List of Experiments

Assignment 1: Soil Type Classification Using PSA Data

Question

Develop and train a deep neural network to classify soil types (Hard, Transitional, or Soft) based on seismic PSA data. Perform data exploration, preprocessing, feature engineering, and apply dimensionality reduction if necessary. Split the dataset into training, validation, and test sets with stratification. Normalize the features and encode the labels appropriately. Implement a deep learning model using TensorFlow/Keras, train it, and evaluate its performance using accuracy, precision, recall, and F1-score. Discuss the model's limitations, challenges, and potential improvements.

प्रश्न:

डीप न्यूरल नेटवर्क विकसित करें और प्रशिक्षित करें ताकि भूकंपीय PSA डेटा के आधार पर मृदा प्रकारों (हार्ड, ट्रांज़िशनल, या सॉफ़्ट) को वर्गीकृत किया जा सके। डेटा एक्सप्लोरेशन करें, प्रीप्रोसेसिंग करें, फीचर इंजीनियरिंग लागू करें, और यदि आवश्यक हो तो डायमेंशनलिटी रिडक्शन लागू करें। डेटा को ट्रेनिंग, वेलिडेशन, और टेस्ट सेटों में समरूप रूप से विभाजित करें। फीचर्स को नॉर्मलाइज़ करें और लेबल्स को उपयुक्त रूप से एन्कोड करें। टेंसफ़्लो/केरस का उपयोग करके एक गहरा शिक्षण मॉडल (डीप लर्निंग मॉडल) लागू करें, इसे प्रशिक्षित करें, और एक्युरेसी, प्रीसिशन, रिकॉल, और F1-स्कोर जैसी मेट्रिक्स के माध्यम से इसके परफॉर्मेंस का मूल्यांकन करें। मॉडल की सीमाएँ, चुनौतियाँ और संभावित सुधारों पर चर्चा करें।

Code:-

```
# Import necessary libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
classification_report, confusion_matrix

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers


# Load dataset

df = pd.read_csv("soil_psa_data.csv")

print(df.head())


# Exploratory Data Analysis
```

```
print(df.describe())
print(df.isnull().sum())

# Visualize class distribution
df['Soil_Type'].value_counts().plot(kind='bar', color=['blue', 'orange', 'green'])
plt.xlabel("Soil Type")
plt.ylabel("Count")
plt.title("Soil Type Distribution")
plt.show()

# Feature Engineering: Extract Spectral Ratio statistics
feature_cols = ["SR_Mean", "SR_Max", "SR_Min", "SR_Median", "SR_Std"]
df_features = df[feature_cols]

# Apply PCA for dimensionality reduction
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df_features)
pca = PCA(n_components=5)
X_pca = pca.fit_transform(X_scaled)

# Data Splitting
X = X_pca
y = df['Soil_Type']
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, stratify=y, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, stratify=y_temp,
random_state=42)

# Normalize features
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)

# Encode labels
encoder = LabelEncoder()
```

```
y_train = encoder.fit_transform(y_train)
y_val = encoder.transform(y_val)
y_test = encoder.transform(y_test)

# Build Deep Learning Model
model = keras.Sequential([
  layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
  layers.Dense(32, activation='relu'),
  layers.Dense(3, activation='softmax')  # 3 output classes: Hard, Transitional, Soft
])

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

# Train model
history = model.fit(X_train, y_train, epochs=50, validation_data=(X_val, y_val), batch_size=16)

# Evaluate model performance
y_pred = model.predict(X_test)
y_pred_classes = y_pred.argmax(axis=1)

# Compute performance metrics
accuracy = accuracy_score(y_test, y_pred_classes)
precision = precision_score(y_test, y_pred_classes, average='weighted')
recall = recall_score(y_test, y_pred_classes, average='weighted')
f1 = f1_score(y_test, y_pred_classes, average='weighted')

# Print classification metrics
print("Model Performance Metrics:")
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-score: {f1:.4f}")
print("\nClassification Report:\n", classification_report(y_test, y_pred_classes))
```

```
# Confusion matrix

conf_matrix = confusion_matrix(y_test, y_pred_classes)

plt.figure(figsize=(6, 4))

sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=['Hard', 'Transitional', 'Soft'], yticklabels=['Hard', 'Transitional', 'Soft'])

plt.xlabel("Predicted Label")

plt.ylabel("True Label")

plt.title("Confusion Matrix")

plt.show()
```

