IMPLEMENTATION OF NEURAL NETWORKS :

1.Multiclass Classification on MNIST using ANN?

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 data

# Build ANN model

model = models.Sequential([

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

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

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=10, validation\_data=(X\_test, y\_test))

2.How to launch tensorboard from the terminal?

tensorboard --logdir=logs/fit.

3.How to compile the model?

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

· **Optimizer**: Controls the learning rate and the method of updating model weights. Common options include:

* adam: Adaptive learning rate optimization algorithm.
* sgd: Stochastic Gradient Descent.
* rmsprop: Root Mean Square Propagation.

· **Loss Function**: Indicates how well the model's predictions match the actual outcomes. Options include:

* sparse\_categorical\_crossentropy: Used for multi-class classification problems with integer labels.
* categorical\_crossentropy: Used for multi-class classification problems with one-hot encoded labels.
* binary\_crossentropy: Used for binary classification problems.

· **Metrics**: Used to evaluate the model's performance during training and testing. Common metrics include:

* accuracy: The proportion of correctly predicted samples.
* precision: The ratio of true positives to the sum of true and false positives.
* recall: The ratio of true positives to the sum of true positives and false negatives.

4.Use transfer learning to classify images from a custom dataset using a pretrained VGG16 model?

from tensorflow.keras.applications import VGG16

from tensorflow.keras import layers, models

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Load pre-trained VGG16 model

base\_model = VGG16(weights='imagenet', include\_top=False, input\_shape=(150, 150, 3))

# Freeze the base model

base\_model.trainable = False

# Add custom top layers

model = models.Sequential([

base\_model,

layers.Flatten(),

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

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

])

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

# Load custom dataset (simulated here as CIFAR-10)

(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

# Train the model

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

1. Predict student performance using features like study time, health, and absences on the Student Performance dataset?.

import pandas as pd

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 the dataset

df = pd.read\_csv('student-mat.csv', sep=';')

X = df[['studytime', 'failures', 'absences']].values

y = df['G3'].values # Final grade

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, 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 ANN model

model = Sequential([

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

Dense(16, activation='relu'),

Dense(1)

])

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))

1. 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)),

1. Build a simple ANN to classify data into two categories using synthetic data.

import numpy as np

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

# Create synthetic binary classification data

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)

# Build ANN model

model = Sequential([

Dense(32, activation='relu', input\_shape=(20,)),

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)

1. Build an ANN to classify data into multiple categories (multiclass classification).

import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras import layers, models

# 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

# ImageDataGenerator for data augmentation

datagen = ImageDataGenerator(

rotation\_range=15,

width\_shift\_range=0.1,

height\_shift\_range=0.1,

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(64, 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))

1. Apply data augmentation to improve CNN performance on image classification.

import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras import layers, models

# 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

# ImageDataGenerator for data augmentation

datagen = ImageDataGenerator(

rotation\_range=15,

width\_shift\_range=0.1,

height\_shift\_range=0.1,

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(64, 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))

1. Use CNN to classify images from the CIFAR-10 dataset?

import tensorflow as tf

from tensorflow.keras import layers, models

# 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

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

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=10, validation\_data=(X\_test, y\_test))