|  |
| --- |
| **Statement 1:**  **Train a Deep Neural Network on the MNIST dataset using the Adam optimizer with a learning rate of 0.001, and generate a classification report and ROC AUC plot.**  # Import necessary libraries  import numpy as np  import matplotlib.pyplot as plt  from sklearn.metrics import classification\_report, roc\_auc\_score, roc\_curve  from sklearn.preprocessing import label\_binarize  from tensorflow.keras.datasets import mnist  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense, Flatten  from tensorflow.keras.optimizers import Adam  from tensorflow.keras.utils import to\_categorical  from sklearn.metrics import RocCurveDisplay  # Load the MNIST dataset  (x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()  # Normalize the images to range [0, 1]  x\_train = x\_train / 255.0  x\_test = x\_test / 255.0  # Convert labels to one-hot encoding  y\_train\_cat = to\_categorical(y\_train, num\_classes=10)  y\_test\_cat = to\_categorical(y\_test, num\_classes=10)  # Build a simple deep neural network model  model = Sequential([  Flatten(input\_shape=(28, 28)),  Dense(128, activation='relu'),  Dense(64, activation='relu'),  Dense(10, activation='softmax') # 10 output classes  ])  # Compile the model using Adam optimizer with learning rate 0.001  optimizer = Adam(learning\_rate=0.001)  model.compile(optimizer=optimizer,  loss='categorical\_crossentropy',  metrics=['accuracy'])  # Train the model  history = model.fit(x\_train, y\_train\_cat, epochs=5, batch\_size=128, validation\_split=0.2)  # Evaluate the model  test\_loss, test\_accuracy = model.evaluate(x\_test, y\_test\_cat, verbose=0)  print(f"Test Accuracy: {test\_accuracy:.4f}")  # Predict classes  y\_pred\_probs = model.predict(x\_test)  y\_pred\_classes = np.argmax(y\_pred\_probs, axis=1)  # Classification report  print("\nClassification Report:")  print(classification\_report(y\_test, y\_pred\_classes))  # ROC AUC Score (Multiclass)  y\_test\_bin = label\_binarize(y\_test, classes=np.arange(10))  roc\_auc = roc\_auc\_score(y\_test\_bin, y\_pred\_probs, average='macro', multi\_class='ovr')  print(f"\nMulticlass ROC AUC Score: {roc\_auc:.4f}")  # Plot ROC Curves for all classes  plt.figure(figsize=(12, 8))  for i in range(10):  fpr, tpr, \_ = roc\_curve(y\_test\_bin[:, i], y\_pred\_probs[:, i])  plt.plot(fpr, tpr, label=f'Class {i} (AUC = {roc\_auc\_score(y\_test\_bin[:, i], y\_pred\_probs[:, i]):.2f})')  plt.plot([0, 1], [0, 1], 'k--') # Diagonal line  plt.xlabel('False Positive Rate')  plt.ylabel('True Positive Rate')  plt.title('ROC Curves for MNIST Classification')  plt.legend(loc='lower right')  plt.grid(True)  plt.show() |
| ***Statement 2***  **Train a DNN using the SGD optimizer with a learning rate of 0.0001 on the MNIST dataset and analyze the model's performance.**  # Import necessary libraries  import numpy as np  import matplotlib.pyplot as plt  import tensorflow as tf  from tensorflow.keras.datasets import mnist  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense, Flatten  from tensorflow.keras.optimizers import SGD  from tensorflow.keras.utils import to\_categorical  # Load the MNIST dataset  (x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()  # Normalize the input data to the range [0, 1]  x\_train = x\_train.astype('float32') / 255.0  x\_test = x\_test.astype('float32') / 255.0  # Convert labels to one-hot encoded vectors  y\_train = to\_categorical(y\_train, 10)  y\_test = to\_categorical(y\_test, 10)  # Define the Deep Neural Network model  model = Sequential([  Flatten(input\_shape=(28, 28)),  Dense(128, activation='relu'),  Dense(64, activation='relu'),  Dense(10, activation='softmax') # Output layer for 10 classes  ])  # Compile the model with SGD optimizer and low learning rate  optimizer = SGD(learning\_rate=0.0001)  model.compile(optimizer=optimizer,  loss='categorical\_crossentropy',  metrics=['accuracy'])  # Train the model  history = model.fit(x\_train, y\_train,  epochs=20,  batch\_size=128,  validation\_split=0.1,  verbose=2)  # Evaluate the model on test data  test\_loss, test\_acc = model.evaluate(x\_test, y\_test, verbose=0)  print(f"\nTest Accuracy: {test\_acc \* 100:.2f}%")  print(f"Test Loss: {test\_loss:.4f}")  # Plot training and validation accuracy and loss  plt.figure(figsize=(14, 5))  plt.subplot(1, 2, 1)  plt.plot(history.history['accuracy'], label='Train Accuracy')  plt.plot(history.history['val\_accuracy'], label='Val Accuracy')  plt.title('Accuracy over Epochs')  plt.xlabel('Epoch')  plt.ylabel('Accuracy')  plt.legend()  plt.grid(True)  plt.subplot(1, 2, 2)  plt.plot(history.history['loss'], label='Train Loss')  plt.plot(history.history['val\_loss'], label='Val Loss')  plt.title('Loss over Epochs')  plt.xlabel('Epoch')  plt.ylabel('Loss')  plt.legend()  plt.grid(True)  plt.tight\_layout()  plt.show() |
| ***Statement 3***  **Train a Deep Neural Network on the MNIST dataset using RMSprop optimizer with a learning rate of 0.0001, and compare results using an accuracy table and ROC curve.**  # Import required libraries  import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  import seaborn as sns  from sklearn.metrics import classification\_report, roc\_curve, auc  from sklearn.preprocessing import label\_binarize  from tensorflow.keras.datasets import mnist  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense, Flatten  from tensorflow.keras.optimizers import RMSprop  from tensorflow.keras.utils import to\_categorical  # Load MNIST dataset  (x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()  # Normalize pixel values  x\_train = x\_train.astype('float32') / 255.0  x\_test = x\_test.astype('float32') / 255.0  # One-hot encode the labels  y\_train\_cat = to\_categorical(y\_train, 10)  y\_test\_cat = to\_categorical(y\_test, 10)  # Build the DNN model  model = Sequential([  Flatten(input\_shape=(28, 28)),  Dense(128, activation='relu'),  Dense(64, activation='relu'),  Dense(10, activation='softmax')  ])  # Compile model with RMSprop optimizer  optimizer = RMSprop(learning\_rate=0.0001)  model.compile(optimizer=optimizer,  loss='categorical\_crossentropy',  metrics=['accuracy'])  # Train the model  history = model.fit(x\_train, y\_train\_cat,  epochs=15,  batch\_size=128,  validation\_split=0.1,  verbose=2)  # Evaluate model on test set  test\_loss, test\_acc = model.evaluate(x\_test, y\_test\_cat, verbose=0)  print(f"\nTest Accuracy: {test\_acc \* 100:.2f}%")  print(f"Test Loss: {test\_loss:.4f}")  # Predict class probabilities  y\_pred\_prob = model.predict(x\_test)  y\_pred\_classes = np.argmax(y\_pred\_prob, axis=1)  # ------------------ Accuracy Table (Classification Report) ------------------  report = classification\_report(y\_test, y\_pred\_classes, output\_dict=True)  report\_df = pd.DataFrame(report).transpose()  plt.figure(figsize=(10, 6))  sns.heatmap(report\_df.iloc[:-1, :-1], annot=True, fmt=".2f", cmap="Blues")  plt.title("Classification Report (Accuracy Table)")  plt.show()  # ------------------ ROC Curve ------------------  # Binarize labels for multi-class ROC  y\_test\_bin = label\_binarize(y\_test, classes=range(10))  fpr = {}  tpr = {}  roc\_auc = {}  # Calculate ROC for each class  for i in range(10):  fpr[i], tpr[i], \_ = roc\_curve(y\_test\_bin[:, i], y\_pred\_prob[:, i])  roc\_auc[i] = auc(fpr[i], tpr[i])  # Plot all ROC curves  plt.figure(figsize=(10, 8))  for i in range(10):  plt.plot(fpr[i], tpr[i], label=f'Class {i} (AUC = {roc\_auc[i]:.2f})')  plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')  plt.title('Multi-Class ROC Curve')  plt.xlabel('False Positive Rate')  plt.ylabel('True Positive Rate')  plt.legend()  plt.grid(True)  plt.show() |
| **Statement 4**  **Use SGD optimizer with a learning rate of 0.01 to train a DNN on the Wildfire dataset, then evaluate precision, recall, and F1-score with supporting bar plots.**  # Import 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.metrics import classification\_report  from sklearn.preprocessing import StandardScaler  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense  from tensorflow.keras.optimizers import SGD  from tensorflow.keras.utils import to\_categorical  # Load the Wildfire dataset (adjust file path if needed)  df = pd.read\_csv('wildfire.csv')  # Display first few rows  print("Dataset Preview:")  print(df.head())  # Assume last column is the target (binary: 0 = No fire, 1 = Fire)  X = df.iloc[:, :-1].values  y = df.iloc[:, -1].values  # Scale the features  scaler = StandardScaler()  X\_scaled = scaler.fit\_transform(X)  # One-hot encode the target if binary classification  y\_cat = to\_categorical(y)  # Split data into training and testing sets  x\_train, x\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y\_cat, test\_size=0.2, random\_state=42)  # Build the DNN model  model = Sequential([  Dense(64, activation='relu', input\_shape=(X.shape[1],)),  Dense(32, activation='relu'),  Dense(2, activation='softmax') # Binary classification with 2 output units  ])  # Compile model with SGD optimizer  optimizer = SGD(learning\_rate=0.01)  model.compile(optimizer=optimizer, loss='categorical\_crossentropy', metrics=['accuracy'])  # Train the model  history = model.fit(x\_train, y\_train, epochs=20, batch\_size=32, validation\_split=0.1, verbose=2)  # Evaluate on test data  loss, accuracy = model.evaluate(x\_test, y\_test, verbose=0)  print(f"\nTest Accuracy: {accuracy\*100:.2f}%")  print(f"Test Loss: {loss:.4f}")  # Predict labels for test data  y\_pred\_probs = model.predict(x\_test)  y\_pred\_classes = np.argmax(y\_pred\_probs, axis=1)  y\_true = np.argmax(y\_test, axis=1)  # Generate classification report  report = classification\_report(y\_true, y\_pred\_classes, output\_dict=True)  report\_df = pd.DataFrame(report).transpose()  # Display precision, recall, f1-score  metrics\_df = report\_df[['precision', 'recall', 'f1-score']].iloc[:2] # Only class 0 and 1  # Plotting  metrics\_df.plot(kind='bar', figsize=(8, 6), colormap='viridis')  plt.title('Precision, Recall & F1-Score for Wildfire Detection')  plt.xlabel('Class (0 = No Fire, 1 = Fire)')  plt.ylabel('Score')  plt.ylim(0, 1)  plt.grid(True)  plt.xticks(rotation=0)  plt.legend(loc='lower right')  plt.tight\_layout()  plt.show() |
| **Statement 5**  **Train a DNN on the Forest Fire dataset using RMSprop optimizer with a learning rate of 0.01. Report training and validation accuracy**  # Import libraries  import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import StandardScaler  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense  from tensorflow.keras.optimizers import RMSprop  from tensorflow.keras.utils import to\_categorical  # Load the Forest Fire dataset (adjust file path if necessary)  df = pd.read\_csv('forestfires.csv')  # Show dataset structure  print("Dataset Preview:")  print(df.head())  # Assuming the last column is the binary class (0: no fire, 1: fire)  X = df.iloc[:, :-1].values  y = df.iloc[:, -1].values  # Feature scaling  scaler = StandardScaler()  X\_scaled = scaler.fit\_transform(X)  # One-hot encode target for binary classification  y\_cat = to\_categorical(y)  # Split into training and test sets  x\_train, x\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y\_cat, test\_size=0.2, random\_state=42)  # Build the DNN model  model = Sequential([  Dense(64, activation='relu', input\_shape=(X.shape[1],)),  Dense(32, activation='relu'),  Dense(2, activation='softmax') # 2 neurons for binary classification  ])  # Compile with RMSprop optimizer and learning rate of 0.01  optimizer = RMSprop(learning\_rate=0.01)  model.compile(optimizer=optimizer, loss='categorical\_crossentropy', metrics=['accuracy'])  # Train the model and store history  history = model.fit(x\_train, y\_train,  epochs=20,  batch\_size=32,  validation\_split=0.2,  verbose=2)  # Plot training & validation accuracy  plt.figure(figsize=(8, 6))  plt.plot(history.history['accuracy'], label='Training Accuracy', marker='o')  plt.plot(history.history['val\_accuracy'], label='Validation Accuracy', marker='s')  plt.title('Training and Validation Accuracy')  plt.xlabel('Epochs')  plt.ylabel('Accuracy')  plt.grid(True)  plt.legend()  plt.tight\_layout()  plt.show() |