Model Limitations, Challenges, and Potential Improvements:

- The model achieved an accuracy of {:.2f}%, which may be lower/higher than traditional ML models like XGBoost.".format(accuracy * 100)
- Class imbalance may affect the performance, particularly in underrepresented categories like Soft or Transitional soil.
- PCA reduced dimensions, but using all Spectral Ratio features might improve performance.
- Additional regularization (dropout, L2) can prevent overfitting."
- Data augmentation or synthetic data generation can help mitigate class imbalance issues.
- Hyperparameter tuning (learning rate, number of layers, batch size) might yield further improvements.
- Future work: testing convolutional architectures (CNNs) or hybrid CNN-LSTM for feature extraction.

Assignment 2: Diabetes Prediction with Pima Indians Diabetes Dataset

Question

Develop and train a deep neural network to predict the onset of diabetes using the Pima Indians Diabetes dataset. Perform data exploration and preprocessing, including handling missing values, anomalies, and class imbalance. Apply feature engineering and prepare the dataset for training. Build a deep learning model using TensorFlow/Keras and evaluate its performance using Accuracy, Precision, Recall, F1-score, and ROC-AUC. Generate a Confusion Matrix and ROC Curve, and plot training curves. Finally, analyze model performance, discuss the impact of class imbalance handling, and suggest potential improvements.

प्रश्न:

एक डीप न्यूरल नेटवर्क विकसित करें और प्रशिक्षित करें ताकि पीमा इंडियंस मधुमेह डेटासेट के आधार पर मधुमेह की भविष्यवाणी की जा सके। डेटा एक्सप्लोरेशन और प्रीप्रोसेसिंग करें, जिसमें मिसिंग वैल्यूज़, विसंगतियों (anomalies), और वर्ग असंतुलन को संभालना शामिल हो। फीचर इंजीनियरिंग लागू करें और डेटा को मॉडल प्रशिक्षण के लिए तैयार करें। टेंसफ़्लो/केरस का उपयोग करके एक डीप लर्निंग मॉडल बनाएं एक्युरेसी, प्रीसिशन, रिकॉल, और F1-स्कोर, और आरओसी-एयूसी जैसी मैट्रिक्स का उपयोग करके मॉडल के प्रदर्शन का मूल्यांकन करें। कन्फ्यूजन मैट्रिक्स और आरओसी कर्व बनाएं और ट्रेनिंग वेलिडेशन कर्व्स को प्लॉट करें। अंत में, मॉडल के प्रदर्शन का विश्लेषण करें, class imbalance को संभालने के प्रभाव पर चर्चा करें, और संभावित सुधारों का सुझाव दें।

