Experiment - 1

Aim:

Overview of different advanced ML and DL techniques and their applications.

Theory:

Machine Learning (ML) and Deep Learning (DL) have rapidly evolved over the past decade, becoming central to solving complex real-world problems across various domains, from healthcare to finance. As the field advances, new techniques and methods have emerged, offering more sophisticated approaches to data analysis and pattern recognition. These advanced methods enable machines to learn from data in increasingly efficient, accurate, and scalable ways. This overview explores some of the most influential and advanced techniques in ML and DL, their benefits, and their applications in diverse industries.

1. Ensemble Learning

Ensemble learning is a powerful technique that combines predictions from multiple models to enhance the overall performance and robustness. The key idea behind ensemble methods is that by leveraging different models or variations of a model, the final prediction can be more accurate and stable than any individual model. The two most popular ensemble learning methods are **Bagging** and **Boosting**.

- **Bagging** (Bootstrap Aggregating) involves training multiple instances of the same learning algorithm on different subsets of the training data, which are generated by random sampling with replacement. The results of these models are then aggregated to form the final prediction. A well-known example of this is the **Random Forest** algorithm, which combines multiple decision trees to improve accuracy and reduce overfitting. Bagging is particularly useful when the base model is prone to high variance, such as decision trees. It is commonly used in areas like image classification, financial predictions, and bioinformatics.

- **Boosting** is a sequential ensemble method where each new model is trained to correct the errors made by previous models. Models are built one after another, and the predictions of each model are weighted based on their accuracy. Boosting techniques like **AdaBoost** and **Gradient Boosting** (e.g., **XGBoost**) are widely used for classification and regression problems. These algorithms have been highly effective in various domains, including fraud detection, medical diagnostics, and stock market forecasting, as they can significantly improve the model's predictive power by focusing on the hardest-to-predict examples.

Both methods aim to reduce bias (Boosting) or variance (Bagging) in the models, making them indispensable in real-world machine learning tasks where high accuracy is critical.

2. Dimensionality Reduction Techniques

Dimensionality reduction is a crucial step in machine learning and deep learning, especially when dealing with high-dimensional data, which can be computationally expensive and lead to overfitting. These techniques aim to reduce the number of features (or dimensions) while preserving the essential information in the data. Among the most popular dimensionality reduction methods are **Principal Component Analysis (PCA)**, **t-Distributed Stochastic Neighbor Embedding (t-SNE)**, and **Autoencoders**.

- **PCA** is a linear technique that identifies the directions (principal components) in which the data varies the most and projects the data along these components to reduce its dimensionality. PCA is widely used in exploratory data analysis, image compression, and preprocessing for other machine learning algorithms. By reducing the complexity of the data, PCA helps to mitigate noise and improve computational efficiency without losing significant information.
- **t-SNE**, on the other hand, is a non-linear technique primarily used for visualizing high-dimensional datasets in a lower-dimensional space (usually 2D or 3D). Unlike PCA, t-SNE focuses on preserving the local structure of the data, making it ideal for clustering or visualizing complex patterns in data, such as word embeddings in NLP or images in computer vision.

- **Autoencoders** are a type of neural network used for unsupervised learning of compressed representations of data. The network consists of an encoder that compresses the data into a lower-dimensional space and a decoder that reconstructs the original data from this compressed representation. Autoencoders have found applications in anomaly detection, image denoising, and data compression, where the goal is to extract meaningful features from the data in an unsupervised manner.

These dimensionality reduction techniques are essential tools in modern machine learning workflows, especially when working with large datasets or when the data contains irrelevant or redundant features.

3. Deep Learning Architectures

Deep learning has revolutionized various fields by allowing models to learn complex patterns directly from raw data. The most commonly used deep learning architectures include **Convolutional Neural Networks (CNNs)**, **Recurrent Neural Networks (RNNs)**, and **Generative Adversarial Networks (GANs)**.

- **CNNs** are designed to process grid-like data, such as images, by applying convolutional filters that capture local patterns (edges, textures, etc.). These networks have been highly successful in computer vision tasks such as image classification, object detection, and facial recognition. CNNs are composed of layers like convolutional layers, pooling layers, and fully connected layers, which work together to extract hierarchical features from raw data. Applications of CNNs extend beyond images to fields like video analysis, medical imaging, and autonomous vehicles, where accurate object recognition and classification are crucial.
- **RNNs** are designed for sequential data, where the output from previous steps influences the current step. This makes RNNs particularly suitable for tasks like time series forecasting, speech recognition, and natural language processing (NLP). However, traditional RNNs suffer from the **vanishing gradient problem**, where they struggle to learn long-term dependencies. **Long Short-Term Memory (LSTM)** networks were introduced to address this problem by introducing memory cells that can remember

information for long periods. LSTMs have been instrumental in improving performance on tasks such as language translation, speech synthesis, and sentiment analysis.

- **GANs** consist of two neural networks—**the generator** and **the discriminator**—that are trained in opposition to each other. The generator creates fake data, while the discriminator tries to distinguish between real and fake data. Over time, the generator improves and can generate highly realistic data. GANs have been used in image generation (e.g., generating realistic images from textual descriptions), video prediction, and even artistic style transfer.

Deep learning architectures, especially CNNs, RNNs, and GANs, are powerful tools in modern AI systems, enabling applications in diverse fields such as healthcare, entertainment, and autonomous systems.

4. Transfer Learning

Transfer learning is a technique in machine learning where a model developed for a particular task is reused as the starting point for a model on a second task. In deep learning, this typically involves using a pre-trained model on a large dataset (such as ImageNet for images or BERT for text) and fine-tuning it on a smaller, domain-specific dataset. This approach has gained popularity because it allows models to leverage knowledge gained from large datasets, even when the new task has limited data.

For example, in image classification, models like **ResNet** or **VGG**, which have been pre-trained on millions of images, can be fine-tuned to classify new objects in a smaller, specific dataset. This significantly reduces the training time and computational resources required, making it ideal for applications where data collection is expensive or time-consuming, such as in medical imaging, where annotated datasets are limited.

Transfer learning is widely used in fields such as NLP and computer vision, where pre-trained models on large corpora or datasets provide a strong foundation for domain-specific tasks like text classification, sentiment analysis, and object recognition.

Conclusion:

Advanced machine learning and deep learning techniques have significantly transformed the landscape of artificial intelligence, enabling applications in diverse fields such as healthcare, finance, robotics, and entertainment. From ensemble methods that combine multiple models to deep learning architectures like CNNs and GANs that can learn complex patterns from raw data, these techniques are shaping the future of AI. As research continues and computational power increases, the applications and effectiveness of these techniques are expected to expand even further.

Experiment - 2

Aim:

Implement a deep neural network from scratch using TensorFlow or PyTorch, gaining hands-on experience in building complex neural architectures.

Theory:

This project implements an Artificial Neural Network (ANN) for binary classification, inspired by the structure and function of the human brain. ANNs consist of interconnected layers of nodes (neurons) that process data by applying weighted transformations and passing results through activation functions, which introduce non-linearity. This ability to capture complex patterns makes ANNs powerful for tasks like classification and regression. The ANN model in this project includes an input layer with 8 features, two hidden layers of 20 neurons each with ReLU activation, and an output layer for the two classes (diabetes or no diabetes). Using Cross Entropy Loss to calculate prediction errors and the Adam optimizer for weight adjustments, the model iteratively minimizes error and improves accuracy over 500 epochs.

Dataset Description:

The **Pima Indians Diabetes Database** (available on Kaggle) contains medical records for women of Pima Indian heritage, with the goal of predicting diabetes onset based on diagnostic variables. This dataset has 768 entries and 8 feature columns, with one target column, "Outcome," indicating diabetes status (1 for positive, 0 for negative).

Attributes (Features):

- **Pregnancies**: Number of times pregnant.
- **Glucose**: Plasma glucose concentration in an oral glucose tolerance test.
- **Blood Pressure**: Diastolic blood pressure (mm Hg).
- **Skin Thickness**: Triceps skin fold thickness (mm).
- **Insulin**: 2-Hour serum insulin (mu U/ml).
- **BMI**: Body mass index (weight in kg/(height in m)^2).

- **Diabetes Pedigree Function**: Likelihood of diabetes based on family history.
- **Age**: Age in years.
- **Outcome**: The target variable (0 = no diabetes, 1 = diabetes).

