**Implementation questions on neural networks**

**1.Implement an ANN to perform binary classification on the Breast Cancer dataset from sklearn**

from sklearn.datasets import load\_breast\_cancer

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

# Load and preprocess the data

data = load\_breast\_cancer()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data.data, data.target, test\_size=0.2, random\_state=42)

scaler = StandardScaler().fit(X\_train)

X\_train = scaler.transform(X\_train)

X\_test = scaler.transform(X\_test)

# Build the model

model = Sequential([

Dense(32, activation='relu', input\_shape=(X\_train.shape[1],)),

Dense(16, activation='relu'),

Dense(1, activation='sigmoid')

])

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Train the model

model.fit(X\_train, y\_train, epochs=10, validation\_data=(X\_test, y\_test))

**2. Build an ANN to predict house prices using the Boston Housing dataset**

from sklearn.datasets import load\_boston

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

# Load and preprocess the data

data = load\_boston()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data.data, data.target, test\_size=0.2, random\_state=42)

scaler = StandardScaler().fit(X\_train)

X\_train = scaler.transform(X\_train)

X\_test = scaler.transform(X\_test)

# Build the model

model = Sequential([

Dense(64, activation='relu', input\_shape=(X\_train.shape[1],)),

Dense(32, activation='relu'),

Dense(1) # Linear output for regression

])

model.compile(optimizer='adam', loss='mean\_squared\_error')

# Train the model

model.fit(X\_train, y\_train, epochs=10, validation\_data=(X\_test, y\_test))

**3. Add dropout regularization to prevent overfitting on the Fashion MNIST dataset**

import tensorflow as tf

from tensorflow.keras import layers, models

# Load Fashion MNIST dataset

(X\_train, y\_train), (X\_test, y\_test) = tf.keras.datasets.fashion\_mnist.load\_data()

X\_train, X\_test = X\_train / 255.0, X\_test / 255.0

# Build the model with Dropout

model = models.Sequential([

layers.Flatten(input\_shape=(28, 28)),

layers.Dense(128, activation='relu'),

layers.Dropout(0.5),

layers.Dense(64, activation='relu'),

layers.Dropout(0.5),

layers.Dense(10, activation='softmax')

])

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

# Train the model

model.fit(X\_train, y\_train, epochs=10, validation\_data=(X\_test, y\_test))

**4. Use early stopping to prevent overfitting on the California Housing Prices dataset**

from sklearn.datasets import fetch\_california\_housing

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from tensorflow.keras.callbacks import EarlyStopping

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

# Load the data

data = fetch\_california\_housing()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data.data, data.target, test\_size=0.2, random\_state=42)

# Standardize the data

scaler = StandardScaler().fit(X\_train)

X\_train = scaler.transform(X\_train)

X\_test = scaler.transform(X\_test)

# Build the model

model = Sequential([

Dense(128, activation='relu', input\_shape=(X\_train.shape[1],)),

Dense(64, activation='relu'),

Dense(1)])

model.compile(optimizer='adam', loss='mean\_squared\_error')

# Early stopping callback

early\_stopping = EarlyStopping(monitor='val\_loss', patience=3)

# Train the model with early stopping

model.fit(X\_train, y\_train, epochs=50, validation\_data=(X\_test, y\_test), callbacks=[early\_stopping])

**5. Implement a CNN to classify handwritten digits from the MNIST dataset**

import tensorflow as tf

from tensorflow.keras import layers, models

# Load MNIST dataset

(X\_train, y\_train), (X\_test, y\_test) = tf.keras.datasets.mnist.load\_data()

X\_train, X\_test = X\_train/ 255.0, X\_test/ 255.0 # Normalize and add channel dimension

# Build CNN model

model = models.Sequential([

layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)),

layers.MaxPooling2D((2, 2)),

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

layers.MaxPooling2D((2, 2)),

layers.Flatten(),

layers.Dense(64, activation='relu'),

layers.Dense(10, activation='softmax')

])

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

# Train the model

model.fit(X\_train, y\_train, epochs=5, validation\_data=(X\_test, y\_test))