| **Statement 6**  **Compare DNN training using Adam and SGD optimizers (both with a learning rate of 0.001) on the Wildfire dataset**  # Import necessary libraries  import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import StandardScaler  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense  from tensorflow.keras.optimizers import SGD, Adam  from tensorflow.keras.utils import to\_categorical  # Load Wildfire dataset  df = pd.read\_csv('wildfire.csv')  # Display first few rows  print("Dataset Sample:")  print(df.head())  # Feature and label split  X = df.iloc[:, :-1].values  y = df.iloc[:, -1].values  # Normalize features  scaler = StandardScaler()  X\_scaled = scaler.fit\_transform(X)  # One-hot encode target  y\_cat = to\_categorical(y)  # Train-test split  x\_train, x\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y\_cat, test\_size=0.2, random\_state=42)  # Define a function to build the model  def build\_model(optimizer):  model = Sequential([  Dense(64, activation='relu', input\_shape=(X.shape[1],)),  Dense(32, activation='relu'),  Dense(2, activation='softmax') # Binary classification  ])  model.compile(optimizer=optimizer, loss='categorical\_crossentropy', metrics=['accuracy'])  return model  # Create and train model with SGD optimizer  sgd\_model = build\_model(SGD(learning\_rate=0.001))  history\_sgd = sgd\_model.fit(x\_train, y\_train, epochs=20, batch\_size=32,  validation\_split=0.2, verbose=0)  # Create and train model with Adam optimizer  adam\_model = build\_model(Adam(learning\_rate=0.001))  history\_adam = adam\_model.fit(x\_train, y\_train, epochs=20, batch\_size=32,  validation\_split=0.2, verbose=0)  # Plot training and validation accuracy for both  plt.figure(figsize=(10, 6))  plt.plot(history\_sgd.history['val\_accuracy'], label='SGD - Validation Accuracy', linestyle='--', marker='o')  plt.plot(history\_adam.history['val\_accuracy'], label='Adam - Validation Accuracy', linestyle='-', marker='s')  plt.plot(history\_sgd.history['accuracy'], label='SGD - Training Accuracy', linestyle='--', alpha=0.7)  plt.plot(history\_adam.history['accuracy'], label='Adam - Training Accuracy', linestyle='-', alpha=0.7)  plt.title('Comparison of DNN Training: SGD vs Adam (lr = 0.001)')  plt.xlabel('Epoch')  plt.ylabel('Accuracy')  plt.grid(True)  plt.legend()  plt.tight\_layout()  plt.show() |
| **Statement 7**  **Image Classification on MNIST Using DNN with Learning Rate Variation**   * **Use the MNIST dataset and build a DNN** * **Train the same model using learning rates: 0.01, 0.001** * **Use SGD optimizer and track accuracy for each run** * **Plot loss and accuracy for comparison**   import numpy as np  import matplotlib.pyplot as plt  from tensorflow.keras.datasets import mnist  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense, Flatten  from tensorflow.keras.optimizers import SGD  from tensorflow.keras.utils import to\_categorical  # Load MNIST data  (x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()  # Normalize pixel values  x\_train = x\_train.astype('float32') / 255.  x\_test = x\_test.astype('float32') / 255.  # One-hot encode labels  y\_train\_cat = to\_categorical(y\_train, 10)  y\_test\_cat = to\_categorical(y\_test, 10)  # Define function to build model  def build\_model():  model = Sequential([  Flatten(input\_shape=(28, 28)),  Dense(128, activation='relu'),  Dense(64, activation='relu'),  Dense(10, activation='softmax')  ])  return model  # Learning rates to compare  learning\_rates = [0.01, 0.001]  # Dictionaries to store histories  histories = {}  for lr in learning\_rates:  print(f"\nTraining model with learning rate = {lr}")  model = build\_model()  optimizer = SGD(learning\_rate=lr)  model.compile(optimizer=optimizer, loss='categorical\_crossentropy', metrics=['accuracy'])  history = model.fit(x\_train, y\_train\_cat, epochs=15, batch\_size=128, validation\_split=0.1, verbose=2)  histories[lr] = history  # Plot comparison of loss and accuracy  plt.figure(figsize=(14, 6))  # Plot Loss  plt.subplot(1, 2, 1)  for lr, history in histories.items():  plt.plot(history.history['loss'], label=f'Train Loss (lr={lr})')  plt.plot(history.history['val\_loss'], linestyle='--', label=f'Val Loss (lr={lr})')  plt.title('Training and Validation Loss')  plt.xlabel('Epochs')  plt.ylabel('Loss')  plt.legend()  plt.grid(True)  # Plot Accuracy  plt.subplot(1, 2, 2)  for lr, history in histories.items():  plt.plot(history.history['accuracy'], label=f'Train Acc (lr={lr})')  plt.plot(history.history['val\_accuracy'], linestyle='--', label=f'Val Acc (lr={lr})')  plt.title('Training and Validation Accuracy')  plt.xlabel('Epochs')  plt.ylabel('Accuracy')  plt.legend()  plt.grid(True)  plt.tight\_layout()  plt.show() |
| **Statement 8**  **Evaluating DNN on CIFAR-10 Using Batch Size Variation**   * **Load CIFAR-10 dataset** * **Use a feed-forward network with BatchNormalization** * **Train with batch sizes 32 and 64, keeping other parameters constant** * **Use Adam optimizer and train for 10 epochs** * **Compare accuracy and plot graphs**   import numpy as np  import matplotlib.pyplot as plt  from tensorflow.keras.datasets import cifar10  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense, Flatten, BatchNormalization, Activation  from tensorflow.keras.optimizers import Adam  from tensorflow.keras.utils import to\_categorical  # Load CIFAR-10 dataset  (x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data()  # Normalize pixel values  x\_train = x\_train.astype('float32') / 255.  x\_test = x\_test.astype('float32') / 255.  # One-hot encode labels  y\_train\_cat = to\_categorical(y\_train, 10)  y\_test\_cat = to\_categorical(y\_test, 10)  # Define model builder with BatchNormalization  def build\_model():  model = Sequential([  Flatten(input\_shape=(32, 32, 3)),  Dense(512),  BatchNormalization(),  Activation('relu'),  Dense(256),  BatchNormalization(),  Activation('relu'),  Dense(128),  BatchNormalization(),  Activation('relu'),  Dense(10, activation='softmax')  ])  return model  # Batch sizes to evaluate  batch\_sizes = [32, 64]  histories = {}  for batch\_size in batch\_sizes:  print(f"\nTraining with batch size = {batch\_size}")  model = build\_model()  model.compile(optimizer=Adam(), loss='categorical\_crossentropy', metrics=['accuracy'])    history = model.fit(x\_train, y\_train\_cat,  epochs=10,  batch\_size=batch\_size,  validation\_split=0.1,  verbose=2)    histories[batch\_size] = history  # Plot accuracy comparison  plt.figure(figsize=(14, 6))  plt.subplot(1, 2, 1)  for bs, history in histories.items():  plt.plot(history.history['accuracy'], label=f'Train Acc (batch={bs})')  plt.plot(history.history['val\_accuracy'], linestyle='--', label=f'Val Acc (batch={bs})')  plt.title('Training and Validation Accuracy')  plt.xlabel('Epoch')  plt.ylabel('Accuracy')  plt.legend()  plt.grid(True)  # Plot loss comparison  plt.subplot(1, 2, 2)  for bs, history in histories.items():  plt.plot(history.history['loss'], label=f'Train Loss (batch={bs})')  plt.plot(history.history['val\_loss'], linestyle='--', label=f'Val Loss (batch={bs})')  plt.title('Training and Validation Loss')  plt.xlabel('Epoch')  plt.ylabel('Loss')  plt.legend()  plt.grid(True)  plt.tight\_layout()  plt.show() |
| **Statement 9**  **Train a DNN on the UCI dataset using batch size 32 and a learning rate of 0.0001. Evaluate training time and accuracy**  import time  import numpy as np  import pandas as pd  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import StandardScaler, LabelEncoder  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense  from tensorflow.keras.optimizers import SGD  from tensorflow.keras.utils import to\_categorical  # Load UCI Wine Quality Dataset (Red Wine)  url = "https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv"  data = pd.read\_csv(url, sep=';')  # Features and target  X = data.drop('quality', axis=1).values  y = data['quality'].values  # Because quality values are integers from 3-8, we will treat this as a classification problem  # Convert target to categorical classes  # First encode labels to consecutive integers starting from 0  label\_encoder = LabelEncoder()  y\_encoded = label\_encoder.fit\_transform(y)  num\_classes = len(np.unique(y\_encoded))  y\_cat = to\_categorical(y\_encoded, num\_classes)  # Train-test split  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_cat, test\_size=0.2, random\_state=42, stratify=y\_cat)  # Feature scaling  scaler = StandardScaler()  X\_train = scaler.fit\_transform(X\_train)  X\_test = scaler.transform(X\_test)  # Build simple DNN model  def build\_model():  model = Sequential([  Dense(64, input\_shape=(X\_train.shape[1],), activation='relu'),  Dense(32, activation='relu'),  Dense(num\_classes, activation='softmax')  ])  return model  # Parameters  batch\_size = 32  learning\_rate = 0.0001  # Compile model  model = build\_model()  optimizer = SGD(learning\_rate=learning\_rate)  model.compile(optimizer=optimizer, loss='categorical\_crossentropy', metrics=['accuracy'])  # Train model with timing  start\_time = time.time()  history = model.fit(X\_train, y\_train, epochs=30, batch\_size=batch\_size, validation\_split=0.1, verbose=2)  end\_time = time.time()  training\_time = end\_time - start\_time  # Evaluate on test data  test\_loss, test\_accuracy = model.evaluate(X\_test, y\_test, verbose=0)  print(f"\nTraining time: {training\_time:.2f} seconds")  print(f"Test Accuracy: {test\_accuracy\*100:.2f}%") |

|  |
| --- |