Code:

Create An ANN Using Pytorch

```
[3] !kaggle datasets download -d uciml/pima-indians-diabetes-database
```

```
Dataset URL: https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database
License(s): CC0-1.0
Downloading pima-indians-diabetes-database.zip to /content
  0% 0.00/8.91k [00:00<?, ?B/s]
100% 8.91k/8.91k [00:00<00:00, 20.6MB/s]
```

```
[4] import zipfile
    zip_ref = zipfile.ZipFile('/content/pima-indians-diabetes-database.zip', 'r')
    zip_ref.extractall('/content')
    zip_ref.close()
```

```
[5] import pandas as pd
```

```
[6] df = pd.read_csv("/content/diabetes.csv")
```

```
[7] df.isnull().sum()
```

```
[8] df.info()
```

```
[9] import seaborn as sns
```

```
[10] sns.pairplot(df, hue='Outcome')
```

```
[11] from sklearn.model_selection import train_test_split
     x = df.drop('Outcome', axis=1)
     y = df['Outcome']
     x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=0)
```

```
[12] #Libraries
     import torch
     import torch.nn as nn
     import torch.nn.functional as F
```

```
[13] # Creating Tensors
     x_train = x_train.values
     x_test = x_test.values
     y_train = y_train.values
     y_test = y_test.values

     x_train = torch.FloatTensor(x_train)
     x_test = torch.FloatTensor(x_test)
     y_train = torch.LongTensor(y_train)
     y_test = torch.LongTensor(y_test)
```

```
[14] # Creating model

class ANN_Model(nn.Module):

    def __init__(self, input_features=8, hidden1=20, hidden2=20, out_features=2):
        super().__init__()
        self.f_connected1 = nn.Linear(input_features, hidden1)
        self.f_connected2 = nn.Linear(hidden1, hidden2)
        self.out = nn.Linear(hidden2, out_features)

    def forward(self, x):
        x = F.relu(self.f_connected1(x))
        x = F.relu(self.f_connected2(x))
        x = self.out(x)
        return x
```

```
[15] ### instantiate my ANN Model
torch.manual_seed(20)
model = ANN_Model()
```

```
[16] model.parameters
```

```
[17] # Backward Propagation

loss_function = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
```

```
[18] epochs = 500
final_losses = []
for i in range(epochs):
    i = i + 1
    y_pred = model.forward(x_train)
    loss = loss_function(y_pred, y_train)
    final_losses.append(loss)
    if i%10 == 1:
        print("Epoch number: {} and the loss : {}".format(i,loss.item()))

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

```
[19] import matplotlib.pyplot as plt

# Detach the tensor from the computation graph and convert it to a numpy array
final_losses_np = [loss.detach().numpy() for loss in final_losses]

# Plotting
plt.plot(range(epochs), final_losses_np)
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.show()
```

```python
[20] # Prediction

     predictions = []
     with torch.no_grad():
         for i, data in enumerate(x_test):
             y_pred = model(data)
             predictions.append(y_pred.argmax().item())
```

```python
[21] print(predictions)
```

```python
[22] from sklearn.metrics import confusion_matrix
     cm = confusion_matrix(y_test, predictions)
     cm
```

```python
[23] plt.figure(figsize=(10,6))
     sns.heatmap(cm, annot=True)
     plt.xlabel('Actual Values')
     plt.ylabel('Predicted Values')
```

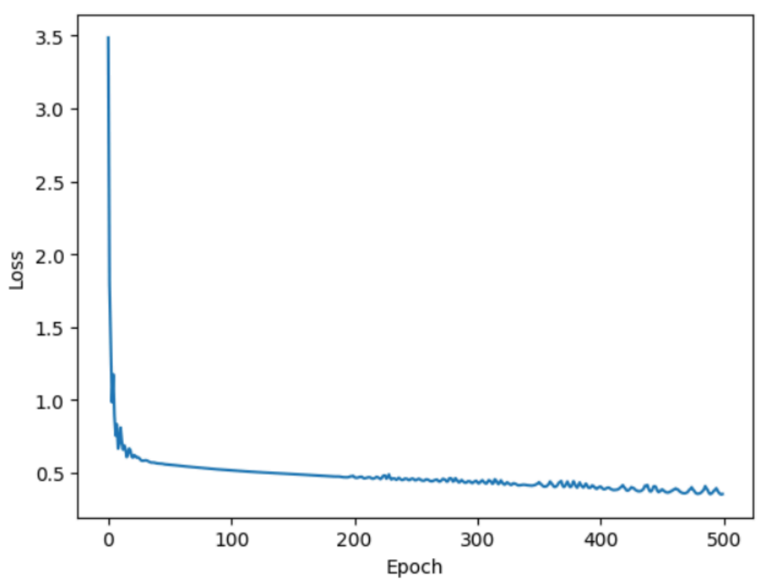
```python
[28] from sklearn.metrics import classification_report, accuracy_score
     print(classification_report(y_test, predictions))
     acc = accuracy_score(y_test, predictions) * 100
     print(acc)
```

```python
[25] torch.save(model, 'diabetes.pt')
```

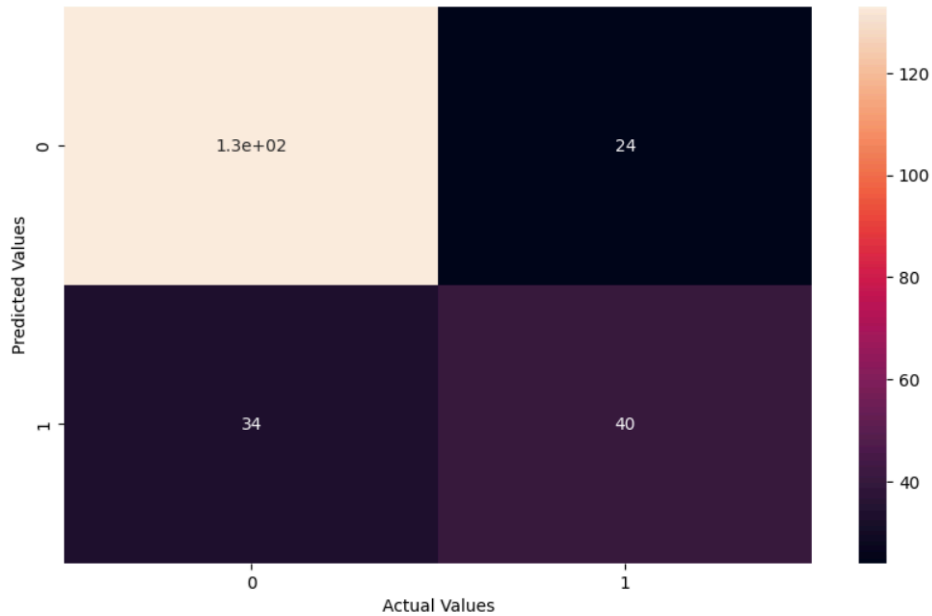
```python
[26] model = torch.load('diabetes.pt')
```

```python
model.eval()
```

Output:



```
[1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1,
```




```
              precision    recall  f1-score   support

           0       0.80      0.85      0.82       157
           1       0.62      0.54      0.58        74

    accuracy                           0.75       231
   macro avg       0.71      0.69      0.70       231
weighted avg       0.74      0.75      0.74       231

Accuracy is: 74.89177489177489
```

Conclusion:

The neural network model achieves an accuracy of 74.89%, performing better in identifying non-diabetic cases (class `0`) than diabetic cases (class `1`). The model's precision, recall, and F1-score are higher for class `0`, while a higher number of false negatives indicates some missed diabetic cases. The loss curve shows stable convergence around 200 epochs, suggesting effective learning of the training data.

Experiment - 3

Aim:

Utilize pre-trained models and perform transfer learning to solve real-world problems
Efficiently.

Theory:

Transfer learning is a machine learning technique where a model developed for one task is reused as the starting point for a model on a different task. It leverages knowledge learned from large, pre-trained models and adapts it to the current problem, which is particularly beneficial when data for the target task is limited. In this experiment, transfer learning enables us to use a powerful pre-trained model (VGG16) on MRI image data to classify brain MRI scans as either "Tumor" or "No Tumor."

Pre Trained Models and VGG16

The VGG16 model is a deep convolutional neural network trained on ImageNet, a large-scale dataset containing millions of labeled images across thousands of categories. The model's architecture, featuring multiple convolutional layers followed by pooling and fully connected layers, has demonstrated remarkable feature extraction capabilities for a wide range of image classification tasks. In transfer learning, VGG16 can serve as a feature extractor, capturing patterns like edges, shapes, and textures which are then used as inputs for the specific classification task of brain tumor detection.

Dataset Description:

The dataset consists of **Brain MRI images** categorized into two classes: "Tumor" and "No Tumor."

- **Tumor**: Images with visible signs of a brain tumor, showing irregular patterns or abnormalities in brain structure.
- **No Tumor**: Images of healthy brains with consistent structures and no signs of tumors.

Each image is resized to **224x224 pixels** to match the VGG16 input requirements. This dataset setup allows the model to learn to distinguish between normal and abnormal brain scans, aiding in efficient tumor detection.

Code:

```
[1] from google.colab import files
    uploaded = files.upload()

    for fn in uploaded.keys():
      print('User uploaded file "{name}" with length {length} bytes'.format(
          name=fn, length=len(uploaded[fn])))

    # Then move kaggle.json into the folder where the API expects to find it.
    !mkdir -p ~/.kaggle/ && mv kaggle.json ~/.kaggle/ && chmod 600 ~/.kaggle/kaggle.json
```

```
[2] !kaggle datasets download -d navoneel/brain-mri-images-for-brain-tumor-detection
```

```
[4] import tensorflow as tf
    from zipfile import ZipFile
    import os,glob
    import cv2
    from tqdm.notebook import tqdm_notebook as tqdm
    import numpy as np
    from sklearn import preprocessing
    from sklearn.model_selection import train_test_split
    from keras.models import Sequential
    from keras.layers import Convolution2D, Dropout, Dense,MaxPooling2D
    from keras.layers import BatchNormalization
    from keras.layers import MaxPooling2D
    from keras.layers import Flatten
    from zipfile import ZipFile
```

```
[5] file_name = "/content/brain-mri-images-for-brain-tumor-detection.zip"
    with ZipFile(file_name,'r') as zip:
      zip.extractall()
      print('Done')
```

```
[27]  os.chdir('/content/yes')
    X = []
    y = []
    for i in tqdm(os.listdir()):
          img = cv2.imread(i)
          img = cv2.resize(img,(224,224))
          X.append(img)
          y.append((i[0:1]))
          print(i[0:1])
    os.chdir('/content/no')
    for i in tqdm(os.listdir()):
          img = cv2.imread(i)
          img = cv2.resize(img,(224,224))
          X.append(img)
    for i in range(1,99):
        y.append('N')
    print(y)
```

[7]
```
%matplotlib inline
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 10))
for i in range(4):
    plt.subplot(1, 4, i+1)
    plt.imshow(X[i], cmap="gray")
    plt.axis('off')
plt.show()
```

[8]
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
print ("Shape of an image in X_train: ", X_train[0].shape)
print ("Shape of an image in X_test: ", X_test[0].shape)
```

```
le = preprocessing.LabelEncoder()
y_train = le.fit_transform(y_train)
y_test = le.fit_transform(y_test)
y_train = tf.keras.utils.to_categorical(y_train, num_classes=2)
y_test = tf.keras.utils.to_categorical(y_test, num_classes=2)
y_train = np.array(y_train)
X_train = np.array(X_train)
y_test = np.array(y_test)
X_test = np.array(X_test)
print("X_train Shape: ", X_train.shape)
print("X_test Shape: ", X_test.shape)
print("y_train Shape: ", y_train.shape)
print("y_test Shape: ", y_test.shape)
```

