**NANDHA ENGINEERING COLLEGE (AUTONOMOUS), ERODE**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

ACADEMIC YEAR : 2024-2025

ASSIGNMENT I(TEAM 14)

NAME:SIVA HARINE .S

REG NO :22CS091

CLASS : CSE-B

YEAR: III

SEM : V

COURSE NAME: DEEP LEARNING

|  |  |  |
| --- | --- | --- |
| **S.NO** | **QUESTIONS** | **MARKS** |
| 1 | Implement a neural network to predict equipment failures in a  manufacturing plant by analyzing historical sensor data, aiming  to reduce downtime and maintenance costs. | 10 |
| 2 | Create an autoencoder that enhances the resolution of low-  resolution images. The model should learn to generate high-  resolution images from their low-resolution counterparts. | 10 |

**Faculty signature Student signature**

1.Implement a neural network to predict equipment failures in a

manufacturing plant by analyzing historical sensor data, aiming

to reduce downtime and maintenance costs.

import pandas as pd

import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout

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

from sklearn.preprocessing import StandardScaler

import os  # For directory operations

# Define the path to save the CSV file

csv\_directory = 'D:/DLtask/data'

csv\_file\_path = os.path.join(csv\_directory, 'dummy\_sensor\_data.csv')

# Create the directory if it does not exist

if not os.path.exists(csv\_directory):

    os.makedirs(csv\_directory)

# Create a DataFrame with dummy data

np.random.seed(42)  # For reproducibility

data = {

    'sensor\_1': np.random.uniform(0.1, 1.0, 100),  # 100 random numbers between 0.1 and 1.0

    'sensor\_2': np.random.uniform(0.1, 1.0, 100),  # 100 random numbers between 0.1 and 1.0

    'failure': np.random.randint(0, 2, 100)         # 100 random binary values (0 or 1)

}

df = pd.DataFrame(data)

# Save the DataFrame as a CSV file

df.to\_csv(csv\_file\_path, index=False)

# Load the dataset

data = pd.read\_csv(csv\_file\_path)

# Split the data into features (X) and target label (y)

X = data.drop(columns=['failure'])  # Features (drop the label column)

y = data['failure']  # Labels (equipment failure)

# Split the data into train and test sets

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

# Standardize the feature data

scaler = StandardScaler()

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

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

# Define the neural network model

model = Sequential()

# Add input layer and first hidden layer

model.add(Dense(units=64, activation='relu', input\_dim=X\_train.shape[1]))

# Add more hidden layers with dropout for regularization

model.add(Dense(units=32, activation='relu'))

model.add(Dropout(0.3))

model.add(Dense(units=16, activation='relu'))

model.add(Dropout(0.2))

# Output layer with sigmoid activation for binary classification

model.add(Dense(units=1, activation='sigmoid'))

# Compile the model

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

# Train the model

history = model.fit(X\_train, y\_train, epochs=50, batch\_size=32, validation\_split=0.2)

# Evaluate the model on the test set

test\_loss, test\_acc = model.evaluate(X\_test, y\_test)

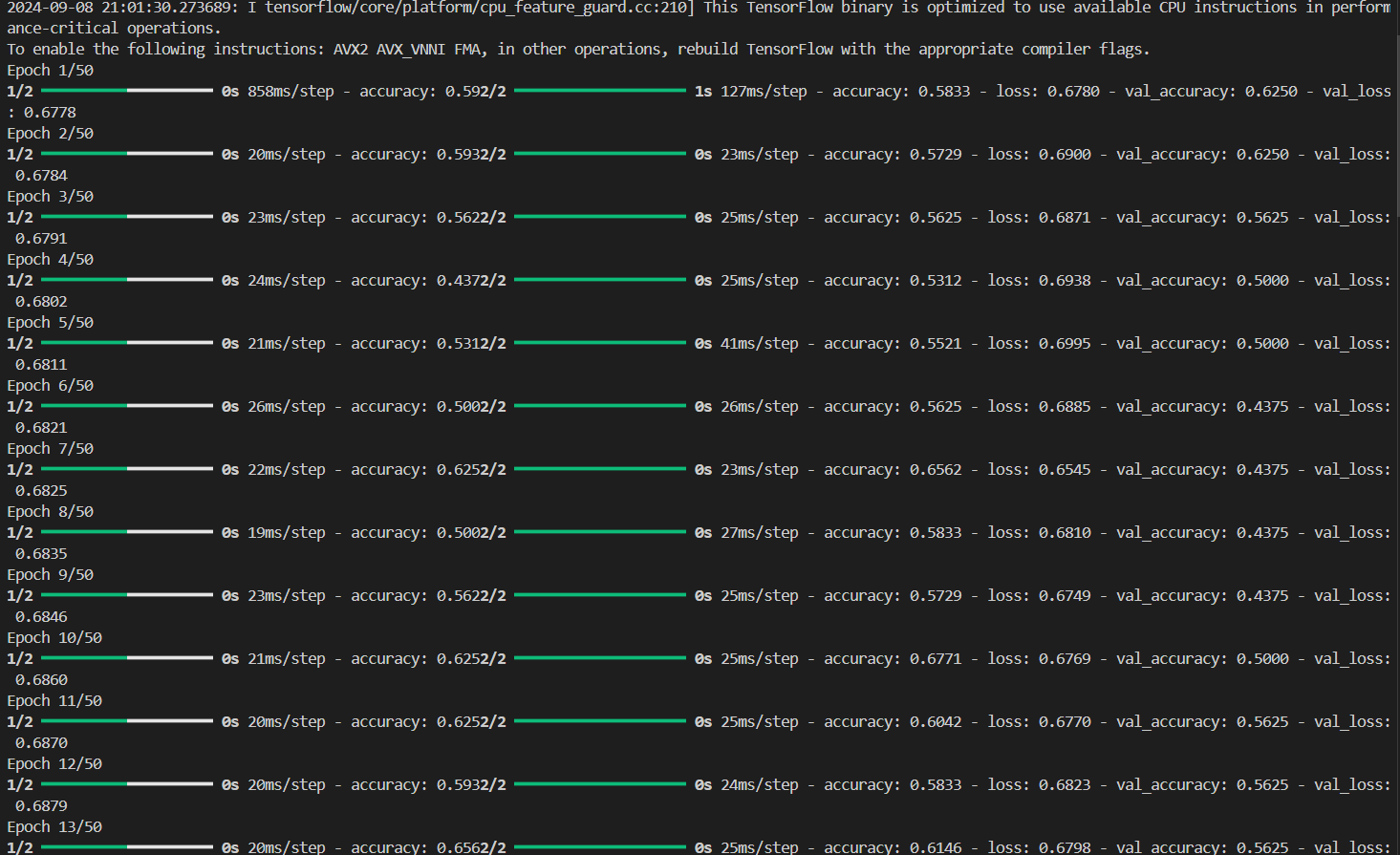
print(f'Test Accuracy: {test\_acc}')