| **Statement 10**  **Preprocess the Alphabet CSV dataset using label encoding and standard scaling, then train a simple DNN using batch size 32 and learning rate 0.0001**  import pandas as pd  import numpy as np  from sklearn.preprocessing import LabelEncoder, StandardScaler  from sklearn.model\_selection import train\_test\_split  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense  from tensorflow.keras.optimizers import SGD  from tensorflow.keras.utils import to\_categorical  # Load dataset (replace 'alphabet.csv' with your actual filename/path)  # For demonstration, assuming the last column is the target (alphabet labels)  data = pd.read\_csv('alphabet.csv')  # Separate features and target  X = data.iloc[:, :-1]  y = data.iloc[:, -1]  # Label encode target (alphabets to integers)  label\_encoder\_target = LabelEncoder()  y\_encoded = label\_encoder\_target.fit\_transform(y)  # Check for categorical features in X and label encode if any  for col in X.columns:  if X[col].dtype == 'object':  le = LabelEncoder()  X[col] = le.fit\_transform(X[col])  # Standard scale features  scaler = StandardScaler()  X\_scaled = scaler.fit\_transform(X)  # One-hot encode target for classification  num\_classes = len(np.unique(y\_encoded))  y\_cat = to\_categorical(y\_encoded, num\_classes)  # Train-test split  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y\_cat, test\_size=0.2, random\_state=42, stratify=y\_cat)  # Build simple DNN  def build\_model():  model = Sequential([  Dense(64, input\_shape=(X\_train.shape[1],), activation='relu'),  Dense(32, activation='relu'),  Dense(num\_classes, activation='softmax')  ])  return model  # Parameters  batch\_size = 32  learning\_rate = 0.0001  # Compile model  model = build\_model()  optimizer = SGD(learning\_rate=learning\_rate)  model.compile(optimizer=optimizer, loss='categorical\_crossentropy', metrics=['accuracy'])  # Train model  history = model.fit(X\_train, y\_train, epochs=30, batch\_size=batch\_size, validation\_split=0.1, verbose=2)  # Evaluate on test set  test\_loss, test\_accuracy = model.evaluate(X\_test, y\_test, verbose=0)  print(f"\nTest Accuracy: {test\_accuracy\*100:.2f}%") |
| **Statement 11**  **Use a batch size of 64 and learning rate of 0.001 to train a DNN on the UCI dataset. Document training accuracy and loss.**  import pandas as pd  import numpy as np  from sklearn.preprocessing import LabelEncoder, StandardScaler  from sklearn.model\_selection import train\_test\_split  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense  from tensorflow.keras.optimizers import SGD  from tensorflow.keras.utils import to\_categorical  import matplotlib.pyplot as plt  # Load UCI Alphabet dataset (Letter Recognition)  # Replace 'alphabet.csv' with your actual path if different  data = pd.read\_csv('alphabet.csv')  # Assuming last column is target (letters)  X = data.iloc[:, :-1]  y = data.iloc[:, -1]  # Label encode target (letters to integers)  label\_encoder\_target = LabelEncoder()  y\_encoded = label\_encoder\_target.fit\_transform(y)  # Check for categorical features in X and label encode if any (usually numeric, but just in case)  for col in X.columns:  if X[col].dtype == 'object':  le = LabelEncoder()  X[col] = le.fit\_transform(X[col])  # Standard scale features  scaler = StandardScaler()  X\_scaled = scaler.fit\_transform(X)  # One-hot encode target for classification  num\_classes = len(np.unique(y\_encoded))  y\_cat = to\_categorical(y\_encoded, num\_classes)  # Train-test split  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y\_cat, test\_size=0.2, random\_state=42, stratify=y\_cat)  # Build DNN model  def build\_model():  model = Sequential([  Dense(128, input\_shape=(X\_train.shape[1],), activation='relu'),  Dense(64, activation='relu'),  Dense(num\_classes, activation='softmax')  ])  return model  # Parameters  batch\_size = 64  learning\_rate = 0.001  # Compile model  model = build\_model()  optimizer = SGD(learning\_rate=learning\_rate)  model.compile(optimizer=optimizer, loss='categorical\_crossentropy', metrics=['accuracy'])  # Train model and record history  history = model.fit(X\_train, y\_train, epochs=30, batch\_size=batch\_size, validation\_split=0.1, verbose=2)  # Plot training accuracy and loss  plt.figure(figsize=(12,5))  plt.subplot(1,2,1)  plt.plot(history.history['accuracy'], label='Train Accuracy')  plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')  plt.title('Training and Validation Accuracy')  plt.xlabel('Epoch')  plt.ylabel('Accuracy')  plt.legend()  plt.subplot(1,2,2)  plt.plot(history.history['loss'], label='Train Loss')  plt.plot(history.history['val\_loss'], label='Validation Loss')  plt.title('Training and Validation Loss')  plt.xlabel('Epoch')  plt.ylabel('Loss')  plt.legend()  plt.show()  # Evaluate on test set  test\_loss, test\_accuracy = model.evaluate(X\_test, y\_test, verbose=0)  print(f"\nTest Accuracy: {test\_accuracy\*100:.2f}%") |
| **Statement 12**  **Preprocess the Alphabet dataset and train a CNN with the architecture using Adam optimizer, 20 epochs, batch size 64, and learning rate 0.001.**  import pandas as pd  import numpy as np  from sklearn.preprocessing import LabelEncoder, StandardScaler  from sklearn.model\_selection import train\_test\_split  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout  from tensorflow.keras.optimizers import Adam  from tensorflow.keras.utils import to\_categorical  import matplotlib.pyplot as plt  # Load Alphabet dataset  data = pd.read\_csv('alphabet.csv') # replace with your path  # Features and target  X = data.iloc[:, :-1]  y = data.iloc[:, -1]  # Label encode target  le\_target = LabelEncoder()  y\_encoded = le\_target.fit\_transform(y)  # Check for categorical features in X and encode if any (usually numeric)  for col in X.columns:  if X[col].dtype == 'object':  le = LabelEncoder()  X[col] = le.fit\_transform(X[col])  # Standard scale features  scaler = StandardScaler()  X\_scaled = scaler.fit\_transform(X)  # Number of samples and features  n\_samples, n\_features = X\_scaled.shape  # Reshape features to 2D for CNN input  # For example, reshape to (samples, 4, 4, 1) if features = 16  # If features != perfect square, pad with zeros  import math  def reshape\_for\_cnn(X):  n = X.shape[1]  sq = int(math.ceil(np.sqrt(n))) # smallest square side >= n  padded\_size = sq\*sq    # Pad zeros if needed  if padded\_size > n:  padding = np.zeros((X.shape[0], padded\_size - n))  X\_padded = np.hstack((X, padding))  else:  X\_padded = X    # Reshape to (samples, sq, sq, 1)  return X\_padded.reshape(-1, sq, sq, 1)  X\_cnn = reshape\_for\_cnn(X\_scaled)  # One-hot encode target  num\_classes = len(np.unique(y\_encoded))  y\_cat = to\_categorical(y\_encoded, num\_classes)  # Train-test split  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_cnn, y\_cat, test\_size=0.2, random\_state=42, stratify=y\_cat)  # Build CNN model  model = Sequential([  Conv2D(32, (3,3), activation='relu', input\_shape=X\_train.shape[1:]),  MaxPooling2D((2,2)),  Dropout(0.25),    Conv2D(64, (3,3), activation='relu'),  MaxPooling2D((2,2)),  Dropout(0.25),    Flatten(),  Dense(128, activation='relu'),  Dropout(0.5),  Dense(num\_classes, activation='softmax')  ])  # Compile  learning\_rate = 0.001  optimizer = Adam(learning\_rate=learning\_rate)  model.compile(optimizer=optimizer, loss='categorical\_crossentropy', metrics=['accuracy'])  # Train  history = model.fit(X\_train, y\_train, epochs=20, batch\_size=64, validation\_split=0.1, verbose=2)  # Plot training accuracy and loss  import matplotlib.pyplot as plt  plt.figure(figsize=(12,5))  plt.subplot(1,2,1)  plt.plot(history.history['accuracy'], label='Train Accuracy')  plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')  plt.title('Accuracy')  plt.xlabel('Epoch')  plt.ylabel('Accuracy')  plt.legend()  plt.subplot(1,2,2)  plt.plot(history.history['loss'], label='Train Loss')  plt.plot(history.history['val\_loss'], label='Validation Loss')  plt.title('Loss')  plt.xlabel('Epoch')  plt.ylabel('Loss')  plt.legend()  plt.show()  # Evaluate on test set  test\_loss, test\_acc = model.evaluate(X\_test, y\_test, verbose=0)  print(f"\nTest Accuracy: {test\_acc\*100:.2f}%") |
| **Statement 13**  **Compare the performance of a CNN and a DNN on the CIFAR-10 dataset. Highlight differences in accuracy and training time.**  import time  import numpy as np  import matplotlib.pyplot as plt  from tensorflow.keras.datasets import cifar10  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D  from tensorflow.keras.utils import to\_categorical  from tensorflow.keras.optimizers import Adam  from tensorflow.keras.callbacks import History  # Load CIFAR-10  (X\_train, y\_train), (X\_test, y\_test) = cifar10.load\_data()  # Normalize data  X\_train = X\_train.astype('float32') / 255.0  X\_test = X\_test.astype('float32') / 255.0  # One-hot encode labels  num\_classes = 10  y\_train\_cat = to\_categorical(y\_train, num\_classes)  y\_test\_cat = to\_categorical(y\_test, num\_classes)  # 1) Define DNN model (simple feedforward)  def build\_dnn():  model = Sequential([  Flatten(input\_shape=X\_train.shape[1:]),  Dense(512, activation='relu'),  Dense(256, activation='relu'),  Dense(128, activation='relu'),  Dense(num\_classes, activation='softmax')  ])  return model  # 2) Define CNN model  def build\_cnn():  model = Sequential([  Conv2D(32, (3,3), activation='relu', padding='same', input\_shape=X\_train.shape[1:]),  MaxPooling2D((2,2)),  Conv2D(64, (3,3), activation='relu', padding='same'),  MaxPooling2D((2,2)),  Conv2D(128, (3,3), activation='relu', padding='same'),  MaxPooling2D((2,2)),  Flatten(),  Dense(128, activation='relu'),  Dense(num\_classes, activation='softmax')  ])  return model  # Training parameters  batch\_size = 64  epochs = 15  learning\_rate = 0.001  optimizer = Adam(learning\_rate=learning\_rate)  # Train and time DNN  dnn = build\_dnn()  dnn.compile(optimizer=optimizer, loss='categorical\_crossentropy', metrics=['accuracy'])  start\_time = time.time()  history\_dnn = dnn.fit(X\_train, y\_train\_cat, epochs=epochs, batch\_size=batch\_size, validation\_split=0.1, verbose=2)  dnn\_time = time.time() - start\_time  # Evaluate DNN  dnn\_loss, dnn\_acc = dnn.evaluate(X\_test, y\_test\_cat, verbose=0)  # Train and time CNN  cnn = build\_cnn()  cnn.compile(optimizer=optimizer, loss='categorical\_crossentropy', metrics=['accuracy'])  start\_time = time.time()  history\_cnn = cnn.fit(X\_train, y\_train\_cat, epochs=epochs, batch\_size=batch\_size, validation\_split=0.1, verbose=2)  cnn\_time = time.time() - start\_time  # Evaluate CNN  cnn\_loss, cnn\_acc = cnn.evaluate(X\_test, y\_test\_cat, verbose=0)  # Print results  print(f"DNN Test Accuracy: {dnn\_acc\*100:.2f}% | Training time: {dnn\_time:.2f} seconds")  print(f"CNN Test Accuracy: {cnn\_acc\*100:.2f}% | Training time: {cnn\_time:.2f} seconds")  # Plot accuracy comparison  plt.figure(figsize=(10,5))  plt.plot(history\_dnn.history['val\_accuracy'], label='DNN Validation Accuracy')  plt.plot(history\_cnn.history['val\_accuracy'], label='CNN Validation Accuracy')  plt.title('Validation Accuracy: DNN vs CNN on CIFAR-10')  plt.xlabel('Epoch')  plt.ylabel('Accuracy')  plt.legend()  plt.show() |
| **Statement 14**  **Implement a Deep Neural Network (DNN) on the MNIST dataset using the Adam optimizer with a learning rate of 0.001 and plot training accuracy and loss.**  import matplotlib.pyplot as plt  from tensorflow.keras.datasets import mnist  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense, Flatten  from tensorflow.keras.optimizers import Adam  from tensorflow.keras.utils import to\_categorical  # Load MNIST data  (X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data()  # Normalize  X\_train = X\_train.astype('float32') / 255.0  X\_test = X\_test.astype('float32') / 255.0  # One-hot encode labels  num\_classes = 10  y\_train\_cat = to\_categorical(y\_train, num\_classes)  y\_test\_cat = to\_categorical(y\_test, num\_classes)  # Build DNN model  model = Sequential([  Flatten(input\_shape=(28, 28)),  Dense(512, activation='relu'),  Dense(256, activation='relu'),  Dense(num\_classes, activation='softmax')  ])  # Compile with Adam optimizer  optimizer = Adam(learning\_rate=0.001)  model.compile(optimizer=optimizer, loss='categorical\_crossentropy', metrics=['accuracy'])  # Train model  history = model.fit(X\_train, y\_train\_cat, epochs=20, batch\_size=64, validation\_split=0.1, verbose=2)  # Plot training accuracy and loss  plt.figure(figsize=(12, 5))  plt.subplot(1, 2, 1)  plt.plot(history.history['accuracy'], label='Train Accuracy')  plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')  plt.title('Accuracy')  plt.xlabel('Epoch')  plt.ylabel('Accuracy')  plt.legend()  plt.subplot(1, 2, 2)  plt.plot(history.history['loss'], label='Train Loss')  plt.plot(history.history['val\_loss'], label='Validation Loss')  plt.title('Loss')  plt.xlabel('Epoch')  plt.ylabel('Loss')  plt.legend()  plt.show() |