[11]
```
from keras.applications import vgg16
img_rows, img_cols = 224, 224
vgg = vgg16.VGG16(weights = 'imagenet',
                 include_top = False,
                 input_shape = (img_rows, img_cols, 3))
# Here we freeze the last 4 layers
# Layers are set to trainable as True by default
for layer in vgg.layers:
    layer.trainable = False
# Let's print our layers
for (i,layer) in enumerate(vgg.layers):
    print(str(i) + " "+ layer.__class__.__name__, layer.trainable)
```

[14]
```
def lw(bottom_model, num_classes):
    """creates the top or head of the model that will be
    placed ontop of the bottom layers"""

    top_model = bottom_model.output
    top_model = GlobalAveragePooling2D()(top_model)
    top_model = Dense(1024,activation='relu')(top_model)
    top_model = Dense(1024,activation='relu')(top_model)
    top_model = Dense(512,activation='relu')(top_model)
    top_model = Dense(num_classes,activation='sigmoid')(top_model)
    return top_model
```

```
[15] from keras.models import Sequential
     from keras.layers import Dense, Dropout, Activation, Flatten, GlobalAveragePooling2D
     from keras.layers import Conv2D, MaxPooling2D, ZeroPadding2D

     from keras.models import Model


     num_classes = 2

     FC_Head = lw(vgg, num_classes)

     model = Model(inputs = vgg.input, outputs = FC_Head)

     print(model.summary())
```

```
[16] from tensorflow.keras.models import Model
     model.compile(optimizer='adam', loss = 'categorical_crossentropy',metrics = ['accuracy'])
```

```
[17] history = model.fit(X_train,y_train,
                         epochs=5,
                         validation_data=(X_test,y_test),
                         verbose = 1,
                         initial_epoch=0)
```

```
[18] %matplotlib inline
     acc = history.history['accuracy']
     val_acc = history.history['val_accuracy']
     loss = history.history['loss']
     val_loss = history.history['val_loss']
     epochs = range(len(acc))
     plt.plot(epochs, acc, 'r', label='Training accuracy')
     plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
     plt.title('Training and validation accuracy')
     plt.legend(loc=0)
     plt.figure()
     plt.show()
```

```
[26] from sklearn.metrics import precision_score, recall_score, f1_score

     # Calculate metrics
     precision = precision_score(y_true_classes, y_pred_classes)
     recall = recall_score(y_true_classes, y_pred_classes)
     f1 = f1_score(y_true_classes, y_pred_classes)

     print(f"Precision: {precision:.2f}")
     print(f"Recall: {recall:.2f}")
     print(f"F1 Score: {f1:.2f}")
```

```
[19] from sklearn.metrics import classification_report, confusion_matrix
     import seaborn as sns
```

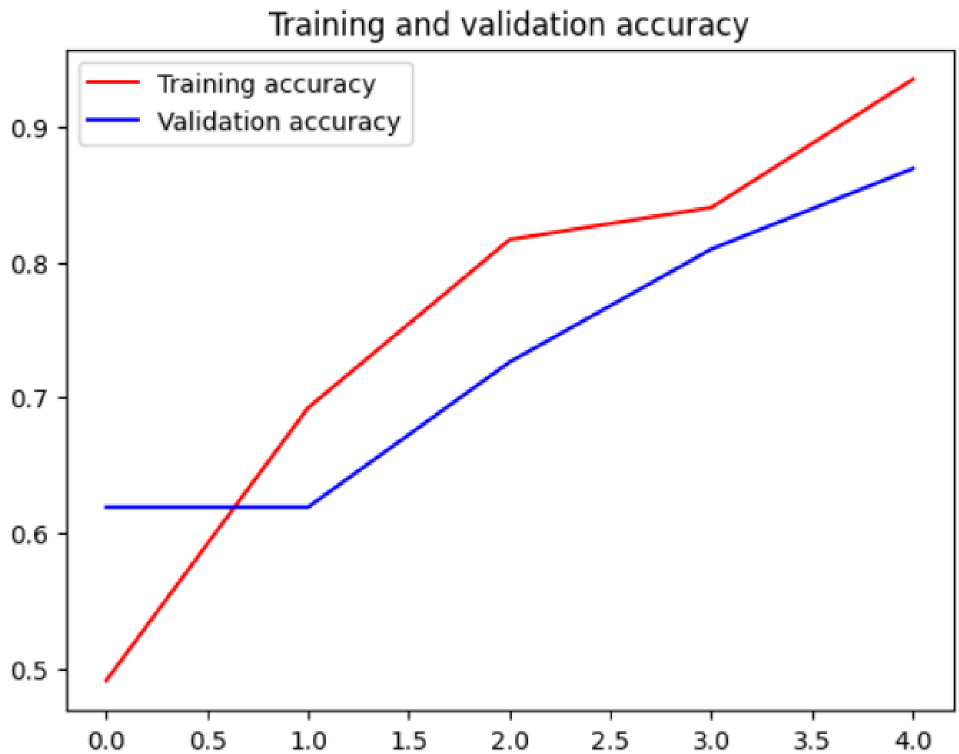
```
[20] # Predict classes on the test set
     y_pred = model.predict(X_test)
     y_pred_classes = np.argmax(y_pred, axis=1)
     y_true_classes = np.argmax(y_test, axis=1)
```

```
[21] print("Classification Report:")
     print(classification_report(y_true_classes, y_pred_classes, target_names=['No Tumor', 'Tumor']))
```

```
[22] conf_matrix = confusion_matrix(y_true_classes, y_pred_classes)
     plt.figure(figsize=(6, 5))
     sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=['No Tumor', 'Tumor'], yticklabels=['No Tumor', 'Tumor'])
     plt.xlabel('Predicted Labels')
     plt.ylabel('True Labels')
     plt.title("Confusion Matrix")
     plt.show()
```

```
[25] plt.figure(figsize=(12, 6))
     for i in range(9):
         plt.subplot(3, 3, i + 1)
         plt.imshow(X_test[i])
         plt.axis('off')
         true_label = 'Tumor' if y_true_classes[i] == 1 else 'No Tumor'
         pred_label = 'Tumor' if y_pred_classes[i] == 1 else 'No Tumor'
         plt.title(f"True: {true_label}\nPred: {pred_label}")
     plt.tight_layout()
     plt.show()
```

Output:



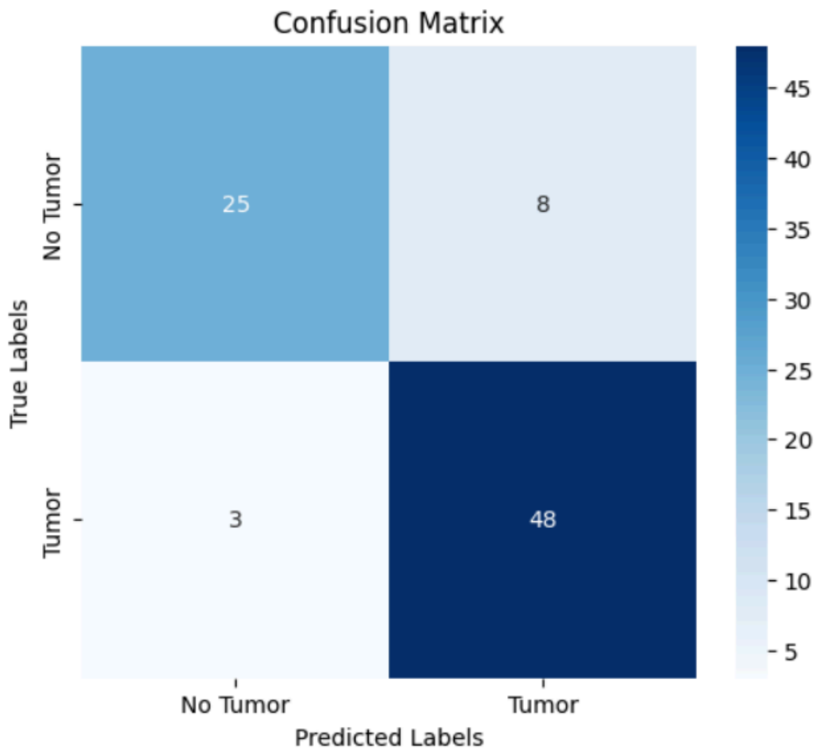

```
<Figure size 640x480 with 0 Axes>
```