**6. Implement a CNN to classify clothing items in the Fashion MNIST dataset**

import tensorflow as tf

from tensorflow.keras import layers, models

# Load Fashion MNIST dataset

(X\_train, y\_train), (X\_test, y\_test) = tf.keras.datasets.fashion\_mnist.load\_data()

X\_train, X\_test = X\_train[..., tf.newaxis] / 255.0, X\_test[..., tf.newaxis] / 255.0 # Normalize and add channel dimension

# Build CNN model

model = models.Sequential([

layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)),

layers.MaxPooling2D((2, 2)),

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

layers.MaxPooling2D((2, 2)),

layers.Flatten(),

layers.Dense(128, activation='relu'),

layers.Dense(10, activation='softmax')

])

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

# Train the model

model.fit(X\_train, y\_train, epochs=10, validation\_data=(X\_test, y\_test))

**7. Implement data augmentation techniques for improving CNN performance on the CIFAR-10 dataset**

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Load CIFAR-10 dataset

(X\_train, y\_train), (X\_test, y\_test) = tf.keras.datasets.cifar10.load\_data()

X\_train, X\_test = X\_train / 255.0, X\_test / 255.0 # Normalize data

# Create ImageDataGenerator for data augmentation

datagen = ImageDataGenerator(

rotation\_range=20,

width\_shift\_range=0.2,

height\_shift\_range=0.2,

horizontal\_flip=True)

datagen.fit(X\_train)

# Build CNN model

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.Flatten(),

layers.Dense(128, activation='relu'),

layers.Dense(10, activation='softmax')])

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

# Train the model with augmented data

model.fit(datagen.flow(X\_train, y\_train, batch\_size=32), epochs=10, validation\_data=(X\_test, y\_test))

**8. Implement a basic RNN for classifying sequences of integers using synthetic sequential data**

import numpy as np

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import SimpleRNN, Dense

# Create synthetic sequential data

X\_train = np.random.random((1000, 10, 1)) # 1000 samples, 10 timesteps, 1 feature

y\_train = np.random.randint(2, size=1000) # Binary labels

# Build RNN model

model = Sequential([

SimpleRNN(32, activation='relu', input\_shape=(10, 1)),

Dense(1, activation='sigmoid')

])

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Train the model

model.fit(X\_train, y\_train, epochs=10)

**9. Use an LSTM-based model to perform sentiment analysis on the IMDB movie reviews dataset**

from tensorflow.keras.datasets import imdb

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Embedding

# Load IMDB dataset

(X\_train, y\_train), (X\_test, y\_test) = imdb.load\_data(num\_words=10000)

# Preprocess data

max\_len = 200

X\_train = pad\_sequences(X\_train, maxlen=max\_len)

X\_test = pad\_sequences(X\_test, maxlen=max\_len)

# Build LSTM model

model = Sequential([

Embedding(10000, 128, input\_length=max\_len),

LSTM(64),

Dense(1, activation='sigmoid')

])

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Train the model

model.fit(X\_train, y\_train, epochs=5, validation\_data=(X\_test, y\_test))

**10. Train an RNN to model the probability of the next word in a sequence based on the previous words**

import numpy as np

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import SimpleRNN, Dense

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad\_sequences

# Simulated text data

text = ["this is an example of text", "language modeling with RNN", "how RNN learns sequences"]

# Tokenize the text

tokenizer = Tokenizer()

tokenizer.fit\_on\_texts(text)

sequences = tokenizer.texts\_to\_sequences(text)

X\_data = pad\_sequences(sequences, maxlen=5) # Padding sequences

# Prepare target data (next word prediction)

y\_data = np.random.randint(0, len(tokenizer.word\_index) + 1, size=(len(text), 1))

# Build RNN model

model = Sequential([

SimpleRNN(50, activation='relu', input\_shape=(X\_data.shape[1], 1)),

**11. Implement an ANN to predict a continuous output value (regression task)**

import numpy as np

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

# Create synthetic regression data

X\_train = np.random.rand(1000, 10) # 1000 samples, 10 features

y\_train = np.random.rand(1000, 1) # Continuous labels

# Build ANN model

model = Sequential([

Dense(64, activation='relu', input\_shape=(10,)),

Dense(32, activation='relu'),

Dense(1) # No activation for regression output

])

model.compile(optimizer='adam', loss='mean\_squared\_error')