Code :–

```
# Import necessary libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, classification_report, confusion_matrix, roc_curve

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers


# Load dataset

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"

column_names = ["Pregnancies", "Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI", "DiabetesPedigreeFunction", "Age", "Outcome"]

df = pd.read_csv(url, names=column_names)


# Exploratory Data Analysis
```

```
print(df.head())
print(df.describe())
print(df.isnull().sum())

# Handling missing values (replace zeros with NaN for specific columns)
zero_cols = ["Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI"]
df[zero_cols] = df[zero_cols].replace(0, np.nan)

# Impute missing values with median
df.fillna(df.median(), inplace=True)

# Visualize class distribution
sns.countplot(x='Outcome', data=df, palette=['blue', 'orange'])
plt.title("Diabetes Class Distribution")
plt.xlabel("Diabetes (0 = No, 1 = Yes)")
plt.ylabel("Count")
plt.show()

# Data Splitting
X = df.drop(columns=["Outcome"])
y = df["Outcome"]

# Split dataset into training (70%), validation (15%), and test (15%) sets
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, stratify=y, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, stratify=y_temp,
random_state=42)

# Normalize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)

# Handle class imbalance using class weights
class_weights = {0: 1.0, 1: sum(y_train == 0) / sum(y_train == 1)}
```

```
# Build Deep Learning Model
model = keras.Sequential([
  layers.Dense(16, activation='relu', input_shape=(X_train.shape[1],)),
  layers.Dense(8, activation='relu'),
  layers.Dense(1, activation='sigmoid')  # Binary classification (Diabetes or not)
])

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

# Train model
history = model.fit(X_train, y_train, epochs=50, validation_data=(X_val, y_val), batch_size=16,
class_weight=class_weights)

# Evaluate model performance
y_pred = (model.predict(X_test) > 0.5).astype("int32")

# Compute performance metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

# Print classification metrics
print("Model Performance Metrics:")
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-score: {f1:.4f}")
print(f"ROC-AUC: {roc_auc:.4f}")
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Confusion matrix
```

```
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=['No Diabetes', 'Diabetes'], yticklabels=['No Diabetes', 'Diabetes'])
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()

# Plot ROC Curve
fpr, tpr, _ = roc_curve(y_test, model.predict(X_test))
plt.plot(fpr, tpr, color='blue', label="ROC Curve (AUC = {:.4f})".format(roc_auc))
plt.plot([0, 1], [0, 1], linestyle="--", color="red")  # Diagonal line
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()

# Training Loss and Accuracy Curves
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training & Validation Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training & Validation Accuracy')
```

```
plt.legend()

plt.show()
```

Discussion on Model Performance

Model Performance Summary:

- The model achieved an accuracy of {:.2f}%, indicating its predictive power.".format(accuracy * 100)
- Balanced precision ({:.2f}) and recall ({:.2f}) suggest the model is neither biased towards false positives nor false negatives.".format(precision, recall)
- The ROC-AUC score of {:.2f} suggests good discrimination ability.".format(roc_auc)

Handling Class Imbalance Impact:

- Class weighting was applied to address the imbalance issue, which improved recall.
- Without class weights, the model might have been biased towards the majority class (non-diabetic cases)

Future improvements could include oversampling (SMOTE) or experimenting with different architectures like CNNs or transformer-based models.

Assignment 3: Music Generation Using the MAESTRO Dataset

Question:

Develop and train a deep neural network to generate music sequences using the MAESTRO Dataset. The model should be capable of composing piano music that mimics the style of the training data. Load and process the MIDI files, analyze their structure, and convert them into piano roll or token-based sequences. Split the dataset into Training, Validation, and Test sets, apply scaling and normalization, and optionally use data augmentation techniques. Design a deep learning model using LSTM, GRU, or Transformer-based architectures, train and evaluate it, and analyze Loss, Accuracy, and the quality of generated music. Finally, discuss model performance, music generation quality, challenges, and potential improvements.

प्रश्न:

एक डीप न्यूरल नेटवर्क विकसित करें और प्रशिक्षित करें ताकि MAESTRO Dataset का उपयोग करके संगीत अनुक्रमों (music sequences) का निर्माण किया जा सके। मॉडल को पियानो संगीत उत्पन्न करने में सक्षम होना चाहिए जो प्रशिक्षण डेटा की शैली की नकल कर सके। डेटा को लोड और संसाधित करें, MIDI फ़ाइलों को विश्लेषण करें, और पियानो रोल या टोकन-आधारित अनुक्रमों में परिवर्तित करें। डेटासेट को प्रशिक्षण, सत्यापन और परीक्षण सेटों में विभाजित करें, स्केलिंग और नॉर्मलाइज़ेशन लागू करें, और डेटा वृद्धि (Data Augmentation) का उपयोग करें। एलएसटीएम, जीआरयू, या ट्रांसफॉर्मर आर्किटेक्चर पर आधारित एक डीप लर्निंग मॉडल डिज़ाइन करें, इसे ट्रेन करें और मूल्यांकन करें, और लॉस, एक्युरेसी और संगीत उत्पादन की गुणवत्ता का विश्लेषण करें। अंत में, मॉडल प्रदर्शन, संगीत निर्माण की गुणवत्ता, और संभावित सुधारों पर चर्चा करें।