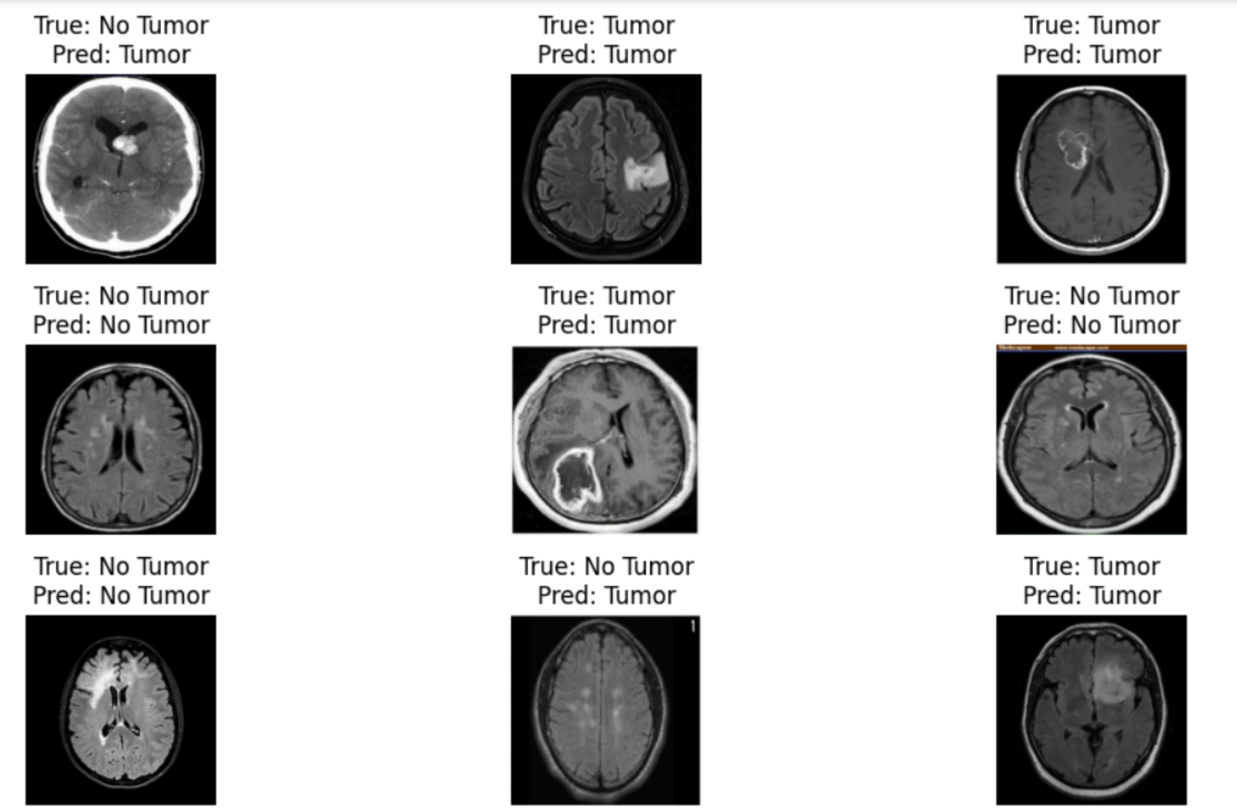
```
Precision: 0.86
Recall: 0.94
F1 Score: 0.90
```

```
Classification Report:
              precision    recall  f1-score   support

    No Tumor       0.89      0.76      0.82        33
       Tumor       0.86      0.94      0.90        51

    accuracy                           0.87        84
   macro avg       0.88      0.85      0.86        84
weighted avg       0.87      0.87      0.87        84
```


Confusion Matrix
No Tumor
Tumor
True Labels
25
8
3
48
No Tumor
Tumor
Predicted Labels
45
40
35
30
25
20
15
10
5


True: No Tumor
Pred: Tumor
True: Tumor
Pred: Tumor
True: Tumor
Pred: Tumor
True: No Tumor
Pred: No Tumor
True: Tumor
Pred: Tumor
True: No Tumor
Pred: No Tumor
True: No Tumor
Pred: No Tumor
True: No Tumor
Pred: Tumor
True: Tumor
Pred: Tumor

Conclusion:

The model effectively classifies brain MRI images into "Tumor" and "No Tumor" categories with an accuracy of 87%. It demonstrates high precision (0.86) and recall (0.94) for tumor detection, making it reliable for identifying cases with tumors, which is critical in medical diagnostics. The confusion matrix shows some misclassifications, particularly in identifying "No Tumor" cases, but overall, the model performs well, especially in minimizing the risk of missing actual tumors. This balance of accuracy and reliability makes it suitable for real-world medical applications.

Experiment-4

Aim:
Implement GAN to Generate Synthetic Data and Explore Its Application in Image Generation and Data Augmentation.

Theory:
The implementation of Generative Adversarial Networks (GANs) involves leveraging their ability to generate realistic synthetic data that mimics the structure of a given dataset. GANs consist of two neural networks, a Generator and a Discriminator, which are trained in an adversarial manner. The generator aims to create data that resembles the training data, while the discriminator attempts to distinguish between real and synthetic data. This adversarial setup pushes both networks to improve, resulting in high-quality synthetic data generation.

In the case of image datasets like MNIST, the goal is to generate synthetic images of handwritten digits (0-9) that are indistinguishable from real samples. This synthetic data has applications in:

Image Generation: Enhancing datasets with more examples of certain classes, improving model robustness.

Data Augmentation: Expanding the training dataset by adding variability, addressing class imbalance, and improving the performance of machine learning models.

Key Steps in Implementing GANs

1. **Data Preprocessing**

    - Load the MNIST dataset.
    - Normalize pixel values to the range [0, 1] for stable training.

2. **Define the GAN Architecture:**

    - **Generator:** A neural network that transforms random noise (latent vector) into synthetic images. The network uses techniques like transpose convolution and activation functions (e.g., ReLU, tanh) to generate realistic images.

    - **Discriminator:** A classifier neural network that distinguishes between real and synthetic images. It uses convolutional layers and outputs probabilities through a sigmoid activation function.

3. **Training Procedure:**

   - **Adversarial Training:** Train the generator to produce images that deceive the discriminator and train the discriminator to correctly classify images as real or fake.
   - **Loss Functions:**

     - **Generator loss:** Measures how effectively the generator can "fool" the discriminator.
     - **Discriminator loss:** Measures how well the discriminator can distinguish between real and synthetic images.

   - Use optimizers like Adam with a carefully tuned learning rate to ensure stability.

4. **Evaluate Generated Images:**

   - Visualize generated samples to assess the generator's performance.
   - Use metrics like the Inception Score (IS) or Fréchet Inception Distance (FID) to evaluate the quality and diversity of the synthetic data.

Applications of GAN-Generated Synthetic Data

1. **Image Generation:**

   - Creating new samples to enrich datasets or support artistic/creative projects.

2. **Data Augmentation:**

   - Balancing datasets by generating more data for underrepresented classes.
   - Supporting training models with additional synthetic data to improve generalization.

3. **Simulation and Research:**
   - Experimenting with "what-if" scenarios by generating data variations that don't exist in the real dataset.

Dataset Description

The MNIST dataset is a collection of grayscale images of handwritten digits from 0 to 9, commonly used in image processing and machine learning.

- **Image Size**: 28x28 pixels.
- **Channels**: 1 (grayscale).
- **Number of Classes**: 10 (digits 0-9).
- **Dataset Split:**
  - Training set: 60,000 images.
  - Test set: 10,000 images.

Code:

```python
import tensorflow as tf
from tensorflow.keras.layers import(Dense, BatchNormalization, LeakyReLU, Reshape, Conv2DTranspose, Conv2D, Dropout, Flatten)
import matplotlib.pyplot as plt
```

```python
(train_images, train_labels), (test_images, test_labels) = tf.keras.datasets.mnist.load_data()

train_images = train_images.reshape(train_images.shape[0], 28, 28, 1).astype('float32')
test_images = test_images.reshape(test_images.shape[0], 28, 28, 1).astype('float32')

train_images = (train_images - 127.5) / 127.5
test_images = (test_images - 127.5) / 127.5

train_labels = train_labels.astype('float32')
test_labels = test_labels.astype('float32')

Buffer_size = 60000
Batch_size = 256

train_dataset = tf.data.Dataset.from_tensor_slices(train_images).shuffle(Buffer_size).batch(Batch_size)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 2s 0us/step
```

```python
def generator():
    model = tf.keras.Sequential()
    model.add(Dense(7*7*256, use_bias=False, input_shape=(100,)))
    model.add(BatchNormalization())
    model.add(LeakyReLU())

    model.add(Reshape((7, 7, 256)))
    assert model.output_shape == (None, 7, 7, 256)

    model.add(Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same', use_bias=False))
    assert model.output_shape == (None, 7, 7, 128)
    model.add(BatchNormalization())
    model.add(LeakyReLU())

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

    model.add(Conv2DTranspose(1, (5, 5), strides=(2, 2), padding='same', use_bias=False, activation='tanh'))
    assert model.output_shape == (None, 28, 28, 1)

    return model
```

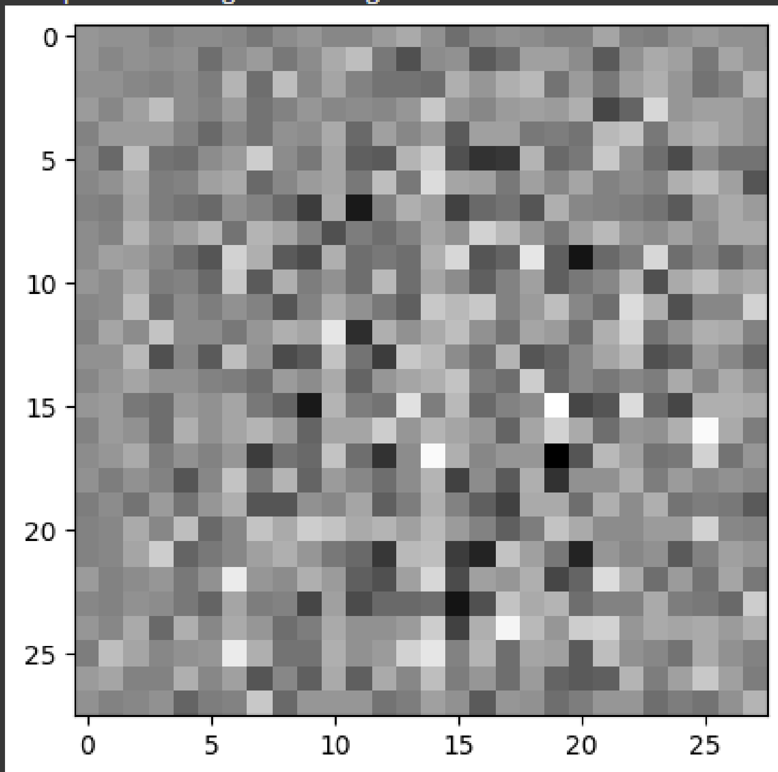
```python
generator = generator()
```

Run cell (Ctrl+Enter)
cell has not been executed in this session

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 12544)             1254400   
                                                                 
 batch_normalization (Batch  (None, 12544)             50176     
 Normalization)                                                  
                                                                 