| **Statement 15**  **Implement *a DNN using RMSprop with learning rates 0.01 and 0.0001 on the Wildfire dataset. Compare training and validation performance.***  import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import StandardScaler, LabelEncoder  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense  from tensorflow.keras.optimizers import RMSprop  from tensorflow.keras.utils import to\_categorical  # Load Wildfire dataset (replace 'wildfire.csv' with actual path)  data = pd.read\_csv('wildfire.csv')  # Example preprocessing - adjust based on your dataset columns:  # Assume last column is the target variable  X = data.iloc[:, :-1].values  y = data.iloc[:, -1].values  # Encode target if categorical  if y.dtype == object or len(np.unique(y)) < 20:  le = LabelEncoder()  y = le.fit\_transform(y)  # If classification, convert to categorical  num\_classes = len(np.unique(y))  y\_cat = to\_categorical(y, num\_classes)  # Split data  X\_train, X\_val, y\_train, y\_val = train\_test\_split(X, y\_cat, test\_size=0.2, random\_state=42)  # Standard scale features  scaler = StandardScaler()  X\_train = scaler.fit\_transform(X\_train)  X\_val = scaler.transform(X\_val)  # Define DNN model builder function  def build\_model(input\_dim, num\_classes):  model = Sequential([  Dense(128, activation='relu', input\_dim=input\_dim),  Dense(64, activation='relu'),  Dense(num\_classes, activation='softmax')  ])  return model  # Training params  batch\_size = 32  epochs = 30  # Train with RMSprop lr=0.01  model\_high\_lr = build\_model(X\_train.shape[1], num\_classes)  model\_high\_lr.compile(optimizer=RMSprop(learning\_rate=0.01),  loss='categorical\_crossentropy', metrics=['accuracy'])  history\_high\_lr = model\_high\_lr.fit(X\_train, y\_train,  validation\_data=(X\_val, y\_val),  epochs=epochs, batch\_size=batch\_size, verbose=2)  # Train with RMSprop lr=0.0001  model\_low\_lr = build\_model(X\_train.shape[1], num\_classes)  model\_low\_lr.compile(optimizer=RMSprop(learning\_rate=0.0001),  loss='categorical\_crossentropy', metrics=['accuracy'])  history\_low\_lr = model\_low\_lr.fit(X\_train, y\_train,  validation\_data=(X\_val, y\_val),  epochs=epochs, batch\_size=batch\_size, verbose=2)  # Plot training and validation accuracy and loss for both learning rates  plt.figure(figsize=(14, 6))  plt.subplot(1, 2, 1)  plt.plot(history\_high\_lr.history['accuracy'], label='Train Acc (lr=0.01)')  plt.plot(history\_high\_lr.history['val\_accuracy'], label='Val Acc (lr=0.01)')  plt.plot(history\_low\_lr.history['accuracy'], label='Train Acc (lr=0.0001)')  plt.plot(history\_low\_lr.history['val\_accuracy'], label='Val Acc (lr=0.0001)')  plt.title('Training and Validation Accuracy')  plt.xlabel('Epoch')  plt.ylabel('Accuracy')  plt.legend()  plt.subplot(1, 2, 2)  plt.plot(history\_high\_lr.history['loss'], label='Train Loss (lr=0.01)')  plt.plot(history\_high\_lr.history['val\_loss'], label='Val Loss (lr=0.01)')  plt.plot(history\_low\_lr.history['loss'], label='Train Loss (lr=0.0001)')  plt.plot(history\_low\_lr.history['val\_loss'], label='Val Loss (lr=0.0001)')  plt.title('Training and Validation Loss')  plt.xlabel('Epoch')  plt.ylabel('Loss')  plt.legend()  plt.show() |
| **Statement 16**  **Multiclass classification using Deep Neural Networks: Example: Use the OCR letter recognition dataset/Alphabet.csv**  import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import LabelEncoder, StandardScaler  from sklearn.metrics import classification\_report  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense  from tensorflow.keras.utils import to\_categorical  # Load dataset (adjust path as needed)  data = pd.read\_csv('Alphabet.csv')  # Inspect columns, usually first column is label, rest are features  print(data.head())  # Separate features and labels  X = data.iloc[:, 1:].values # all columns except first are features  y = data.iloc[:, 0].values # first column is the label (letters)  # Encode labels (letters) to integers  le = LabelEncoder()  y\_enc = le.fit\_transform(y)  # One-hot encode output labels for multiclass classification  num\_classes = len(np.unique(y\_enc))  y\_cat = to\_categorical(y\_enc, num\_classes)  # Train/test split  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_cat, test\_size=0.2, random\_state=42)  # Standardize features  scaler = StandardScaler()  X\_train = scaler.fit\_transform(X\_train)  X\_test = scaler.transform(X\_test)  # Build DNN model  model = Sequential([  Dense(128, activation='relu', input\_shape=(X\_train.shape[1],)),  Dense(64, activation='relu'),  Dense(num\_classes, activation='softmax')  ])  # Compile model  model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])  # Train model  history = model.fit(X\_train, y\_train, epochs=30, batch\_size=64, validation\_split=0.1, verbose=2)  # Evaluate on test data  loss, accuracy = model.evaluate(X\_test, y\_test, verbose=0)  print(f"Test Accuracy: {accuracy\*100:.2f}%")  # Predict classes for test set  y\_pred\_prob = model.predict(X\_test)  y\_pred = np.argmax(y\_pred\_prob, axis=1)  y\_true = np.argmax(y\_test, axis=1)  # Classification report  print("\nClassification Report:\n")  print(classification\_report(y\_true, y\_pred, target\_names=le.classes\_))  # Plot accuracy and loss  plt.figure(figsize=(12, 5))  plt.subplot(1, 2, 1)  plt.plot(history.history['accuracy'], label='Train Accuracy')  plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')  plt.title('Accuracy')  plt.xlabel('Epoch')  plt.ylabel('Accuracy')  plt.legend()  plt.subplot(1, 2, 2)  plt.plot(history.history['loss'], label='Train Loss')  plt.plot(history.history['val\_loss'], label='Validation Loss')  plt.title('Loss')  plt.xlabel('Epoch')  plt.ylabel('Loss')  plt.legend()  plt.show() |
| **Statement 17**  **Implement the training of a DNN using Adam and SGD optimizers with a learning rate of 0.001 on the Wildfire dataset. Provide comparative plots.**  import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import StandardScaler, LabelEncoder  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense  from tensorflow.keras.optimizers import Adam, SGD  from tensorflow.keras.utils import to\_categorical  # Load Wildfire dataset - replace with your actual file path  data = pd.read\_csv('wildfire.csv')  # Example preprocessing - adjust based on your dataset structure  X = data.iloc[:, :-1].values  y = data.iloc[:, -1].values  # Encode target if categorical  if y.dtype == object or len(np.unique(y)) < 20:  le = LabelEncoder()  y = le.fit\_transform(y)  num\_classes = len(np.unique(y))  y\_cat = to\_categorical(y, num\_classes)  # Split data  X\_train, X\_val, y\_train, y\_val = train\_test\_split(X, y\_cat, test\_size=0.2, random\_state=42)  # Standardize features  scaler = StandardScaler()  X\_train = scaler.fit\_transform(X\_train)  X\_val = scaler.transform(X\_val)  # Build DNN model function  def build\_model(input\_dim, num\_classes):  model = Sequential([  Dense(128, activation='relu', input\_dim=input\_dim),  Dense(64, activation='relu'),  Dense(num\_classes, activation='softmax')  ])  return model  batch\_size = 32  epochs = 30  learning\_rate = 0.001  # Train with Adam optimizer  model\_adam = build\_model(X\_train.shape[1], num\_classes)  model\_adam.compile(optimizer=Adam(learning\_rate=learning\_rate),  loss='categorical\_crossentropy', metrics=['accuracy'])  history\_adam = model\_adam.fit(X\_train, y\_train,  validation\_data=(X\_val, y\_val),  epochs=epochs, batch\_size=batch\_size, verbose=2)  # Train with SGD optimizer  model\_sgd = build\_model(X\_train.shape[1], num\_classes)  model\_sgd.compile(optimizer=SGD(learning\_rate=learning\_rate),  loss='categorical\_crossentropy', metrics=['accuracy'])  history\_sgd = model\_sgd.fit(X\_train, y\_train,  validation\_data=(X\_val, y\_val),  epochs=epochs, batch\_size=batch\_size, verbose=2)  # Plot comparison graphs  plt.figure(figsize=(14, 6))  # Accuracy plot  plt.subplot(1, 2, 1)  plt.plot(history\_adam.history['accuracy'], label='Adam Train Acc')  plt.plot(history\_adam.history['val\_accuracy'], label='Adam Val Acc')  plt.plot(history\_sgd.history['accuracy'], label='SGD Train Acc')  plt.plot(history\_sgd.history['val\_accuracy'], label='SGD Val Acc')  plt.title('Training and Validation Accuracy')  plt.xlabel('Epoch')  plt.ylabel('Accuracy')  plt.legend()  # Loss plot  plt.subplot(1, 2, 2)  plt.plot(history\_adam.history['loss'], label='Adam Train Loss')  plt.plot(history\_adam.history['val\_loss'], label='Adam Val Loss')  plt.plot(history\_sgd.history['loss'], label='SGD Train Loss')  plt.plot(history\_sgd.history['val\_loss'], label='SGD Val Loss')  plt.title('Training and Validation Loss')  plt.xlabel('Epoch')  plt.ylabel('Loss')  plt.legend()  plt.show() |
| **Statement 18**  **Implement *a DNN using batch sizes 32 and 64 with a fixed learning rate of 0.001 on the UCI dataset. Compare model loss and performance.***  import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import LabelEncoder, StandardScaler  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense  from tensorflow.keras.optimizers import Adam  from tensorflow.keras.utils import to\_categorical  # Load UCI dataset - replace path accordingly  data = pd.read\_csv('uci\_dataset.csv')  # Example preprocessing (adjust based on dataset specifics)  X = data.iloc[:, :-1].values  y = data.iloc[:, -1].values  # Encode labels if categorical  if y.dtype == object or len(np.unique(y)) < 20:  le = LabelEncoder()  y = le.fit\_transform(y)  num\_classes = len(np.unique(y))  y\_cat = to\_categorical(y, num\_classes)  # Train/test split  X\_train, X\_val, y\_train, y\_val = train\_test\_split(X, y\_cat, test\_size=0.2, random\_state=42)  # Standardize features  scaler = StandardScaler()  X\_train = scaler.fit\_transform(X\_train)  X\_val = scaler.transform(X\_val)  # Build model function  def build\_model(input\_dim, num\_classes):  model = Sequential([  Dense(128, activation='relu', input\_dim=input\_dim),  Dense(64, activation='relu'),  Dense(num\_classes, activation='softmax')  ])  return model  learning\_rate = 0.001  epochs = 30  # Train with batch size 32  model\_32 = build\_model(X\_train.shape[1], num\_classes)  model\_32.compile(optimizer=Adam(learning\_rate=learning\_rate),  loss='categorical\_crossentropy', metrics=['accuracy'])  history\_32 = model\_32.fit(X\_train, y\_train,  validation\_data=(X\_val, y\_val),  epochs=epochs, batch\_size=32, verbose=2)  # Train with batch size 64  model\_64 = build\_model(X\_train.shape[1], num\_classes)  model\_64.compile(optimizer=Adam(learning\_rate=learning\_rate),  loss='categorical\_crossentropy', metrics=['accuracy'])  history\_64 = model\_64.fit(X\_train, y\_train,  validation\_data=(X\_val, y\_val),  epochs=epochs, batch\_size=64, verbose=2)  # Plot Loss comparison  plt.figure(figsize=(14, 6))  plt.subplot(1, 2, 1)  plt.plot(history\_32.history['loss'], label='Batch Size 32 - Train Loss')  plt.plot(history\_32.history['val\_loss'], label='Batch Size 32 - Val Loss')  plt.plot(history\_64.history['loss'], label='Batch Size 64 - Train Loss')  plt.plot(history\_64.history['val\_loss'], label='Batch Size 64 - Val Loss')  plt.title('Loss Comparison')  plt.xlabel('Epoch')  plt.ylabel('Loss')  plt.legend()  # Plot Accuracy comparison  plt.subplot(1, 2, 2)  plt.plot(history\_32.history['accuracy'], label='Batch Size 32 - Train Acc')  plt.plot(history\_32.history['val\_accuracy'], label='Batch Size 32 - Val Acc')  plt.plot(history\_64.history['accuracy'], label='Batch Size 64 - Train Acc')  plt.plot(history\_64.history['val\_accuracy'], label='Batch Size 64 - Val Acc')  plt.title('Accuracy Comparison')  plt.xlabel('Epoch')  plt.ylabel('Accuracy')  plt.legend()  plt.show() |