Code:-

```
# Import necessary libraries

import numpy as np

import pandas as pd

import os

import matplotlib.pyplot as plt

import seaborn as sns

import pretty_midi

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

from sklearn.model_selection import train_test_split


# Load MAESTRO Dataset (Example Path)

dataset_path = "/path/to/maestro-v3.0.0/"

midi_files = [os.path.join(dataset_path, f) for f in os.listdir(dataset_path) if f.endswith('.midi') or
f.endswith('.mid')]


# Display number of MIDI files available

print(f"Total MIDI files: {len(midi_files)}")
```

```
# Function to extract MIDI note sequences
def midi_to_notes(midi_file):
    pm = pretty_midi.PrettyMIDI(midi_file)
    notes = []
    for instrument in pm.instruments:
        for note in instrument.notes:
            notes.append([note.start, note.end, note.pitch, note.velocity])
    return np.array(notes)

# Process all MIDI files
all_notes = [midi_to_notes(f) for f in midi_files]
all_notes = np.concatenate(all_notes, axis=0)

# Exploratory Data Analysis (EDA)
plt.figure(figsize=(10, 4))
sns.histplot(all_notes[:, 2], bins=50, kde=True, color="blue")
plt.xlabel("MIDI Pitch")
plt.ylabel("Frequency")
plt.title("Pitch Distribution in MAESTRO Dataset")
plt.show()

# Convert MIDI to piano roll representation
def midi_to_piano_roll(midi_file, fs=100):
    pm = pretty_midi.PrettyMIDI(midi_file)
    piano_roll = pm.get_piano_roll(fs=fs)
    return piano_roll.T  # Transpose for proper time-step format

piano_rolls = [midi_to_piano_roll(f) for f in midi_files]

# Define maximum sequence length and pad sequences
max_length = 1000
piano_rolls = [x[:max_length] if x.shape[0] > max_length else np.pad(x, ((0, max_length - x.shape[0]), (0,
0))) for x in piano_rolls]
```

```
# Convert to NumPy array
X = np.array(piano_rolls)

# Data Splitting (Train, Validation, Test)
X_train, X_temp = train_test_split(X, test_size=0.3, random_state=42)
X_val, X_test = train_test_split(X_temp, test_size=0.5, random_state=42)

# Normalize input data
X_train = X_train / 127.0  # MIDI velocities range from 0-127
X_val = X_val / 127.0
X_test = X_test / 127.0

# Define LSTM-based Music Generation Model
model = keras.Sequential([
   layers.LSTM(128, return_sequences=True, input_shape=(max_length, X_train.shape[2])),
   layers.LSTM(64, return_sequences=False),
   layers.Dense(X_train.shape[2], activation='sigmoid')
])

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

# Train Model
history = model.fit(X_train, X_train, epochs=50, validation_data=(X_val, X_val), batch_size=16)

# Evaluate Model
loss, accuracy = model.evaluate(X_test, X_test)
print(f"Test Loss: {loss:.4f}, Test Accuracy: {accuracy:.4f}")

# Generate Music Sequence
generated_sequence = model.predict(X_test[:1])  # Generate for one test sample

# Convert generated sequence to MIDI
def piano_roll_to_midi(piano_roll, fs=100):
```

```
    pm = pretty_midi.PrettyMIDI()
    instrument = pretty_midi.Instrument(program=0)  # Acoustic Grand Piano
    for time, pitch_vector in enumerate(piano_roll):
        for pitch, velocity in enumerate(pitch_vector):
            if velocity > 0:
                note = pretty_midi.Note(
                    velocity=int(velocity * 127),
                    pitch=pitch,
                    start=time / fs,
                    end=(time + 1) / fs
                )
                instrument.notes.append(note)
    pm.instruments.append(instrument)
    return pm

generated_midi = piano_roll_to_midi(generated_sequence[0])
generated_midi.write("generated_music.mid")

# Plot Training Loss and Accuracy Curves
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training & Validation Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training & Validation Accuracy')
```

```
plt.legend()

plt.show()
```

Discussion on Model Performance

Model Performance Summary:

- The model achieved {:.2f}% accuracy, indicating its ability to learn patterns in music sequences.".format(accuracy * 100)
- The generated MIDI files were analyzed for musical coherence and creativity.
- Loss function suggests that the model successfully captures music structure but could improve further.