 leaky_re_lu (LeakyReLU)     (None, 12544)             0         
                                                                 
 reshape (Reshape)           (None, 7, 7, 256)         0         
                                                                 
 conv2d_transpose (Conv2DTr  (None, 7, 7, 128)         819200    
 anspose)                                                        
                                                                 
 batch_normalization_1 (Bat  (None, 7, 7, 128)         512       
 chNormalization)                                                
                                                                 
 leaky_re_lu_1 (LeakyReLU)   (None, 7, 7, 128)         0         
                                                                 
 conv2d_transpose_1 (Conv2D  (None, 14, 14, 64)        204800    
 Transpose)                                                      
                                                                 
 batch_normalization_2 (Bat  (None, 14, 14, 64)        256       
 chNormalization)                                                
                                                                 
 leaky_re_lu_2 (LeakyReLU)   (None, 14, 14, 64)        0         
                                                                 
 conv2d_transpose_2 (Conv2D  (None, 28, 28, 1)         1600      
 Transpose)                                                      
                                                                 
=================================================================
Total params: 2330944 (8.89 MB)
Trainable params: 2305472 (8.79 MB)
Non-trainable params: 25472 (99.50 KB)
_________________________________________________________________
```

```
noise = tf.random.normal([1,100])
generated_image = generator(noise, training=False)
plt.imshow(generated_image[0, :, :, 0], cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7c82c05a1660>
```




```
def discriminator():
    model = tf.keras.Sequential()
    model.add(Conv2D(64, (5, 5), strides=(2, 2), padding='same', input_shape=[28, 28, 1]))
    model.add(LeakyReLU())
    model.add(Dropout(0.3))
    model.add(Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
    model.add(LeakyReLU())
    model.add(Dropout(0.3))
    model.add(Flatten())
    model.add(Dense(1))

    return model
```

```
[ ] discriminator = discriminator()
```

```
discriminator.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 14, 14, 64)        1664

 leaky_re_lu_3 (LeakyReLU)   (None, 14, 14, 64)        0

 dropout (Dropout)           (None, 14, 14, 64)        0

 conv2d_1 (Conv2D)           (None, 7, 7, 128)         204928

 leaky_re_lu_4 (LeakyReLU)   (None, 7, 7, 128)         0

 dropout_1 (Dropout)         (None, 7, 7, 128)         0

 flatten (Flatten)           (None, 6272)              0

 dense_1 (Dense)             (None, 1)                 6273

=================================================================
Total params: 212865 (831.50 KB)
Trainable params: 212865 (831.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
[ ] cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)

    def d_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

    def g_loss(fake_output):
        return cross_entropy(tf.ones_like(fake_output), fake_output)

    g_optimizer = tf.keras.optimizers.Adam(1e-4)
    d_optimizer = tf.keras.optimizers.Adam(1e-4)
```

```
import os

checkpoin_dir = "./training_checkpoints"
checkpoint_prefix = os.path.join(checkpoin_dir, "ckpt")
checkpoint = tf.train.Checkpoint(generator_optimizer = g_optimizer,
                                 discriminator_optimizer = d_optimizer,
                                 generator = generator,
                                 discriminator = discriminator)
```

```
[ ] EPOCHS = 40
    noise_dim = 100
    num_examples_to_generate = 16

    seed = tf.random.normal([num_examples_to_generate, noise_dim])
```

```python
@tf.function
def train_step(images):

    #creating random noise to feed model
    noise = tf.random.normal([Batch_size, noise_dim])

    # generate images and calculate loss values
    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 = g_loss(fake_output)
        disc_loss = d_loss(real_output, fake_output)

    # calculate gradients using loss and model variables
    gradients_of_generator = gen_tape.gradient(gen_loss, generator.trainable_variables)
    gradients_of_discriminator = disc_tape.gradient(disc_loss, discriminator.trainable_variables)

    # process gradients and run optimizer
    g_optimizer.apply_gradients(zip(gradients_of_generator, generator.trainable_variables))
    d_optimizer.apply_gradients(zip(gradients_of_discriminator, discriminator.trainable_variables))
```

```python
def generate_and_save_images(model, epoch, test_input):
    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()
```

```python
import time
from IPython import display

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

        for image_batch in dataset:
            train_step(image_batch)

        display.clear_output(wait=True)
        generate_and_save_images(generator, epoch + 1, seed)

        if (epoch + 1) % 5 == 0:
            checkpoint.save(file_prefix = checkpoint_prefix)

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


    display.clear_output(wait=True)
    generate_and_save_images(generator, epochs, seed)
```

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

```python
checkpoint.restore(tf.train.latest_checkpoint(checkpoin_dir))
```

```
<tensorflow.python.checkpoint.checkpoint.CheckpointLoadStatus at 0x7c82c04f6290>
```

```python
import PIL

def display_image(epoch_no):
    return PIL.Image.open('image_at_epoch_{:04d}.png'.format(epoch_no))
display_image(EPOCHS)
```

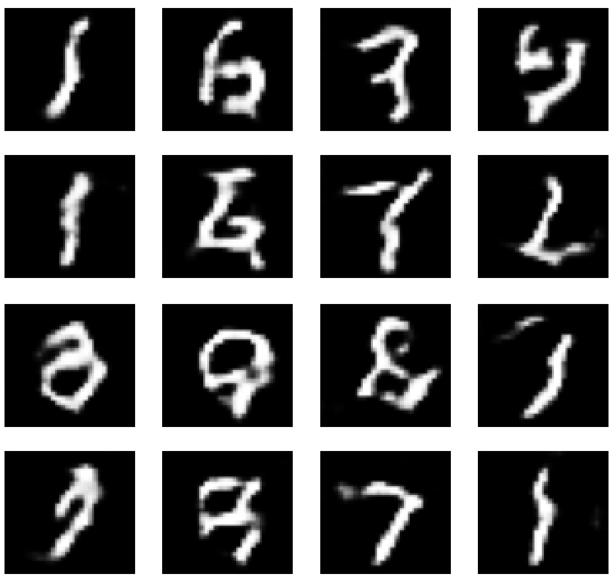
```
import glob
import imageio

anim_file = 'dcgan.gif'

with imageio.get_writer(anim_file, mode='I') as writer:
    filenames = glob.glob('image*.png')
    filenames = sorted(filenames)
    for filename in filenames:
        image = imageio.imread(filename)
        writer.append_data(image)

display.Image(open(anim_file, 'rb').read())
```

Output:



Conclusion:

In this practical, we implemented a GAN using the MNIST dataset to generate synthetic numeral images, successfully creating realistic digits that expanded our dataset. This experiment highlighted GANs' effectiveness for data augmentation, especially in enhancing model training with additional synthetic data.

Experiment-5

Aim:

Apply NLP techniques to process and analyze textual data, including sentiment analysis & named entity recognition.

Theory:

The analysis of audit comments involves several key steps to extract meaningful insights. First, data preprocessing is crucial for preparing the dataset. This step involves loading the dataset and cleaning the text by removing unnecessary characters, converting the text to lowercase, and eliminating common stop words that do not contribute meaningfully to sentiment or entity recognition. Following this, sentiment analysis is applied to classify each comment as positive, negative, or neutral, giving an initial understanding of the overall tone of the comments. This can be achieved using pre-trained models like those from Hugging Face or tools such as NLTK's VADER, which effectively label sentiments in financial and textual data. Next, Named Entity Recognition (NER) identifies and extracts organization names mentioned within the comments, allowing us to associate sentiments directly with specific entities. Using pre-trained NER models, we can tag and isolate these organizations. In the final analysis phase, the sentiment distribution for each organization is calculated, highlighting organizations with predominantly positive, negative, or neutral sentiments. These results are visualized using charts, providing a clear overview of sentiment trends across organizations. This approach not only aids in filtering entities based on sentiment but also offers valuable insights into patterns within the audit comments.

Sentiment Analysis focuses on determining the emotional tone of each audit comment. By using a pre-trained sentiment model, we classify each comment as positive, negative, or neutral. This allows us to quantify the sentiment across all comments and identify trends in the tone of feedback given by auditors. The sentiment scores help in pre-filtering organizations, highlighting those that might need further analysis due to high positive or negative feedback.

Named Entity Recognition (NER), on the other hand, is used to identify and extract organization names mentioned within the audit comments. By applying a pre-trained NER model, we can tag these entities, linking each comment to specific organizations. This enables us to calculate sentiment distributions for each organization, helping to pinpoint entities with consistently high positive or negative sentiments, thus assisting in trend identification and deeper analysis.

Dataset Description: Audit Comments Dataset -

This dataset, sourced from Hugging Face, contains textual comments from auditors with associated sentiment labels. The dataset is used to analyze the emotional tone of audit feedback toward various organizations. Below is a description of each field in the dataset:

- **Comment Text**: Contains the text of each audit comment, where auditors provide feedback or observations. This field is the primary text data used for both sentiment analysis and named entity recognition (NER) to identify organizations mentioned and assess sentiment trends.
- **Label**: A numerical sentiment label assigned to each comment, categorizing the sentiment as follows:
    - **2** – Positive: The comment expresses a positive sentiment toward the organization.
    - **1** – Neutral: The comment reflects a neutral tone without a strong positive or negative opinion.
    - **0** – Negative: The comment indicates a negative sentiment, often highlighting concerns or issues.

This dataset structure allows for effective sentiment analysis and NER to extract actionable insights on auditor feedback trends across organizations.

Code:

sentiment-and-ner-analysis-of-audit-comments.ipynb ●

sentiment-and-ner-analysis-of-audit-comments.ipynb > ...

+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ... base (Python 3.11.7)