| **Statement 19**  ***Preprocess the Alphabet dataset and train both a DNN and a CNN. Use Adam optimizer with a batch size of 64. Compare accuracy across 20 epochs.***  import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import LabelEncoder, StandardScaler  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, Dropout  from tensorflow.keras.optimizers import Adam  from tensorflow.keras.utils import to\_categorical  # Load Alphabet dataset (adjust path as needed)  data = pd.read\_csv('Alphabet.csv')  # Separate features and target (assuming last column is label)  X = data.iloc[:, :-1].values  y = data.iloc[:, -1].values  # Encode labels  le = LabelEncoder()  y\_encoded = le.fit\_transform(y)  num\_classes = len(np.unique(y\_encoded))  y\_cat = to\_categorical(y\_encoded, num\_classes)  # Train-test split  X\_train, X\_val, y\_train, y\_val = train\_test\_split(X, y\_cat, test\_size=0.2, random\_state=42)  # Standardize features for DNN and CNN  scaler = StandardScaler()  X\_train\_scaled = scaler.fit\_transform(X\_train)  X\_val\_scaled = scaler.transform(X\_val)  # For CNN: reshape input into image format  # Alphabet dataset is usually 16x16 images (256 features), reshape accordingly  img\_dim = 16 # update if different  X\_train\_cnn = X\_train\_scaled.reshape(-1, img\_dim, img\_dim, 1)  X\_val\_cnn = X\_val\_scaled.reshape(-1, img\_dim, img\_dim, 1)  # Parameters  batch\_size = 64  epochs = 20  learning\_rate = 0.001  # Build DNN model  def build\_dnn(input\_dim, num\_classes):  model = Sequential([  Dense(256, activation='relu', input\_dim=input\_dim),  Dense(128, activation='relu'),  Dense(num\_classes, activation='softmax')  ])  return model  # Build CNN model  def build\_cnn(input\_shape, num\_classes):  model = Sequential([  Conv2D(32, (3, 3), activation='relu', input\_shape=input\_shape),  MaxPooling2D((2, 2)),  Conv2D(64, (3, 3), activation='relu'),  MaxPooling2D((2, 2)),  Flatten(),  Dense(128, activation='relu'),  Dropout(0.5),  Dense(num\_classes, activation='softmax')  ])  return model  # Train DNN  dnn = build\_dnn(X\_train\_scaled.shape[1], num\_classes)  dnn.compile(optimizer=Adam(learning\_rate=learning\_rate),  loss='categorical\_crossentropy', metrics=['accuracy'])  history\_dnn = dnn.fit(X\_train\_scaled, y\_train,  validation\_data=(X\_val\_scaled, y\_val),  epochs=epochs, batch\_size=batch\_size, verbose=2)  # Train CNN  cnn = build\_cnn(X\_train\_cnn.shape[1:], num\_classes)  cnn.compile(optimizer=Adam(learning\_rate=learning\_rate),  loss='categorical\_crossentropy', metrics=['accuracy'])  history\_cnn = cnn.fit(X\_train\_cnn, y\_train,  validation\_data=(X\_val\_cnn, y\_val),  epochs=epochs, batch\_size=batch\_size, verbose=2)  # Plot Accuracy Comparison  plt.figure(figsize=(12,5))  plt.plot(history\_dnn.history['val\_accuracy'], label='DNN Validation Accuracy')  plt.plot(history\_cnn.history['val\_accuracy'], label='CNN Validation Accuracy')  plt.title('Validation Accuracy Comparison')  plt.xlabel('Epoch')  plt.ylabel('Accuracy')  plt.legend()  plt.show() |
| **Statement 20**  **Classify Apple leaf images using a CNN without data augmentation for 10 epochs.**  dataset/  train/  class1/  class2/  ...  validation/  class1/  class2/  ...  import tensorflow as tf  from tensorflow.keras.preprocessing.image import ImageDataGenerator  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout  from tensorflow.keras.optimizers import Adam  import matplotlib.pyplot as plt  # Paths to dataset directories (update these paths)  train\_dir = 'dataset/train'  val\_dir = 'dataset/validation'  # Image parameters  img\_height, img\_width = 150, 150  batch\_size = 32  epochs = 10  learning\_rate = 0.001  # Use ImageDataGenerator for loading images WITHOUT augmentation  train\_datagen = ImageDataGenerator(rescale=1./255)  val\_datagen = ImageDataGenerator(rescale=1./255)  train\_generator = train\_datagen.flow\_from\_directory(  train\_dir,  target\_size=(img\_height, img\_width),  batch\_size=batch\_size,  class\_mode='categorical'  )  val\_generator = val\_datagen.flow\_from\_directory(  val\_dir,  target\_size=(img\_height, img\_width),  batch\_size=batch\_size,  class\_mode='categorical'  )  num\_classes = len(train\_generator.class\_indices)  # Build CNN model  model = Sequential([  Conv2D(32, (3, 3), activation='relu', input\_shape=(img\_height, img\_width, 3)),  MaxPooling2D((2, 2)),  Conv2D(64, (3, 3), activation='relu'),  MaxPooling2D((2, 2)),  Conv2D(128, (3, 3), activation='relu'),  MaxPooling2D((2, 2)),  Flatten(),  Dense(128, activation='relu'),  Dropout(0.5),  Dense(num\_classes, activation='softmax')  ])  model.compile(optimizer=Adam(learning\_rate=learning\_rate),  loss='categorical\_crossentropy',  metrics=['accuracy'])  # Train the model  history = model.fit(  train\_generator,  validation\_data=val\_generator,  epochs=epochs,  verbose=2  )  # Plot training & validation accuracy and loss  plt.figure(figsize=(12,5))  plt.subplot(1,2,1)  plt.plot(history.history['accuracy'], label='Train Accuracy')  plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')  plt.title('Accuracy')  plt.xlabel('Epoch')  plt.ylabel('Accuracy')  plt.legend()  plt.subplot(1,2,2)  plt.plot(history.history['loss'], label='Train Loss')  plt.plot(history.history['val\_loss'], label='Validation Loss')  plt.title('Loss')  plt.xlabel('Epoch')  plt.ylabel('Loss')  plt.legend()  plt.show() |
| **Statement 21**  **Implement *a CNN on Tomato dataset using batch sizes of 32 and 64 separately. Keep the learning*** |

|  |
| --- |
| ***rate fixed at 0.0001 and compare results.***  tomato\_dataset/  train/  class1/  class2/  validation/  class1/  class2/  import tensorflow as tf  from tensorflow.keras.preprocessing.image import ImageDataGenerator  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout  from tensorflow.keras.optimizers import Adam  import matplotlib.pyplot as plt  # Update these paths to your dataset location  train\_dir = 'tomato\_dataset/train'  val\_dir = 'tomato\_dataset/validation'  img\_height, img\_width = 150, 150  learning\_rate = 0.0001  epochs = 10  def create\_data\_generators(batch\_size):  train\_datagen = ImageDataGenerator(rescale=1./255)  val\_datagen = ImageDataGenerator(rescale=1./255)  train\_generator = train\_datagen.flow\_from\_directory(  train\_dir,  target\_size=(img\_height, img\_width),  batch\_size=batch\_size,  class\_mode='categorical',  shuffle=True  )  val\_generator = val\_datagen.flow\_from\_directory(  val\_dir,  target\_size=(img\_height, img\_width),  batch\_size=batch\_size,  class\_mode='categorical',  shuffle=False  )  return train\_generator, val\_generator  def build\_cnn\_model(input\_shape, num\_classes):  model = Sequential([  Conv2D(32, (3,3), activation='relu', input\_shape=input\_shape),  MaxPooling2D(2,2),  Conv2D(64, (3,3), activation='relu'),  MaxPooling2D(2,2),  Conv2D(128, (3,3), activation='relu'),  MaxPooling2D(2,2),  Flatten(),  Dense(128, activation='relu'),  Dropout(0.5),  Dense(num\_classes, activation='softmax')  ])  return model  # Train with batch size 32  batch\_size\_32 = 32  train\_gen\_32, val\_gen\_32 = create\_data\_generators(batch\_size\_32)  num\_classes = len(train\_gen\_32.class\_indices)  input\_shape = (img\_height, img\_width, 3)  model\_32 = build\_cnn\_model(input\_shape, num\_classes)  model\_32.compile(optimizer=Adam(learning\_rate=learning\_rate),  loss='categorical\_crossentropy',  metrics=['accuracy'])  history\_32 = model\_32.fit(  train\_gen\_32,  validation\_data=val\_gen\_32,  epochs=epochs,  verbose=2  )  # Train with batch size 64  batch\_size\_64 = 64  train\_gen\_64, val\_gen\_64 = create\_data\_generators(batch\_size\_64)  model\_64 = build\_cnn\_model(input\_shape, num\_classes)  model\_64.compile(optimizer=Adam(learning\_rate=learning\_rate),  loss='categorical\_crossentropy',  metrics=['accuracy'])  history\_64 = model\_64.fit(  train\_gen\_64,  validation\_data=val\_gen\_64,  epochs=epochs,  verbose=2  )  # Plot accuracy and loss comparison  plt.figure(figsize=(14,6))  plt.subplot(1,2,1)  plt.plot(history\_32.history['val\_accuracy'], label='Batch size 32')  plt.plot(history\_64.history['val\_accuracy'], label='Batch size 64')  plt.title('Validation Accuracy Comparison')  plt.xlabel('Epoch')  plt.ylabel('Accuracy')  plt.legend()  plt.subplot(1,2,2)  plt.plot(history\_32.history['val\_loss'], label='Batch size 32')  plt.plot(history\_64.history['val\_loss'], label='Batch size 64')  plt.title('Validation Loss Comparison')  plt.xlabel('Epoch')  plt.ylabel('Loss')  plt.legend()  plt.show() |
| **Statement 22**  **Implement *CNNs using Adam and RMSprop optimizers with a learning rate of 0.001 on Peach images. Record validation loss and accuracy.***  peach\_dataset/  train/  ripe/  img\_001.jpg  img\_002.jpg  ...  unripe/  img\_001.jpg  img\_002.jpg  ...  validation/  ripe/  img\_101.jpg  img\_102.jpg  ...  unripe/  img\_101.jpg  img\_102.jpg  ...  import tensorflow as tf  from tensorflow.keras.preprocessing.image import ImageDataGenerator  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout  from tensorflow.keras.optimizers import Adam, RMSprop  import matplotlib.pyplot as plt  # Update these paths to your Peach dataset locations  train\_dir = 'peach\_dataset/train'  val\_dir = 'peach\_dataset/validation'  img\_height, img\_width = 150, 150  learning\_rate = 0.001  epochs = 10  batch\_size = 32  def create\_data\_generators(batch\_size):  train\_datagen = ImageDataGenerator(rescale=1./255)  val\_datagen = ImageDataGenerator(rescale=1./255)  train\_generator = train\_datagen.flow\_from\_directory(  train\_dir,  target\_size=(img\_height, img\_width),  batch\_size=batch\_size,  class\_mode='categorical',  shuffle=True  )  val\_generator = val\_datagen.flow\_from\_directory(  val\_dir,  target\_size=(img\_height, img\_width),  batch\_size=batch\_size,  class\_mode='categorical',  shuffle=False  )  return train\_generator, val\_generator  def build\_cnn\_model(input\_shape, num\_classes):  model = Sequential([  Conv2D(32, (3,3), activation='relu', input\_shape=input\_shape),  MaxPooling2D(2,2),  Conv2D(64, (3,3), activation='relu'),  MaxPooling2D(2,2),  Conv2D(128, (3,3), activation='relu'),  MaxPooling2D(2,2),  Flatten(),  Dense(128, activation='relu'),  Dropout(0.5),  Dense(num\_classes, activation='softmax')  ])  return model  # Prepare data generators  train\_gen, val\_gen = create\_data\_generators(batch\_size)  num\_classes = len(train\_gen.class\_indices)  input\_shape = (img\_height, img\_width, 3)  # Model with Adam optimizer  model\_adam = build\_cnn\_model(input\_shape, num\_classes)  model\_adam.compile(optimizer=Adam(learning\_rate=learning\_rate),  loss='categorical\_crossentropy',  metrics=['accuracy'])  history\_adam = model\_adam.fit(  train\_gen,  validation\_data=val\_gen,  epochs=epochs,  verbose=2  )  # Model with RMSprop optimizer  model\_rmsprop = build\_cnn\_model(input\_shape, num\_classes)  model\_rmsprop.compile(optimizer=RMSprop(learning\_rate=learning\_rate),  loss='categorical\_crossentropy',  metrics=['accuracy'])  history\_rmsprop = model\_rmsprop.fit(  train\_gen,  validation\_data=val\_gen,  epochs=epochs,  verbose=2  )  # Plot validation accuracy and loss for comparison  plt.figure(figsize=(14,6))  plt.subplot(1,2,1)  plt.plot(history\_adam.history['val\_accuracy'], label='Adam')  plt.plot(history\_rmsprop.history['val\_accuracy'], label='RMSprop')  plt.title('Validation Accuracy')  plt.xlabel('Epoch')  plt.ylabel('Accuracy')  plt.legend()  plt.subplot(1,2,2)  plt.plot(history\_adam.history['val\_loss'], label='Adam')  plt.plot(history\_rmsprop.history['val\_loss'], label='RMSprop')  plt.title('Validation Loss')  plt.xlabel('Epoch')  plt.ylabel('Loss')  plt.legend()  plt.show() |