# Train the model

model.fit(X\_train, y\_train, epochs=10)

**12. Predict a continuous value (e.g., age) based on an image using CNN**

import numpy as np

from tensorflow.keras import layers, models

# Create synthetic image data and continuous labels

X\_train = np.random.rand(1000, 32, 32, 3) # 1000 images, 32x32 size, 3 channels

y\_train = np.random.rand(1000, 1) # Continuous values (e.g., age)

# Build CNN model

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.Flatten(),

layers.Dense(64, activation='relu'),

layers.Dense(1) # No activation for regression output

])

model.compile(optimizer='adam', loss='mean\_squared\_error')

# Train the model

model.fit(X\_train, y\_train, epochs=10)

**13. How to build model architecture of Neural Networks in tensorflow**

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

# Build a simple feedforward neural network (ANN)

model = Sequential([

Dense(64, activation='relu', input\_shape=(20,)), # 1st hidden layer with 64 neurons

Dense(32, activation='relu'), # 2nd hidden layer with 32 neurons

Dense(1, activation='sigmoid') # Output layer with 1 neuron (for binary classification)

])

**14. How to train Neural Networks in tensorflow**

# Sample training data

import numpy as np

X\_train = np.random.rand(1000, 20) # 1000 samples, 20 features

y\_train = np.random.randint(2, size=(1000, 1)) # Binary labels (0 or 1)

# Train the model

model.fit(X\_train, y\_train, epochs=10, batch\_size=32)

**15. How to compile Neural Networks in tensorflow**

model.compile(optimizer='adam', # Optimizer

loss='binary\_crossentropy', # Loss function for binary classification

metrics=['accuracy']) # Metric to monitor

* optimizer='adam': Uses the Adam optimizer, which is commonly used and performs well across different tasks.
* loss='binary\_crossentropy': Appropriate for binary classification problems.
* metrics=['accuracy']: The model will monitor accuracy during training.

**16. How to add callbacks to stop training when the models performance stops improving?**

from tensorflow.keras.callbacks import EarlyStopping

early\_stop = EarlyStopping(monitor='val\_loss', patience=3)

model.fit(X\_train, y\_train, epochs=50, batch\_size=32, validation\_data=(X\_val, y\_val), callbacks=[early\_stop])

**17. Implement a code for simple feedforward neural network in ANN**

# Step 1: Import necessary libraries

import numpy as np

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

# Step 2: Create synthetic training data

# 1000 samples, 20 features, binary labels (0 or 1)

X\_train = np.random.rand(1000, 20) # Random data: 1000 samples, 20 features

y\_train = np.random.randint(2, size=(1000, 1)) # Binary labels (0 or 1)

# Step 3: Define the ANN model

# Sequential model: one layer after another

model = Sequential([

Dense(64, activation='relu', input\_shape=(20,)), # First hidden layer with 64 neurons

Dense(32, activation='relu'), # Second hidden layer with 32 neurons

Dense(1, activation='sigmoid') # Output layer with 1 neuron for binary classification

])

# Step 4: Compile the model

# Adam optimizer, binary cross-entropy loss, and accuracy as a metric

model.compile(optimizer='adam',

loss='binary\_crossentropy',

metrics=['accuracy'])

# Step 5: Train the model

# Train the model for 10 epochs with a batch size of 32

model.fit(X\_train, y\_train, epochs=10, batch\_size=32)

# Step 6: Evaluate the model on the same data (for simplicity)

loss, accuracy = model.evaluate(X\_train, y\_train)

print(f"Training Accuracy: {accuracy:.4f}")

**18. Implement a code to train a neural network with custom loss function**

import tensorflow as tf

def custom\_loss(y\_true, y\_pred):

return tf.reduce\_mean(tf.square(y\_true - y\_pred))

model = Sequential([

Dense(64, activation='relu', input\_shape=(10,)),

Dense(1)

])

model.compile(optimizer='adam', loss=custom\_loss)

# Example usage

# model.fit(X\_train, y\_train, epochs=10)