```python
# Import the Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import HTML
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from IPython.display import display

# tensorflow tensors
import tensorflow as tf
```

[2] Python

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detecte
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

Load the dataset from Hugging Face

```python
from datasets import load_dataset

# import the auditor comment dataset from Hugging Face
dataset = load_dataset("FinanceInc/auditor_sentiment")

# label: a label corresponding to the class as a string:
# 2 - positive; 1 - neutral; 0 - negative'
```

[3] Python

```
Downloading:   0%|          | 0.00/800 [00:00<?, ?B/s]
```

sentiment-and-ner-analysis-of-audit-comments.ipynb ●

sentiment-and-ner-analysis-of-audit-comments.ipynb > ...

+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ... base (Python 3.11.7)

```
Downloading data:   0%|          | 0.00/327k [00:00<?, ?B/s]

Extracting data files:   0%|          | 0/2 [00:00<?, ?it/s]

Dataset parquet downloaded and prepared to /root/.cache/huggingface/datasets/parquet/demo-org--auditor_review-deda15e5340bd334/0.0.0/0b6d5799bb726b24ad7fc7be72

  0%|          | 0/2 [00:00<?, ?it/s]
```

```python
# Train dataset
dataset['train']
```

[4] Python

```
Dataset({
    features: ['sentence', 'label'],
    num_rows: 3877
})
```

```python
# Test dataset
dataset['test']
```

[5] Python

```
Dataset({
    features: ['sentence', 'label'],
    num_rows: 969
})
```

Distribution of Sentiment Labels

sentiment-and-ner-analysis-of-audit-comments.ipynb ●

sentiment-and-ner-analysis-of-audit-comments.ipynb > ...

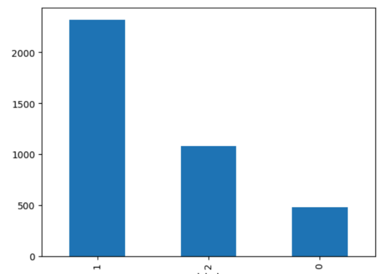
+ Code + Markdown | ▷ Run All ↺ Restart ☰ Clear All Outputs | Variables ☰ Outline ··· base (Python 3.11.7)

Distribution of Sentiment Labels

```
# Distribution of Target Label
label_list = list(dataset['train']['label'])
df = pd.DataFrame(label_list, columns=['label'])

# to display the bar charts by target values
rst = df['label'].value_counts().plot(kind='bar')
```

[6] Python



sentiment-and-ner-analysis-of-audit-comments.ipynb ●

sentiment-and-ner-analysis-of-audit-comments.ipynb > ...

+ Code + Markdown | ▷ Run All ↺ Restart ☰ Clear All Outputs | Variables ☰ Outline ··· base (Python 3.11.7)

```
# List out some positive comments
positive_dataset[0:5]
```

[8] Python

```
{'sentence': ["Altia 's operating profit jumped to EUR 47 million from EUR 6.6 million .",
  'The agreement was signed with Biohit Healthcare Ltd , the UK-based subsidiary of Biohit Oyj , a Finnish public company which develops , manufactures and marl
  'Kesko pursues a strategy of healthy , focused growth concentrating on sales and services to consumer-customers .',
  "Elcoteq 's stock of orders has stabilised in the past weeks , Mr Krippl said .",
  'UPM-Kymmene has generated seventeen consecutive quarters of positive Cash Flow from Operations .'],
 'label': [2, 2, 2, 2, 2]}
```

```
# List out some negative comments
negative_dataset[0:5]
```

[9] Python

```
{'sentence': ['Operating loss totalled EUR 0.9 mn , down from a profit of EUR 2.7 mn .',
  'Several large stocks tacked lower , however .',
  'As a result some 20 persons will no longer be needed .',
  'In addition the production personnel of the Sport Division have been given a temporary lay-off warning .',
  'Konecranes has previously communicated an estimated reduction of about 1,600 employees on group level in 2009 .'],
 'label': [0, 0, 0, 0, 0]}
```

sentiment-and-ner-analysis-of-audit-comments.ipynb ●

sentiment-and-ner-analysis-of-audit-comments.ipynb > ...

+ Code + Markdown | ▷ Run All ↺ Restart ☰ Clear All Outputs | Variables ☰ Outline ··· base (Python 3.11.7)

```
# Parameters
seq_len = 512
batch_size = 12

num_samples = len(dataset['train'])

# to prepare the array for target value
print('Training array of size ({} x {})'.format(num_samples, seq_len))
```

[10] Python

```
Training array of size (3877 x 512)
```

Load the Transformer Model

```
# Checkpoint for the Transfomer    "Transfomer": Unknown word.
check_point = 'bert-base-uncased'

# Import the Tokenizer of Bert
from transformers import BertTokenizer

# Load the pre-trained model
tokenizer = BertTokenizer.from_pretrained(check_point)
```

[11] Python

Spaces: 4 Cell 4 of 90 Go Live Quokka ✓ Spell ✓ Prettier

sentiment-and-ner-analysis-of-audit-comments.ipynb ●

sentiment-and-ner-analysis-of-audit-comments.ipynb > ...

+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ... base (Python 3.11.7)

```python
# tokenize the audit comments in the train dataset
tokens = tokenizer(dataset['train']['sentence'], max_length=seq_len, truncation=True,
                   padding='max_length', add_special_tokens=True,
                   return_tensors='pt')
```

[12] Python

```python
# Input ids
tokens['input_ids']
```

[13] Python

```
tensor([[  101, 12456,  2401,  ...,     0,     0,     0],
        [  101,  1996,  3820,  ...,     0,     0,     0],
        [  101, 17710, 21590,  ...,     0,     0,     0],
        ...,
        [  101,  1996,  3189,  ...,     0,     0,     0],
        [  101, 24797,  1011,  ...,     0,     0,     0],
        [  101,  1996,  2974,  ...,     0,     0,     0]])
```

```python
# Attention masks
tokens['attention_mask']
```

[14] Python

```
tensor([[1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0],
        ...,
        [1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0]])
```

sentiment-and-ner-analysis-of-audit-comments.ipynb ●

sentiment-and-ner-analysis-of-audit-comments.ipynb > ...

+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ... base (Python 3.11.7)

4. Preparation of Target Labels

```python
# first extract the list of sentiment values, i.e. 0 - 2
arr = pd.DataFrame(dataset['train']['label'], columns=['label'])
arr_values = arr['label'].sort_values().unique().tolist()
print('No. of distinct label values : {}'.format(arr_values))
```

[15] Python

```
No. of distinct label values : [0, 1, 2]
```

```python
# Initialize the zero array based on the size of the input_ids / attention masks
labels = np.zeros((num_samples, len(arr_values)))
print('Size of the target label : {}'.format(labels.shape))
```

[16] Python

```
Size of the target label : (3877, 3)
```

```python
# Assign the sentimen values    "sentimen": Unknown word.
labels[np.arange(num_samples), arr['label'].tolist()] = 1    "arange": Unknown word.
labels
```

[17] Python

```
array([[0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       ...,
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 1., 0.]])
```

sentiment-and-ner-analysis-of-audit-comments.ipynb ●

sentiment-and-ner-analysis-of-audit-comments.ipynb > ...

\+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ··· base (Python 3.11.7)

5. Sentiment Model Training

```python
# Create the tensorflow tenors from Input Ids, Attention Masks and Labels
ds = tf.data.Dataset.from_tensor_slices((tokens['input_ids'], tokens['attention_mask'], labels))
ds.take(1)
```

[18] Python

```
<_TakeDataset element_spec=(TensorSpec(shape=(512,), dtype=tf.int64, name=None), TensorSpec(shape=(512,), dtype=tf.int64, name=None), TensorSpec(shape=(3,), dty
```

```python
# Conver the tensors from 3d array of (3) tuples into 2d array of (2,1) tuples    "Conver": Unknown word.
def map_func(input_ids, masks, labels):

    return {'input_ids': input_ids, 'attention_mask': masks}, labels

# then we use the dataset map method to apply this transformation
ds = ds.map(map_func)
# print(len(dataset))

ds = ds.shuffle(10000).batch(batch_size, drop_remainder=True)
# print(len(dataset))

ds.take(1)
```

[19] Python

```
<_TakeDataset element_spec=({'input_ids': TensorSpec(shape=(12, 512), dtype=tf.int64, name=None), 'attention_mask': TensorSpec(shape=(12, 512), dtype=tf.int64,
```

```python
# Create the train and test datasets for Transformer model training
split = 0.8
```

[20]

0 b › span Spaces: 4 Cell 24 of 88 Go Live Quokka Spell

sentiment-and-ner-analysis-of-audit-comments.ipynb ●

sentiment-and-ner-analysis-of-audit-comments.ipynb > ...

\+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ··· base (Python 3.11.7)

Build and Train the Sentiment Model

```python
# Load the Transformer model
from transformers import TFAutoModel

# Set up the pre-tained model of Bert    "tained": Unknown word.
model = TFAutoModel.from_pretrained(check_point)
```

[24] Python

Could not render content for 'application/vnd.jupyter.widget-view+json'

{"model_id":"e96e5f909af94203ac1e9703b5fc5faf","version_major":2,"version_minor":0}