| **Statement 23**  **Build and train a CNN model for Apple image classification that includes Dropout layers. Train using 15 epochs and evaluate performance.**  import tensorflow as tf  from tensorflow.keras.preprocessing.image import ImageDataGenerator  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout  import matplotlib.pyplot as plt  # Set paths to your Apple image dataset folders  train\_dir = 'apple\_dataset/train'  val\_dir = 'apple\_dataset/validation'  img\_height, img\_width = 150, 150  batch\_size = 32  epochs = 15  # Data generators with rescaling  train\_datagen = ImageDataGenerator(rescale=1./255)  val\_datagen = ImageDataGenerator(rescale=1./255)  train\_generator = train\_datagen.flow\_from\_directory(  train\_dir,  target\_size=(img\_height, img\_width),  batch\_size=batch\_size,  class\_mode='categorical',  shuffle=True  )  val\_generator = val\_datagen.flow\_from\_directory(  val\_dir,  target\_size=(img\_height, img\_width),  batch\_size=batch\_size,  class\_mode='categorical',  shuffle=False  )  num\_classes = len(train\_generator.class\_indices)  # Build CNN model with Dropout  model = Sequential([  Conv2D(32, (3,3), activation='relu', input\_shape=(img\_height, img\_width, 3)),  MaxPooling2D(2,2),  Conv2D(64, (3,3), activation='relu'),  MaxPooling2D(2,2),  Conv2D(128, (3,3), activation='relu'),  MaxPooling2D(2,2),  Flatten(),  Dense(128, activation='relu'),  Dropout(0.5), # Dropout layer to reduce overfitting  Dense(num\_classes, activation='softmax')  ])  model.compile(optimizer='adam',  loss='categorical\_crossentropy',  metrics=['accuracy'])  # Train the model  history = model.fit(  train\_generator,  validation\_data=val\_generator,  epochs=epochs,  verbose=2  )  # Evaluate performance on validation set  val\_loss, val\_accuracy = model.evaluate(val\_generator)  print(f'Validation Loss: {val\_loss:.4f}')  print(f'Validation Accuracy: {val\_accuracy:.4f}')  # Plot training & validation accuracy and loss  plt.figure(figsize=(12,5))  plt.subplot(1,2,1)  plt.plot(history.history['accuracy'], label='Train Accuracy')  plt.plot(history.history['val\_accuracy'], label='Val Accuracy')  plt.title('Training & Validation Accuracy')  plt.xlabel('Epoch')  plt.ylabel('Accuracy')  plt.legend()  plt.subplot(1,2,2)  plt.plot(history.history['loss'], label='Train Loss')  plt.plot(history.history['val\_loss'], label='Val Loss')  plt.title('Training & Validation Loss')  plt.xlabel('Epoch')  plt.ylabel('Loss')  plt.legend()  plt.show() |
| **Statement 24**  **Split Grape image data into 70% train, 15% validation, and 15% test. Train a CNN for 10 epochs using a fixed learning rate of 0.001.**  path\_to\_dataset/  class1/  img1.jpg  img2.jpg  class2/  img3.jpg  ...  import tensorflow as tf  from tensorflow.keras import layers, models  from tensorflow.keras.optimizers import Adam  import matplotlib.pyplot as plt  import os  # Set dataset path  dataset\_dir = 'path\_to\_dataset' # e.g. 'Grape\_images/'  # Parameters  img\_height, img\_width = 150, 150  batch\_size = 32  learning\_rate = 0.001  epochs = 15  validation\_split = 0.15 # for validation set  test\_split = 0.15 # test set will be created separately below  # Load full dataset with validation split using image\_dataset\_from\_directory  full\_dataset = tf.keras.utils.image\_dataset\_from\_directory(  dataset\_dir,  shuffle=True,  image\_size=(img\_height, img\_width),  batch\_size=batch\_size,  validation\_split=validation\_split + test\_split,  subset="training",  seed=123  )  val\_test\_dataset = tf.keras.utils.image\_dataset\_from\_directory(  dataset\_dir,  shuffle=True,  image\_size=(img\_height, img\_width),  batch\_size=batch\_size,  validation\_split=validation\_split + test\_split,  subset="validation",  seed=123  )  # Split val\_test\_dataset into validation and test sets manually  val\_batches = int(len(val\_test\_dataset)\*validation\_split/(validation\_split + test\_split))  val\_dataset = val\_test\_dataset.take(val\_batches)  test\_dataset = val\_test\_dataset.skip(val\_batches)  # Normalize pixel values to [0,1]  normalization\_layer = layers.Rescaling(1./255)  full\_dataset = full\_dataset.map(lambda x, y: (normalization\_layer(x), y))  val\_dataset = val\_dataset.map(lambda x, y: (normalization\_layer(x), y))  test\_dataset = test\_dataset.map(lambda x, y: (normalization\_layer(x), y))  # Build CNN Model with Dropout  num\_classes = len(full\_dataset.class\_names)  model = models.Sequential([  layers.Conv2D(32, (3,3), activation='relu', input\_shape=(img\_height, img\_width, 3)),  layers.MaxPooling2D(2,2),  layers.Conv2D(64, (3,3), activation='relu'),  layers.MaxPooling2D(2,2),  layers.Conv2D(128, (3,3), activation='relu'),  layers.MaxPooling2D(2,2),  layers.Flatten(),  layers.Dropout(0.5),  layers.Dense(128, activation='relu'),  layers.Dense(num\_classes, activation='softmax')  ])  # Compile model  model.compile(  optimizer=Adam(learning\_rate=learning\_rate),  loss='sparse\_categorical\_crossentropy',  metrics=['accuracy']  )  # Train model  history = model.fit(  full\_dataset,  validation\_data=val\_dataset,  epochs=epochs  )  # Evaluate on test data  test\_loss, test\_acc = model.evaluate(test\_dataset)  print(f'Test accuracy: {test\_acc:.4f}')  print(f'Test loss: {test\_loss:.4f}')  # Plot training & validation accuracy and loss  plt.figure(figsize=(12,4))  plt.subplot(1,2,1)  plt.plot(history.history['accuracy'], label='Train Acc')  plt.plot(history.history['val\_accuracy'], label='Val Acc')  plt.title('Accuracy')  plt.xlabel('Epoch')  plt.ylabel('Accuracy')  plt.legend()  plt.subplot(1,2,2)  plt.plot(history.history['loss'], label='Train Loss')  plt.plot(history.history['val\_loss'], label='Val Loss')  plt.title('Loss')  plt.xlabel('Epoch')  plt.ylabel('Loss')  plt.legend()  plt.show() |
| **Statement 25**  **Use LeNet architecture to classify the Cats and Dogs dataset, and plot training loss and accuracy curves.**  import tensorflow as tf  from tensorflow.keras import layers, models  import matplotlib.pyplot as plt  import os  # Set dataset path (assumes directory structure like:  # cats\_and\_dogs/  # cats/  # cat1.jpg  # dogs/  # dog1.jpg  dataset\_dir = 'path\_to\_cats\_and\_dogs' # Change this path accordingly  # Parameters  img\_height, img\_width = 32, 32 # LeNet input size is 32x32 grayscale; we keep RGB and resize to 32x32  batch\_size = 32  epochs = 15  learning\_rate = 0.001  # Load dataset with 80/20 train-validation split  train\_ds = tf.keras.utils.image\_dataset\_from\_directory(  dataset\_dir,  validation\_split=0.2,  subset="training",  seed=123,  image\_size=(img\_height, img\_width),  batch\_size=batch\_size  )  val\_ds = tf.keras.utils.image\_dataset\_from\_directory(  dataset\_dir,  validation\_split=0.2,  subset="validation",  seed=123,  image\_size=(img\_height, img\_width),  batch\_size=batch\_size  )  # Normalize pixel values to [0,1]  normalization\_layer = layers.Rescaling(1./255)  train\_ds = train\_ds.map(lambda x, y: (normalization\_layer(x), y))  val\_ds = val\_ds.map(lambda x, y: (normalization\_layer(x), y))  # Define LeNet architecture  def LeNet():  model = models.Sequential()  model.add(layers.Conv2D(6, kernel\_size=(5,5), activation='tanh', input\_shape=(img\_height, img\_width, 3), padding='same'))  model.add(layers.AveragePooling2D())  model.add(layers.Conv2D(16, kernel\_size=(5,5), activation='tanh'))  model.add(layers.AveragePooling2D())  model.add(layers.Flatten())  model.add(layers.Dense(120, activation='tanh'))  model.add(layers.Dense(84, activation='tanh'))  model.add(layers.Dense(2, activation='softmax')) # 2 classes: cat, dog  return model  model = LeNet()  # Compile model  model.compile(  optimizer=tf.keras.optimizers.Adam(learning\_rate=learning\_rate),  loss='sparse\_categorical\_crossentropy',  metrics=['accuracy']  )  # Train model  history = model.fit(  train\_ds,  validation\_data=val\_ds,  epochs=epochs  )  # Plot training & validation accuracy and loss  plt.figure(figsize=(12,5))  plt.subplot(1,2,1)  plt.plot(history.history['accuracy'], label='Train Acc')  plt.plot(history.history['val\_accuracy'], label='Val Acc')  plt.title('Training and Validation Accuracy')  plt.xlabel('Epoch')  plt.ylabel('Accuracy')  plt.legend()  plt.subplot(1,2,2)  plt.plot(history.history['loss'], label='Train Loss')  plt.plot(history.history['val\_loss'], label='Val Loss')  plt.title('Training and Validation Loss')  plt.xlabel('Epoch')  plt.ylabel('Loss')  plt.legend()  plt.show() |