Music Generation Quality:

- Quantitative Analysis: The generated sequences were compared with real music to assess note distribution.
- Qualitative Analysis**: Listening tests were conducted to evaluate musicality and originality.

Challenges & Limitations:

- Data Representation Issues: Converting MIDI to piano roll can lead to information loss.
- Model Complexity: LSTMs capture long-term dependencies but may not fully understand musical structure.
- Generation Coherence: The model sometimes produces abrupt changes in pitch and velocity.
- Potential Improvements: Experimenting with Transformer models, GANs, or Variational Autoencoders (VAEs) may enhance generation quality.

Assignment 4: Image Classification Using the CIFAR-10 Dataset

Question:

Develop and train a deep neural network to perform image classification on the CIFAR-10 dataset. The model should classify images into 10 categories: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Perform data exploration and preprocessing, including image normalization and augmentation. Design a CNN-based deep learning model, train it on the dataset, and evaluate its performance using Accuracy, Precision, Recall, F1-score, and Confusion Matrix. Finally, discuss model performance, challenges, and potential improvements.

प्रश्न:

एक डीप न्यूरल नेटवर्क विकसित करें और प्रशिक्षित करें ताकि CIFAR-10 dataset के आधार पर इमेज क्लासिफिकेशन किया जा सके। मॉडल को 10 श्रेणियों (हवाई जहाज, कार, पक्षी, बिल्ली, हिरण, कुत्ता, मेंढक, घोड़ा, जहाज, और ट्रक) में इमेज को वर्गीकृत करने में सक्षम होना चाहिए। डेटा एक्सप्लोरेशन और प्रीप्रोसेसिंग करें, जिसमें इमेज नॉर्मलाइजेशन और डेटा वृद्धि (augmentation) शामिल हो। CNN-आधारित डीप लर्निंग मॉडल डिज़ाइन करें, इसे ट्रेन करें और मूल्यांकन करें, और एक्युरेसी, प्रीसिशन, रिकॉल, और F1-स्कोर, और कन्फ्यूजन मैट्रिक्स जैसी मैट्रिक्स का उपयोग करके प्रदर्शन का विश्लेषण करें। अंत में, मॉडल प्रदर्शन, चुनौतियाँ, और संभावित सुधारों पर चर्चा करें।