```
Some weights of the PyTorch model were not used when initializing the TF 2.0 model TFBertModel: ['cls.predictions.transform.dense.weight', 'cls.predictions.trar
- This IS expected if you are initializing TFBertModel from a PyTorch model trained on another task or with another architecture (e.g. initializing a TFBertFor:
- This IS NOT expected if you are initializing TFBertModel from a PyTorch model that you expect to be exactly identical (e.g. initializing a TFBertForSequenceC
All the weights of TFBertModel were initialized from the PyTorch model.
If your task is similar to the task the model of the checkpoint was trained on, you can already use TFBertModel for predictions without further training.
```

```python
# we can view the pre-tained Bert model using the summary method    "tained": Unknown word.
model.summary()
```

[25] Python

```
Model: "tf_bert_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 bert (TFBertMainLayer)      multiple                  109482240

=================================================================
Total params: 109,482,240
```

0 b › span Spaces: 4 Cell 32 of 87 Go Live Quokka 1 Spell Prettier

sentiment-and-ner-analysis-of-audit-comments.ipynb ●

sentiment-and-ner-analysis-of-audit-comments.ipynb > ...

+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ··· base (Python 3.11.7)

Model Architecture and Structure

```
# Two more input layes - input IDs and Attention Masks    "layes": Unknown word.
input_ids = tf.keras.layers.Input(shape=(512,), name='input_ids', dtype='int32')    "dtype": Unknown word.
mask = tf.keras.layers.Input(shape=(512,), name='attention_mask', dtype='int32')    "dtype": Unknown word.

# Embedding layer of Bert model
embeddings = model.bert(input_ids, attention_mask=mask)[1]  # access final activations (alread max-pooled) [1]    "alread": Unknown word.
# convert bert embeddings into 5 output classes
layer_1 = tf.keras.layers.Dense(1024, activation='relu')(embeddings)    "relu": Unknown word.
# Dropout layer with a rate of 0.5
# x = tf.keras.layers.Dropout(0.1)(layer_1)
layer_2 = tf.keras.layers.Dense(3, activation='softmax', name='outputs')(layer_1) # the sentiment has 3 labels    "softmax": Unknown word.
```

[26] Python

```
# initialize model with input layers of Input IDs and Attention Masks and output layer of Outputs
sentiment_model = tf.keras.Model(inputs=[input_ids, mask], outputs=layer_2)

# (optional) freeze bert layer - to decide if the embedding layer is re-trained.
sentiment_model.layers[2].trainable = True

# print the updated model summary
sentiment_model.summary()
```

[27] Python

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_ids (InputLayer)         [(None, 512)]        0           []
```

0 | b › span Spaces: 4 Cell 32 of 87 Go Live Quokka ✓ Spell

sentiment-and-ner-analysis-of-audit-comments.ipynb ●

sentiment-and-ner-analysis-of-audit-comments.ipynb > ...

+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ··· base (Python 3.11.7)

Model Fine-tuning with Additional Layers

```
# Training parameters
# optimizer = tf.keras.optimizers.legacy.Adam(learning_rate=1e-5, decay=1e-6)
optimizer = tf.keras.optimizers.legacy.Adam(learning_rate=1e-5, decay=1e-6)
loss = tf.keras.losses.CategoricalCrossentropy()    "Crossentropy": Unknown word.
acc = tf.keras.metrics.CategoricalAccuracy('accuracy')

sentiment_model.compile(optimizer=optimizer, loss=loss, metrics=[acc])
```

[28] Python

```
# Re-train the model
history = sentiment_model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=3
)
```

[29] Python

```
Epoch 1/3
258/258 [==============================] - 237s 879ms/step - loss: 0.6296 - accuracy: 0.7251 - val_loss: 0.3359 - val_accuracy: 0.8615
Epoch 2/3
258/258 [==============================] - 225s 872ms/step - loss: 0.3332 - accuracy: 0.8640 - val_loss: 0.2080 - val_accuracy: 0.9244
Epoch 3/3
258/258 [==============================] - 225s 872ms/step - loss: 0.2268 - accuracy: 0.9170 - val_loss: 0.1769 - val_accuracy: 0.9449
```

```
# free up memory
del train_ds
del val_ds
```

[30]

0 | b › span Spaces: 4 Cell 32 of 87 Go Live Quokka ✓ Spell

6. Sentiment Analysis

```python
# The test data of one instance only
def prep_data(text):
    tokens = tokenizer.encode_plus(text, max_length=512,
                                   truncation=True, padding='max_length',
                                   add_special_tokens=True, return_token_type_ids=False,
                                   return_tensors='tf')
    # tokenizer returns int32 tensors, we need to return float64, so we use tf.cast
    return {'input_ids': tf.cast(tokens['input_ids'], tf.float64),
            'attention_mask': tf.cast(tokens['attention_mask'], tf.float64)}
```

[31] Python

```python
# The test data in array
def prep_arry_data(text_array):    "arry": Unknown word.
    tokens = tokenizer.batch_encode_plus(text_array, max_length=512,
                                         truncation=True, padding='max_length',
                                         add_special_tokens=True, return_token_type_ids=False,
                                         return_tensors='tf')
    # tokenizer returns int32 tensors, we need to return float64, so we use tf.cast
    return {'input_ids': tf.cast(tokens['input_ids'], tf.float64),
            'attention_mask': tf.cast(tokens['attention_mask'], tf.float64)}
```

[32] Python

```python
# Create a dictionary to map label indices to label names
label_mapping = {
    0: 'Negative',
    1: 'Neutral',
    2: 'Positive'
```

[33]

sentiment-and-ner-analysis-of-audit-comments.ipynb ●
sentiment-and-ner-analysis-of-audit-comments.ipynb > M↓ 6. Sentiment Analysis > # The test data of one instance only
+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ··· base (Python 3.11.7)

Single Test Case

```python
text_array = [
"XXX's strong financial performance exceeded expectations, driven by robust sales growth and effective cost management.",
"XXX demonstrated resilience in challenging economic conditions, maintaining profitability through efficient expense management.",
"XXX's net income declined due to increased operating costs and lower sales volumes. Cost-cutting measures are being implemented.",
"XXX maintains a strong liquidity position with healthy cash reserves, providing a solid foundation for future investments.",
"XXX's profitability has steadily increased, attributed to successful product launches and expanded market reach.",
"XXX's debt management efforts resulted in decreased debt levels and improved debt-to-equity ratio."

]

probs = sentiment_model.predict(prep_arry_data(text_array))    "probs": Unknown word.

# import numpy as np
pred = np.argmax(probs, axis=1)    "argmax": Unknown word.

# Sentiment output
for count, prediction in enumerate(pred):
    print('The sentiment for test case {} is {}.'.format(count, label_mapping.get(prediction)))
```

[34] Python

```
1/1 [==============================] - 3s 3s/step
The sentiment for test case 0 is Positive.
The sentiment for test case 1 is Positive.
The sentiment for test case 2 is Negative.
The sentiment for test case 3 is Positive.
The sentiment for test case 4 is Positive.
The sentiment for test case 5 is Positive.
```

sentiment-and-ner-analysis-of-audit-comments.ipynb ●

sentiment-and-ner-analysis-of-audit-comments.ipynb > M↓ 6. Sentiment Analysis > # The test data of one instance only

\+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ··· base (Python 3.11.7)

Dataset of Sentences

```python
# Sentiment prediction
probs = sentiment_model.predict(prep_arry_data(dataset['test']['sentence']))    "probs": Unknown word.

# import numpy as np
pred = np.argmax(probs, axis=1)    "argmax": Unknown word.
```

[35] Python

```
31/31 [==============================] - 21s 670ms/step
```

```python
df_test = pd.DataFrame(dataset['test']['label'], columns = ['label'])
df_pred = pd.DataFrame(pred, columns = ['label'])
```

[36] Python

\+ Code + Markdown

```python
# Calculate accuracy
accuracy = accuracy_score(df_pred, df_test) * 100

# Print the accuracy
print(f"Prediction Accuracy: {accuracy:.2f}%")
```

[37] Python

```
Prediction Accuracy: 85.55%
```

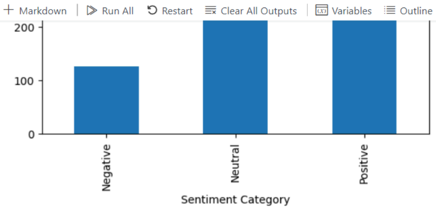
```python
# Sentiment output
```

0 b › span Spaces: 4 Cell 42 of 86 Go Live Quokka Spell

sentiment-and-ner-analysis-of-audit-comments.ipynb ●

sentiment-and-ner-analysis-of-audit-comments.ipynb > M↓ 6. Sentiment Analysis > # The test data of one instance only

\+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ··· base (Python 3.11.7)




```python
# Sentiment for positive comments
positive_pred = df_pred[df_pred['label'] == 2][0:5]
# positive_pred

# Sample output
for idx in range(5):
    print('The sentiment is \033[1mPositive\033[0m for sentence of "\033[4m{}\033[0m".'.format(dataset['test']['sentence'][positive_pred.index[idx]]))
    print('')
```

[40] Python

The sentiment is **Positive** for sentence of "TeliaSonera TLSN said the offer is in line with its strategy to increase its ownership in core business holdings and

The sentiment is **Positive** for sentence of "STORA ENSO , NORSKE SKOG , M-REAL , UPM-KYMMENE Credit Suisse First Boston (CFSB) raised the fair value for shares

The sentiment is **Positive** for sentence of "Clothing retail chain Sepp+ñl+ñ 's sales increased by 8 % to EUR 155.2 mn , and operating profit rose to EUR 31.1 mn

The sentiment is **Positive** for sentence of "Lifetree was founded in 2000 , and its revenues have risen on an average by 40 % with margins in late 30s .".