| **Statement 26**  **Use MobileNet architecture perform transfer learning on the Cats and Dogs dataset, and evaluate model performance using a classification report.**  import tensorflow as tf  from tensorflow.keras import layers, models  from tensorflow.keras.applications import MobileNet  from tensorflow.keras.applications.mobilenet import preprocess\_input  from tensorflow.keras.preprocessing import image\_dataset\_from\_directory  from sklearn.metrics import classification\_report  import numpy as np  import os  # Dataset directory structure (example):  # cats\_and\_dogs/  # cats/  # cat1.jpg  # ...  # dogs/  # dog1.jpg  # ...  dataset\_dir = 'path\_to\_cats\_and\_dogs' # Change to your actual path  # Parameters  img\_height, img\_width = 224, 224 # MobileNet default input size  batch\_size = 32  epochs = 10  learning\_rate = 0.0001  # Load datasets with 80/20 split  train\_ds = image\_dataset\_from\_directory(  dataset\_dir,  validation\_split=0.2,  subset="training",  seed=123,  image\_size=(img\_height, img\_width),  batch\_size=batch\_size  )  val\_ds = image\_dataset\_from\_directory(  dataset\_dir,  validation\_split=0.2,  subset="validation",  seed=123,  image\_size=(img\_height, img\_width),  batch\_size=batch\_size  )  # Preprocess input for MobileNet  train\_ds = train\_ds.map(lambda x, y: (preprocess\_input(x), y))  val\_ds = val\_ds.map(lambda x, y: (preprocess\_input(x), y))  # Cache and prefetch for performance optimization  AUTOTUNE = tf.data.AUTOTUNE  train\_ds = train\_ds.cache().prefetch(buffer\_size=AUTOTUNE)  val\_ds = val\_ds.cache().prefetch(buffer\_size=AUTOTUNE)  # Load MobileNet base model with pretrained weights, exclude top layers  base\_model = MobileNet(input\_shape=(img\_height, img\_width, 3),  include\_top=False,  weights='imagenet')  base\_model.trainable = False # Freeze base model layers initially  # Add classification head  model = models.Sequential([  base\_model,  layers.GlobalAveragePooling2D(),  layers.Dense(128, activation='relu'),  layers.Dropout(0.3),  layers.Dense(2, activation='softmax') # 2 classes: cats and dogs  ])  # Compile model  model.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=learning\_rate),  loss='sparse\_categorical\_crossentropy',  metrics=['accuracy'])  # Train the top layers first  history = model.fit(train\_ds, validation\_data=val\_ds, epochs=epochs)  # Optional: Fine-tune some base model layers  base\_model.trainable = True  # Fine-tune from this layer onwards  fine\_tune\_at = 100  for layer in base\_model.layers[:fine\_tune\_at]:  layer.trainable = False  # Recompile with lower learning rate for fine-tuning  model.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=learning\_rate/10),  loss='sparse\_categorical\_crossentropy',  metrics=['accuracy'])  # Continue training  fine\_tune\_epochs = 5  total\_epochs = epochs + fine\_tune\_epochs  history\_fine = model.fit(train\_ds,  validation\_data=val\_ds,  epochs=total\_epochs,  initial\_epoch=history.epoch[-1])  # Evaluate on validation set and print classification report  # Extract true labels and predictions  y\_true = []  y\_pred = []  for images, labels in val\_ds:  preds = model.predict(images)  y\_true.extend(labels.numpy())  y\_pred.extend(np.argmax(preds, axis=1))  print("Classification Report on Validation Set:")  print(classification\_report(y\_true, y\_pred, target\_names=train\_ds.class\_names)) |
| **Statement 27**  **Build both CNN and DNN models for the CIFAR-10 dataset, compare their accuracy and loss**  import tensorflow as tf  from tensorflow.keras import layers, models  import matplotlib.pyplot as plt  # Load CIFAR-10 dataset  (x\_train, y\_train), (x\_test, y\_test) = tf.keras.datasets.cifar10.load\_data()  # Normalize pixel values to [0,1]  x\_train, x\_test = x\_train / 255.0, x\_test / 255.0  # Flatten labels  y\_train = y\_train.flatten()  y\_test = y\_test.flatten()  num\_classes = 10  # Build CNN model  def build\_cnn():  model = models.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(num\_classes, activation='softmax')  ])  return model  # Build DNN model  def build\_dnn():  model = models.Sequential([  layers.Flatten(input\_shape=(32,32,3)),  layers.Dense(512, activation='relu'),  layers.Dense(256, activation='relu'),  layers.Dense(128, activation='relu'),  layers.Dense(num\_classes, activation='softmax')  ])  return model  # Compile and train model helper  def compile\_and\_train(model, epochs=15):  model.compile(optimizer='adam',  loss='sparse\_categorical\_crossentropy',  metrics=['accuracy'])  history = model.fit(x\_train, y\_train,  validation\_data=(x\_test, y\_test),  epochs=epochs,  batch\_size=64,  verbose=2)  return history  # Train CNN  cnn\_model = build\_cnn()  print("Training CNN model...")  cnn\_history = compile\_and\_train(cnn\_model)  # Train DNN  dnn\_model = build\_dnn()  print("\nTraining DNN model...")  dnn\_history = compile\_and\_train(dnn\_model)  # Plot accuracy and loss comparison  plt.figure(figsize=(12,5))  # Accuracy plot  plt.subplot(1,2,1)  plt.plot(cnn\_history.history['accuracy'], label='CNN Train Acc')  plt.plot(cnn\_history.history['val\_accuracy'], label='CNN Val Acc')  plt.plot(dnn\_history.history['accuracy'], label='DNN Train Acc')  plt.plot(dnn\_history.history['val\_accuracy'], label='DNN Val Acc')  plt.title('Training and Validation Accuracy')  plt.xlabel('Epochs')  plt.ylabel('Accuracy')  plt.legend()  # Loss plot  plt.subplot(1,2,2)  plt.plot(cnn\_history.history['loss'], label='CNN Train Loss')  plt.plot(cnn\_history.history['val\_loss'], label='CNN Val Loss')  plt.plot(dnn\_history.history['loss'], label='DNN Train Loss')  plt.plot(dnn\_history.history['val\_loss'], label='DNN Val Loss')  plt.title('Training and Validation Loss')  plt.xlabel('Epochs')  plt.ylabel('Loss')  plt.legend()  plt.show() |
| **Statement 28**  **Implement an RNN on the GOOGL.csv dataset and compare its training time and loss curve with an LSTM model.**  import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  import time  from sklearn.preprocessing import MinMaxScaler  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import SimpleRNN, LSTM, Dense  from tensorflow.keras.optimizers import Adam  # Load the GOOGL.csv dataset (make sure it's in the working directory)  data = pd.read\_csv('GOOGL.csv')  # Assuming dataset has 'Date' and 'Close' columns; focus on 'Close' prices  close\_prices = data['Close'].values.reshape(-1, 1)  # Normalize prices between 0 and 1  scaler = MinMaxScaler()  scaled\_close = scaler.fit\_transform(close\_prices)  # Prepare time series sequences  def create\_sequences(data, seq\_length=20):  X, y = [], []  for i in range(len(data) - seq\_length):  X.append(data[i:i+seq\_length])  y.append(data[i+seq\_length])  return np.array(X), np.array(y)  SEQ\_LENGTH = 20  X, y = create\_sequences(scaled\_close, SEQ\_LENGTH)  # Split into train and test sets (e.g., 80%-20%)  split = int(0.8 \* len(X))  X\_train, X\_test = X[:split], X[split:]  y\_train, y\_test = y[:split], y[split:]  # Build Simple RNN model  def build\_rnn():  model = Sequential([  SimpleRNN(50, activation='tanh', input\_shape=(SEQ\_LENGTH, 1)),  Dense(1)  ])  model.compile(optimizer=Adam(), loss='mse')  return model  # Build LSTM model  def build\_lstm():  model = Sequential([  LSTM(50, activation='tanh', input\_shape=(SEQ\_LENGTH, 1)),  Dense(1)  ])  model.compile(optimizer=Adam(), loss='mse')  return model  # Train and record time + loss history  def train\_model(model, epochs=30, batch\_size=32):  start\_time = time.time()  history = model.fit(X\_train, y\_train, epochs=epochs, batch\_size=batch\_size,  validation\_data=(X\_test, y\_test), verbose=2)  end\_time = time.time()  training\_time = end\_time - start\_time  return history, training\_time  # Prepare data shape (samples, seq\_length, features)  X\_train = X\_train.reshape((X\_train.shape[0], SEQ\_LENGTH, 1))  X\_test = X\_test.reshape((X\_test.shape[0], SEQ\_LENGTH, 1))  # Train RNN  print("Training Simple RNN model...")  rnn\_model = build\_rnn()  rnn\_history, rnn\_time = train\_model(rnn\_model)  # Train LSTM  print("\nTraining LSTM model...")  lstm\_model = build\_lstm()  lstm\_history, lstm\_time = train\_model(lstm\_model)  # Plot loss curves  plt.figure(figsize=(10,5))  plt.plot(rnn\_history.history['loss'], label='RNN Train Loss')  plt.plot(rnn\_history.history['val\_loss'], label='RNN Val Loss')  plt.plot(lstm\_history.history['loss'], label='LSTM Train Loss')  plt.plot(lstm\_history.history['val\_loss'], label='LSTM Val Loss')  plt.title('Training and Validation Loss')  plt.xlabel('Epoch')  plt.ylabel('Loss (MSE)')  plt.legend()  plt.show()  # Print training time comparison  print(f"Simple RNN Training Time: {rnn\_time:.2f} seconds")  print(f"LSTM Training Time: {lstm\_time:.2f} seconds") |
| **Statement 29**  **Use transfer learning with VGG16 on the Cats and Dogs dataset, freezing the first 4 layers, and train the classifier and evaluate model performance using a classification report.**  cats\_and\_dogs/  ├── train/  │ ├── cats/  │ └── dogs/  ├── validation/  │ ├── cats/  │ └── dogs/  import tensorflow as tf  from tensorflow.keras.preprocessing.image import ImageDataGenerator  from tensorflow.keras.applications import VGG16  from tensorflow.keras.layers import Flatten, Dense, Dropout  from tensorflow.keras.models import Model  from tensorflow.keras.optimizers import Adam  from sklearn.metrics import classification\_report, confusion\_matrix  import numpy as np  import os  # Define paths (adjust to your dataset folder structure)  base\_dir = 'cats\_and\_dogs' # folder containing 'train' and 'validation' subfolders  train\_dir = os.path.join(base\_dir, 'train')  val\_dir = os.path.join(base\_dir, 'validation')  # Parameters  IMG\_SIZE = (224, 224)  BATCH\_SIZE = 32  EPOCHS = 10  LEARNING\_RATE = 0.0001  # Data Generators  train\_datagen = ImageDataGenerator(  rescale=1./255,  horizontal\_flip=True,  rotation\_range=15,  zoom\_range=0.1  )  val\_datagen = ImageDataGenerator(rescale=1./255)  train\_generator = train\_datagen.flow\_from\_directory(  train\_dir,  target\_size=IMG\_SIZE,  batch\_size=BATCH\_SIZE,  class\_mode='binary'  )  val\_generator = val\_datagen.flow\_from\_directory(  val\_dir,  target\_size=IMG\_SIZE,  batch\_size=BATCH\_SIZE,  class\_mode='binary',  shuffle=False  )  # Load VGG16 base model without top layers  base\_model = VGG16(weights='imagenet', include\_top=False, input\_shape=(\*IMG\_SIZE, 3))  # Freeze first 4 layers  for layer in base\_model.layers[:4]:  layer.trainable = False  for layer in base\_model.layers[4:]:  layer.trainable = True  # Add custom classification head  x = base\_model.output  x = Flatten()(x)  x = Dense(256, activation='relu')(x)  x = Dropout(0.5)(x)  output = Dense(1, activation='sigmoid')(x) # Binary classification  model = Model(inputs=base\_model.input, outputs=output)  # Compile model  model.compile(optimizer=Adam(learning\_rate=LEARNING\_RATE),  loss='binary\_crossentropy',  metrics=['accuracy'])  # Train model  history = model.fit(  train\_generator,  epochs=EPOCHS,  validation\_data=val\_generator  )  # Predict on validation data  val\_generator.reset()  preds = model.predict(val\_generator)  predicted\_classes = (preds > 0.5).astype(int).reshape(-1)  # True classes  true\_classes = val\_generator.classes  # Classification report  target\_names = list(train\_generator.class\_indices.keys())  print(classification\_report(true\_classes, predicted\_classes, target\_names=target\_names))  # Optionally print confusion matrix  print("Confusion Matrix:")  print(confusion\_matrix(true\_classes, predicted\_classes)) |
