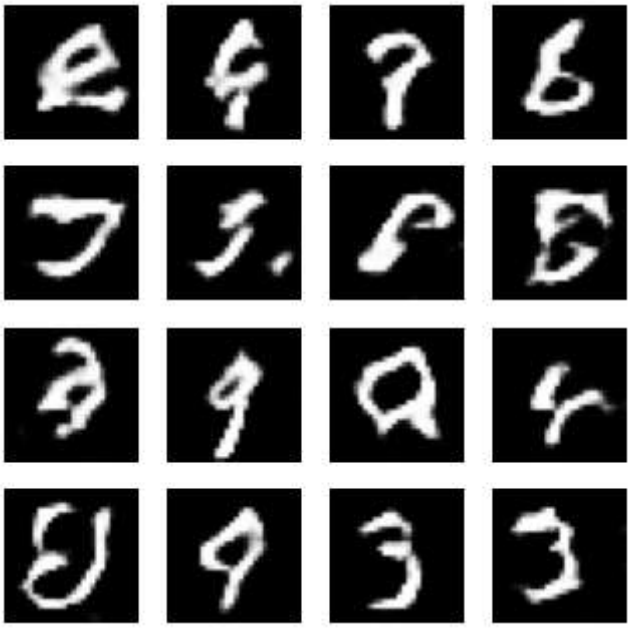
Time for epoch 44 is 13.226375102996826 sec



Time for epoch 45 is 13.357990741729736 sec

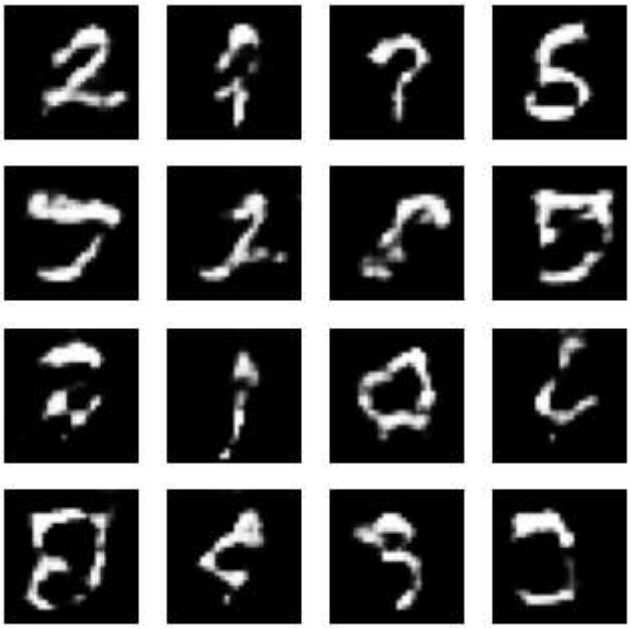
Time for epoch 46 is 13.242566585540771 sec

Time for epoch 47 is 13.22094440460205 sec

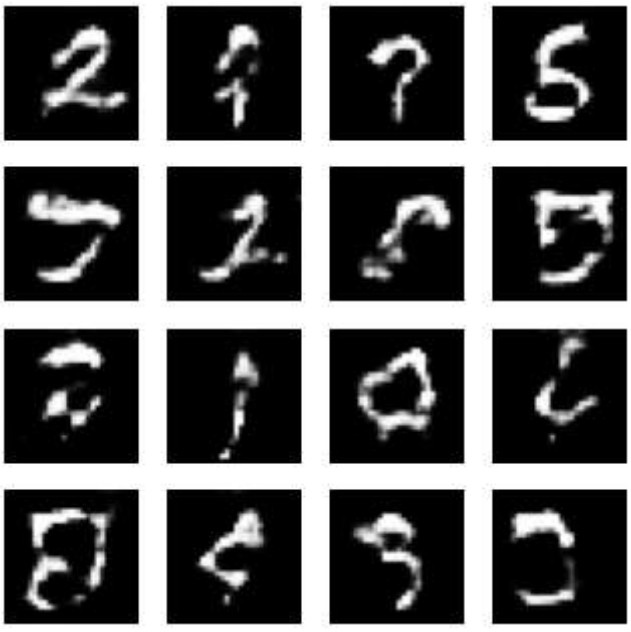


Time for epoch 48 is 13.52785587310791 sec

```
Time for epoch 49 is 13.285894632339478 sec
```



```
Time for epoch 50 is 13.408785581588745 sec
```



In [23]:

```
checkpoint.restore(tf.train.latest_checkpoint(checkpoint_dir))
```

Out[23]:
```
<tensorflow.python.checkpoint.checkpoint.CheckpointLoadStatus at 0x7cae7c75
cf50>
```

In [24]:
```
# Display a single image using the epoch number
def display_image(epoch_no):
  return PIL.Image.open('image_at_epoch_{:04d}.png'.format(epoch_no))
```

In [25]:
```
display_image(EPOCHS)
```

Out[25]:

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.