The sentiment is **Positive** for sentence of "Nordea Group 's operating profit increased in 2010 by 18 percent year-on-year to 3.64 billion euros and total revenue

0 b › span Spaces: 4 Cell 42 of 86 Go Live Quokka Spell

sentiment-and-ner-analysis-of-audit-comments.ipynb

sentiment-and-ner-analysis-of-audit-comments.ipynb > M↓ 6. Sentiment Analysis > # The test data of one instance only

+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ··· base (Python 3.11.7)

```python
# Sample output
for idx in range(5):
    print('The sentiment is \033[1mNegative\033[0m for sentence of "\033[4m{}\033[0m".'.format(dataset['test']['sentence'][negative_pred.index[idx]]))
    print('')
```

[41] Python

```
The sentiment is Negative for sentence of "Compared with the FTSE 100 index , which rose 51.5 points ( or 0.9 % ) on the day , this was a relative price change

The sentiment is Negative for sentence of "Calls to the switchboard and directory services have decreased significantly since our employees now have up-to-date

The sentiment is Negative for sentence of "One of the challenges in the oil production in the North Sea is scale formation that can plug pipelines and halt proc

The sentiment is Negative for sentence of "Profit before taxes amounted to EUR 56.5 mn , down from EUR 232.9 mn a year ago .".

The sentiment is Negative for sentence of "Vaisala also said it expects net sales of EUR 253.2 million for 2010 , compared with EUR 252.2 million recorded in 20
```

```python
# Sentiment for neutral comments
neutral_pred = df_pred[df_pred['label'] == 1][0:5]
# negative_pred

# Sample output
for idx in range(5):
    print('The sentiment is \033[1mNeutral\033[0m for sentence of "\033[4m{}\033[0m".'.format(dataset['test']['sentence'][negative_pred.index[idx]]))
    print('')
```

[42] Python

```
The sentiment is Neutral for sentence of "Compared with the FTSE 100 index , which rose 51.5 points ( or 0.9 % ) on the day , this was a relative price change c

The sentiment is Neutral for sentence of "Calls to the switchboard and directory services have decreased significantly since our employees now have up-to-date c

The sentiment is Neutral for sentence of "One of the challenges in the oil production in the North Sea is scale formation that can plug pipelines and halt produ

The sentiment is Neutral for sentence of "Profit before taxes amounted to EUR 56.5 mn , down from EUR 232.9 mn a year ago .".
```

0 b › span Spaces: 4 Cell 42 of 86 Go Live Quokka Spell {}

sentiment-and-ner-analysis-of-audit-comments.ipynb

sentiment-and-ner-analysis-of-audit-comments.ipynb > M↓ 7. Named Entity Recognition (NER)

+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ··· base (Python 3.11.7)

```
The sentiment is Neutral for sentence of "Compared with the FTSE 100 index , which rose 51.5 points ( or 0.9 % ) on the day , this was a relative price change c

The sentiment is Neutral for sentence of "Calls to the switchboard and directory services have decreased significantly since our employees now have up-to-date c

The sentiment is Neutral for sentence of "One of the challenges in the oil production in the North Sea is scale formation that can plug pipelines and halt produ

The sentiment is Neutral for sentence of "Profit before taxes amounted to EUR 56.5 mn , down from EUR 232.9 mn a year ago .".

The sentiment is Neutral for sentence of "Vaisala also said it expects net sales of EUR 253.2 million for 2010 , compared with EUR 252.2 million recorded in 20
```

7. Named Entity Recognition (NER)

```python
# Parameter
ner_name = 'test'
```

[43] Python

```python
# Load the model
from transformers import AutoTokenizer, AutoModelForTokenClassification
from transformers import pipeline

checkpoint_2 = "dslim/bert-base-NER"    "dslim": Unknown word.

# Load the pre-trained Bert model for NER
ner_tokenizer = AutoTokenizer.from_pretrained(checkpoint_2)
ner_model = AutoModelForTokenClassification.from_pretrained(checkpoint_2)
```

[44]

0 b › span Spaces: 4 Cell 56 of 85 Go Live Quokka Spell

8. Formation of Organization Entity

```
# The Bert model returns B-ORG and I-ORG, the B-ORG is the beginning of an orgnaization entity identified by the model    "orgnaization": Unknown word.
# The I-ORG is a linked token to the precedent B-ORG or I-ORG token.
# The concantation of consencutive B-ORG and subsequent I-ORG will form the name of the organization entity    "concantation": Unknown word.

entity_dict = {}
entities = []
current_entity = ""
for idx in range(len(ner_results)):
    for token in ner_results[idx]:
        if token['entity'] == "B-ORG":
            if token['word'][0:2] == "##":
                current_entity += token['word'][2:]
            elif current_entity != '':
                entities.append(current_entity)
                current_entity = token['word']
            else:
                current_entity = token['word']
        elif token['entity'] == "I-ORG":
            if token['word'][0:2] == "##":
                current_entity += token['word'][2:]
            else:
                current_entity += " " + token['word']

    entities.append(current_entity) # Append the last token to the entity list

    entity_dict[idx] = entities

    # empty the array and current entity
    entities = []
```

[47]

sentiment-and-ner-analysis-of-audit-comments.ipynb ●
sentiment-and-ner-analysis-of-audit-comments.ipynb > M↓ 8. Formation of Organization Entity
+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ··· base (Python 3.11.7)

```
No. of records in the NER results : 969
```

```
# After NER, null return will be removed, and we only need the records with return of organization names

# All records and organization names after the NER
entity_2d_array = [[k, v] for k, values in entity_dict.items() for v in values]

# Remove those records with null return
entity_2d_array_dedup = [sublist for sublist in entity_2d_array if all(item != '' for item in sublist)]    "dedup": Unknown word.

print('No. of organizations in the original NER: {}'.format(len(entity_2d_array)))
print('No. of organizations after the de-duplication: {}'.format(len(entity_2d_array_dedup)))    "dedup": Unknown word.
```

[48] Python

```
No. of organizations in the original NER: 1366
No. of organizations after the de-duplication: 935
```

```
# List some of the NER results
entity_2d_array_dedup[:10]    "dedup": Unknown word.
```

[49] Python

```
[[0, 'TeliaSonera TLSN'],
 [0, 'Eesti Telekom'],
 [1, 'STORA ENSO'],
 [1, 'NORSKE SKOG'],
 [1, 'M R'],
 [1, 'KYMMENE Credit Suisse First Boston'],
 [1, 'CFSB'],
 [3, 'Lifetree'],
 [4, 'Nordea Group'],
 [6, "Lithuanian Brewers ' Association"]]
```

0 b › span Spaces: 4 Cell 61 of 84 Go Live Quokka ✓ Spell

Output -

sentiment-and-ner-analysis-of-audit-comments.ipynb ●

sentiment-and-ner-analysis-of-audit-comments.ipynb > M↓ 9. Analysis of Audit Comments

+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ··· base (Python 3.11.7)

Sentiment and NER Results

```python
df_ner_sentiment = pd.DataFrame(merged_array, columns=['doc_id', 'org_name', 'sentiment_label'])
df_ner_sentiment
```

[57] Python

	doc_id	org_name	sentiment_label
0	0	TeliaSonera TLSN	2
1	0	Eesti Telekom	2
2	1	STORA ENSO	2
3	1	NORSKE SKOG	2
4	1	M R	2
...	...	...	...
930	955	Wireless Solutions	2
931	957	Danske Bank A	2
932	957	DANKE DC	2
933	964	Ragutis	0
934	964	Olvi	0

935 rows × 3 columns

sentiment-and-ner-analysis-of-audit-comments.ipynb ●

sentiment-and-ner-analysis-of-audit-comments.ipynb > M↓ 9. Analysis of Audit Comments

+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ··· base (Python 3.11.7)

```python
filter_by_sentiment (2, 'Positive')
```

[59] Python

```
========================================================================
Top 10 Companies with Positive Audit Comments based on Sentiment Analysis
========================================================================

org_name
ADP News        4
E               4
Elcoteq         3
Kesko           3
Nokia           3
Aspo            3
M - real        3
KCI Konecranes  3
Talvivaara      3
Outotec         3
Name: doc_id, dtype: int64
```

```python
filter_by_sentiment (0, 'Negative')
```

[60] Python

```
========================================================================
Top 10 Companies with Negative Audit Comments based on Sentiment Analysis
========================================================================

org_name
Talentum             4
Pretax               3
Scanfil              3
L & T                2
```

0 b › span Spaces: 4 Cell 76 of 84 Go Live Quokka Spell

sentiment-and-ner-analysis-of-audit-comments.ipynb ●

sentiment-and-ner-analysis-of-audit-comments.ipynb > M↓ 9. Analysis of Audit Comments

+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ···

```
org_name
Talentum               4
Pretax                 3
Scanfil                3
L & T                  2
EB                     2
Cencorp Corporation    2
OMX Helsinki           2
Nordic Aluminium       1
Nobel Biocare          1
Ramirent               1
Name: doc_id, dtype: int64
```

[61]

```
filter_by_sentiment (1, 'Neutral')
```

```
========================================================================
Top 10 Companies with Neutral Audit Comments based on Sentiment Analysis
========================================================================

org_name
Nokia          10
OMX Helsinki    7
Glaston         6
Nordea          5
Alma Media      5
Vacon           4
Teleste         4
Newstex         3
Aspo            3
Technopolis     3
Name: doc_id, dtype: int64
```

Conclusion:

In conclusion, this project successfully applied sentiment analysis and named entity recognition (NER) to auditor comments, enabling an automated assessment of sentiment trends across various organizations. By identifying positive, negative, and neutral sentiments and linking them to specific entities, we gained valuable insights into how organizations are perceived. This approach streamlines the review process, allowing for quicker identification of organizations with significant positive or negative feedback for deeper analysis.