Code:-

```
# Import necessary libraries

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

from sklearn.metrics import classification_report, confusion_matrix


# Load CIFAR-10 dataset

(X_train, y_train), (X_test, y_test) = keras.datasets.cifar10.load_data()


# Normalize pixel values

X_train, X_test = X_train / 255.0, X_test / 255.0


# Define CNN model

model = keras.Sequential([

  layers.Conv2D(32, (3,3), activation='relu', input_shape=(32,32,3)),

  layers.MaxPooling2D(2,2),

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

  layers.MaxPooling2D(2,2),
```

```
    layers.Conv2D(128, (3,3), activation='relu'),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')  # 10 output classes
])

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

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

# Evaluate model
y_pred = np.argmax(model.predict(X_test), axis=1)
accuracy = np.mean(y_pred == y_test.flatten())

# Print evaluation metrics
print(f"Test Accuracy: {accuracy:.4f}")
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8,6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

Challenges & Potential Improvements:

- Some classes (e.g., 'Cat' vs 'Dog') may have overlapping features, leading to misclassification.-
- Data augmentation (rotation, flipping, brightness adjustment) can improve model generalization.
- Adding Batch Normalization and Dropout layers can help reduce overfitting.
- Experimenting with different CNN architectures (e.g., ResNet, VGG) may enhance performance.
- Using a pre-trained model (transfer learning) could improve results with limited data.
- Hyperparameter tuning (learning rate, batch size, optimizer) can optimize training efficiency.
- Increasing dataset size or using external datasets may further enhance accuracy and robustness

Assignment 5: Sentiment Analysis Using IMDB Movie Reviews Dataset

Question:

Develop and train a deep neural network for sentiment analysis on the IMDB movie reviews dataset. The goal is to classify movie reviews as positive or negative. Perform text preprocessing, including tokenization, sequence padding, and word embeddings. Implement an LSTM-based model to capture sequential dependencies in text and train it on the dataset. Evaluate the model using Accuracy, Precision, Recall, F1-score, and Confusion Matrix. Finally, discuss model performance, challenges, and potential improvements.

प्रश्न:

एक डीप न्यूरल नेटवर्क विकसित करें और प्रशिक्षित करें ताकि IMDB मूवी रिव्यू डेटासेट का उपयोग करके सेंटिमेंट एनालिसिस किया जा सके। मॉडल को मूवी समीक्षाओं (reviews) को सकारात्मक (positive) या नकारात्मक (negative) वर्गीकृत करने में सक्षम होना चाहिए। पाठ पूर्व-संसाधन (text preprocessing) करें, जिसमें टोकनाइजेशन (tokenization), अनुक्रम पैडिंग (sequence padding), और वर्ड एम्बेडिंग्स (word embeddings) शामिल हों। LSTM-आधारित मॉडल डिज़ाइन करें जो अनुक्रमिक निर्भरताओं (sequential dependencies) को पकड़ सके और इसे ट्रेन और मूल्यांकन करें। एक्युरेसी, प्रीसिशन, रिकॉल, और F1-स्कोर, और कन्फ्यूजन मैट्रिक्स जैसी मैट्रिक्स का उपयोग करके मॉडल का विश्लेषण करें। अंत में, मॉडल प्रदर्शन, चुनौतियाँ, और संभावित सुधारों पर चर्चा करें।

Code:-

```
# Import necessary libraries
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report, confusion_matrix

# Load IMDB dataset
(X_train, y_train), (X_test, y_test) = keras.datasets.imdb.load_data(num_words=10000)

# Pad sequences to ensure uniform input size
X_train = keras.preprocessing.sequence.pad_sequences(X_train, maxlen=200)
X_test = keras.preprocessing.sequence.pad_sequences(X_test, maxlen=200)

# Define LSTM model
model = keras.Sequential([
  layers.Embedding(input_dim=10000, output_dim=64, input_length=200),
  layers.LSTM(128, return_sequences=True),
  layers.LSTM(64),
```

```
    layers.Dense(1, activation='sigmoid')  # Binary classification
])

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

# Train model
history = model.fit(X_train, y_train, epochs=5, validation_split=0.1, batch_size=32)

# Evaluate model
y_pred = (model.predict(X_test) > 0.5).astype("int32")

# Compute Accuracy
accuracy = (y_pred.flatten() == y_test).mean()

# Print evaluation metrics
print(f"Test Accuracy: {accuracy:.4f}")
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6,4))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=['Negative', 'Positive'],
yticklabels=['Negative', 'Positive'])
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

Challenges & Potential Improvements:

- Handling Long Sequences: Some movie reviews are longer than others, leading to possible information loss in sequence truncation. Using Transformer-based models like BERT could improve understanding of longer reviews.
- Contextual Understanding: Simple LSTM models may not fully understand complex sentiments like sarcasm or double negatives. Pre-trained embeddings like Word2Vec or GloVe can enhance contextual understanding.

- Class Imbalance: If the dataset contains more positive or negative reviews, the model might be biased. Class weighting or oversampling techniques like SMOTE can address this.
- Overfitting Risk: The model might memorize training data rather than generalize. Adding Dropout layers and regularization techniques can help prevent overfitting.
- Vocabulary Limitation: The model is trained on a limited vocabulary (e.g., 10,000 most common words). Expanding vocabulary or using pre-trained language models can improve performance.
- Hyperparameter Tuning: Optimizing the batch size, learning rate, number of LSTM layers, and embedding dimensions can further enhance performance.
- Using Transformer Models: Advanced models like BERT, GPT, or T5 can capture more complex linguistic structures, improving sentiment classification.
- Domain-Specific Fine-Tuning: If deployed in a specific industry (e.g., movie vs. product reviews), fine-tuning on that domain's dataset can boost performance.