| **Statement 30**  ***Load and visualize sample images from the Potato dataset,train CNN for 5 epochs***  potato\_dataset/  ├── class1/  ├── class2/  └── ...  import matplotlib.pyplot as plt  import tensorflow as tf  from tensorflow.keras.preprocessing.image import ImageDataGenerator  from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout  from tensorflow.keras.models import Sequential  import os  import numpy as np  # Define dataset path - update this to your local Potato dataset directory  dataset\_dir = 'potato\_dataset' # should contain subfolders for each class  # Parameters  IMG\_SIZE = (128, 128)  BATCH\_SIZE = 32  EPOCHS = 5  # Data generators with simple augmentation for training, rescale only for validation  train\_datagen = ImageDataGenerator(  rescale=1./255,  validation\_split=0.2  )  train\_generator = train\_datagen.flow\_from\_directory(  dataset\_dir,  target\_size=IMG\_SIZE,  batch\_size=BATCH\_SIZE,  class\_mode='categorical',  subset='training',  shuffle=True  )  val\_generator = train\_datagen.flow\_from\_directory(  dataset\_dir,  target\_size=IMG\_SIZE,  batch\_size=BATCH\_SIZE,  class\_mode='categorical',  subset='validation',  shuffle=False  )  # Visualize some sample images  def plot\_sample\_images(generator):  images, labels = next(generator) # batch of images and labels  class\_indices = {v: k for k, v in generator.class\_indices.items()}  plt.figure(figsize=(10, 10))  for i in range(9):  plt.subplot(3, 3, i + 1)  plt.imshow(images[i])  label\_index = np.argmax(labels[i])  plt.title(class\_indices[label\_index])  plt.axis('off')  plt.show()  plot\_sample\_images(train\_generator)  # Build simple CNN model  model = Sequential([  Conv2D(32, (3,3), activation='relu', input\_shape=(\*IMG\_SIZE, 3)),  MaxPooling2D(2, 2),    Conv2D(64, (3,3), activation='relu'),  MaxPooling2D(2, 2),    Conv2D(128, (3,3), activation='relu'),  MaxPooling2D(2, 2),    Flatten(),  Dense(128, activation='relu'),  Dropout(0.5),  Dense(train\_generator.num\_classes, activation='softmax')  ])  model.compile(optimizer='adam',  loss='categorical\_crossentropy',  metrics=['accuracy'])  # Train model  history = model.fit(  train\_generator,  epochs=EPOCHS,  validation\_data=val\_generator  ) |
| **Statement 31**  **Implement *LSTM models on GOOGL.csv with learning rates 0.001 and 0.0001 for 20 and 50 epochs. Compare accuracy and convergence.***  your\_project\_folder/  ├── GOOGL.csv  ├── lstm\_googl.py # (your code file, if saving separately)  import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  import tensorflow as tf  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import LSTM, Dense, Dropout  from sklearn.preprocessing import MinMaxScaler  from sklearn.metrics import mean\_squared\_error  # Load dataset  df = pd.read\_csv('GOOGL.csv') # Ensure file is in your working directory  df = df[['Date', 'Close']]  df['Date'] = pd.to\_datetime(df['Date'])  df.sort\_values('Date', inplace=True)  df.dropna(inplace=True)  # Normalize closing prices  scaler = MinMaxScaler()  scaled\_data = scaler.fit\_transform(df[['Close']])  # Prepare sequences for LSTM  def create\_sequences(data, time\_steps=60):  X, y = [], []  for i in range(time\_steps, len(data)):  X.append(data[i - time\_steps:i])  y.append(data[i])  return np.array(X), np.array(y)  time\_steps = 60  X, y = create\_sequences(scaled\_data, time\_steps)  # Split into train/test  train\_size = int(len(X) \* 0.8)  X\_train, X\_test = X[:train\_size], X[train\_size:]  y\_train, y\_test = y[:train\_size], y[train\_size:]  # Define function to build LSTM model  def build\_model(learning\_rate):  model = Sequential([  LSTM(50, return\_sequences=True, input\_shape=(X\_train.shape[1], 1)),  Dropout(0.2),  LSTM(50),  Dropout(0.2),  Dense(1)  ])  optimizer = tf.keras.optimizers.Adam(learning\_rate=learning\_rate)  model.compile(loss='mean\_squared\_error', optimizer=optimizer, metrics=['mae'])  return model  # Train models with different settings  configs = [  {'lr': 0.001, 'epochs': 20},  {'lr': 0.0001, 'epochs': 50}  ]  histories = []  for cfg in configs:  print(f"\nTraining with LR={cfg['lr']} for {cfg['epochs']} epochs")  model = build\_model(cfg['lr'])  history = model.fit(  X\_train, y\_train,  epochs=cfg['epochs'],  batch\_size=32,  validation\_data=(X\_test, y\_test),  verbose=1  )  histories.append((cfg, history))  # Plot training and validation loss  for cfg, history in histories:  plt.plot(history.history['val\_loss'], label=f"Val Loss (LR={cfg['lr']}, E={cfg['epochs']})")  plt.plot(history.history['loss'], linestyle='--', label=f"Train Loss (LR={cfg['lr']}, E={cfg['epochs']})")  plt.title("LSTM Loss Comparison")  plt.xlabel("Epochs")  plt.ylabel("Loss (MSE)")  plt.legend()  plt.grid(True)  plt.show() |
| **Statement 32**  **Implement a CNN on Tomato dataset using batch sizes of 32 and 64 separately. Keep the learning rate fixed at 0.0001 and compare results.**  project\_folder/  ├── tomato\_data/  │ ├── train/  │ │ ├── class1/  │ │ └── class2/  │ ├── val/  │ │ ├── class1/  │ │ └── class2/  import tensorflow as tf  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout  from tensorflow.keras.preprocessing.image import ImageDataGenerator  import matplotlib.pyplot as plt  # Constants  IMG\_SIZE = (128, 128)  EPOCHS = 10  LEARNING\_RATE = 0.0001  DATA\_DIR = 'tomato\_data' # Adjust path if needed  # Data preparation  datagen = ImageDataGenerator(rescale=1./255)  train\_batches = {}  val\_batches = {}  for batch\_size in [32, 64]:  train\_batches[batch\_size] = datagen.flow\_from\_directory(  DATA\_DIR + '/train',  target\_size=IMG\_SIZE,  batch\_size=batch\_size,  class\_mode='categorical',  shuffle=True  )  val\_batches[batch\_size] = datagen.flow\_from\_directory(  DATA\_DIR + '/val',  target\_size=IMG\_SIZE,  batch\_size=batch\_size,  class\_mode='categorical',  shuffle=False  )  # Define CNN model builder  def build\_model(input\_shape, num\_classes):  model = Sequential([  Conv2D(32, (3, 3), activation='relu', input\_shape=input\_shape),  MaxPooling2D(2, 2),  Conv2D(64, (3, 3), activation='relu'),  MaxPooling2D(2, 2),  Flatten(),  Dense(128, activation='relu'),  Dropout(0.5),  Dense(num\_classes, activation='softmax')  ])  optimizer = tf.keras.optimizers.Adam(learning\_rate=LEARNING\_RATE)  model.compile(optimizer=optimizer, loss='categorical\_crossentropy', metrics=['accuracy'])  return model  # Train and evaluate for both batch sizes  histories = {}  for batch\_size in [32, 64]:  print(f"\nTraining with batch size {batch\_size}")  model = build\_model((IMG\_SIZE[0], IMG\_SIZE[1], 3), train\_batches[batch\_size].num\_classes)  history = model.fit(  train\_batches[batch\_size],  validation\_data=val\_batches[batch\_size],  epochs=EPOCHS,  verbose=1  )  histories[batch\_size] = history  # Plot results  for metric in ['loss', 'accuracy']:  plt.figure(figsize=(8, 5))  for batch\_size in [32, 64]:  plt.plot(histories[batch\_size].history[metric], label=f'Train {metric} (BS={batch\_size})')  plt.plot(histories[batch\_size].history['val\_' + metric], linestyle='--', label=f'Val {metric} (BS={batch\_size})')  plt.title(f'CNN {metric.capitalize()} Comparison')  plt.xlabel('Epochs')  plt.ylabel(metric.capitalize())  plt.legend()  plt.grid(True)  plt.show() |
| **Statement 34**  **Implement CNN model on Potato leaf images using the Adam optimizer and i Use a learning rate of** |

|  |
| --- |
| * 1. **evaluate model Performance**   project\_folder/  ├── potato\_data/  │ ├── train/  │ │ ├── class1/  │ │ └── class2/  │ └── val/  │ ├── class1/  │ └── class2/  import tensorflow as tf  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout  from tensorflow.keras.preprocessing.image import ImageDataGenerator  import matplotlib.pyplot as plt  from sklearn.metrics import classification\_report, confusion\_matrix  import numpy as np  # Constants  IMG\_SIZE = (128, 128)  BATCH\_SIZE = 32  EPOCHS = 10  LEARNING\_RATE = 0.001  DATA\_DIR = 'potato\_data' # change to your dataset path  # Image preprocessing  datagen = ImageDataGenerator(rescale=1./255)  train\_gen = datagen.flow\_from\_directory(  DATA\_DIR + '/train',  target\_size=IMG\_SIZE,  batch\_size=BATCH\_SIZE,  class\_mode='categorical',  shuffle=True  )  val\_gen = datagen.flow\_from\_directory(  DATA\_DIR + '/val',  target\_size=IMG\_SIZE,  batch\_size=BATCH\_SIZE,  class\_mode='categorical',  shuffle=False  )  # Build CNN model  model = Sequential([  Conv2D(32, (3,3), activation='relu', input\_shape=(IMG\_SIZE[0], IMG\_SIZE[1], 3)),  MaxPooling2D(2,2),  Conv2D(64, (3,3), activation='relu'),  MaxPooling2D(2,2),  Flatten(),  Dense(128, activation='relu'),  Dropout(0.5),  Dense(train\_gen.num\_classes, activation='softmax')  ])  # Compile model with Adam optimizer  model.compile(  optimizer=tf.keras.optimizers.Adam(learning\_rate=LEARNING\_RATE),  loss='categorical\_crossentropy',  metrics=['accuracy']  )  # Train the model  history = model.fit(  train\_gen,  validation\_data=val\_gen,  epochs=EPOCHS,  verbose=1  )  # Evaluate model performance  val\_loss, val\_acc = model.evaluate(val\_gen)  print(f"\nValidation Loss: {val\_loss:.4f}")  print(f"Validation Accuracy: {val\_acc:.4f}")  # Classification Report  val\_gen.reset()  y\_pred = model.predict(val\_gen)  y\_pred\_classes = np.argmax(y\_pred, axis=1)  y\_true = val\_gen.classes  class\_labels = list(val\_gen.class\_indices.keys())  print("\nClassification Report:")  print(classification\_report(y\_true, y\_pred\_classes, target\_names=class\_labels))  # Plotting training curves  plt.figure(figsize=(12,5))  plt.subplot(1,2,1)  plt.plot(history.history['accuracy'], label='Train Acc')  plt.plot(history.history['val\_accuracy'], label='Val Acc')  plt.title('Accuracy')  plt.xlabel('Epochs')  plt.ylabel('Accuracy')  plt.legend()  plt.subplot(1,2,2)  plt.plot(history.history['loss'], label='Train Loss')  plt.plot(history.history['val\_loss'], label='Val Loss')  plt.title('Loss')  plt.xlabel('Epochs')  plt.ylabel('Loss')  plt.legend()  plt.tight\_layout()  plt.show() |
| **Statement 35**  **Build a Deep Neural Network for Fashion MNIST Classification**   * **Load Fashion MNIST dataset** * **Preprocess the data using standardization** * **Define a feed-forward neural network with 3 Dense layers** * **Use RMSprop optimizer and categorical crossentropy loss** * **Train the model for 15 epochs and evaluate performance** * **Plot the training and validation curves**   import tensorflow as tf  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense, Flatten  from tensorflow.keras.optimizers import RMSprop  from tensorflow.keras.utils import to\_categorical  import matplotlib.pyplot as plt  # Load Fashion MNIST dataset  (X\_train, y\_train), (X\_test, y\_test) = tf.keras.datasets.fashion\_mnist.load\_data()  # Standardization (mean=0, std=1)  X\_train = X\_train.astype('float32') / 255.0  X\_test = X\_test.astype('float32') / 255.0  # One-hot encoding of labels  y\_train\_cat = to\_categorical(y\_train, 10)  y\_test\_cat = to\_categorical(y\_test, 10)  # Define the model  model = Sequential([  Flatten(input\_shape=(28, 28)),  Dense(128, activation='relu'),  Dense(64, activation='relu'),  Dense(10, activation='softmax')  ])  # Compile the model  model.compile(  optimizer=RMSprop(learning\_rate=0.001),  loss='categorical\_crossentropy',  metrics=['accuracy']  )  # Train the model  history = model.fit(  X\_train, y\_train\_cat,  validation\_data=(X\_test, y\_test\_cat),  epochs=15,  batch\_size=32,  verbose=1  )  # Evaluate the model  loss, accuracy = model.evaluate(X\_test, y\_test\_cat)  print(f"\nTest Accuracy: {accuracy:.4f}")  print(f"Test Loss: {loss:.4f}")  # Plot training and validation curves  plt.figure(figsize=(12,5))  # Accuracy plot  plt.subplot(1,2,1)  plt.plot(history.history['accuracy'], label='Train Acc')  plt.plot(history.history['val\_accuracy'], label='Val Acc')  plt.title('Training and Validation Accuracy')  plt.xlabel('Epochs')  plt.ylabel('Accuracy')  plt.legend()  # Loss plot  plt.subplot(1,2,2)  plt.plot(history.history['loss'], label='Train Loss')  plt.plot(history.history['val\_loss'], label='Val Loss')  plt.title('Training and Validation Loss')  plt.xlabel('Epochs')  plt.ylabel('Loss')  plt.legend()  plt.tight\_layout()  plt.show() |