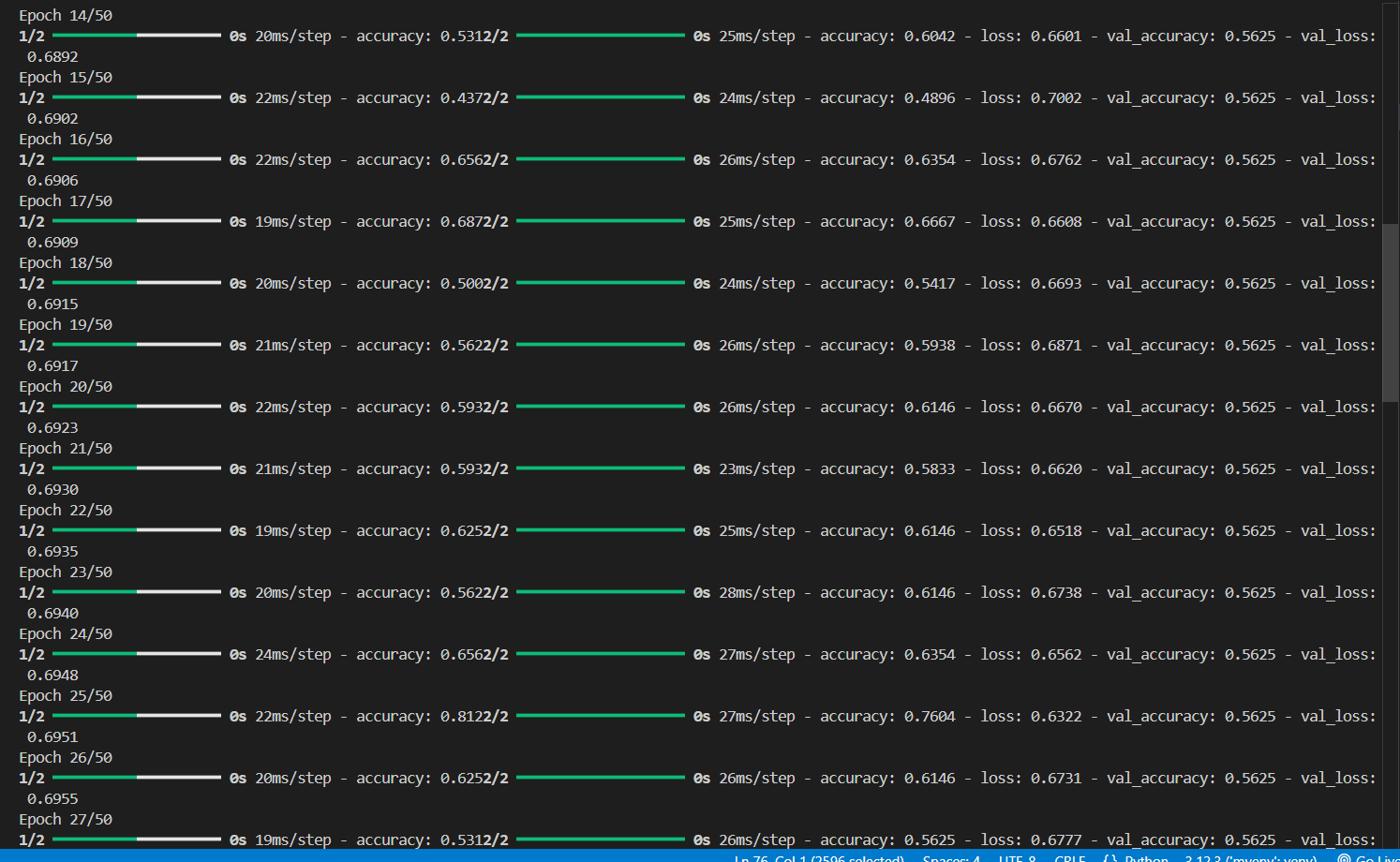
# Predict equipment failure on new sensor data (example)

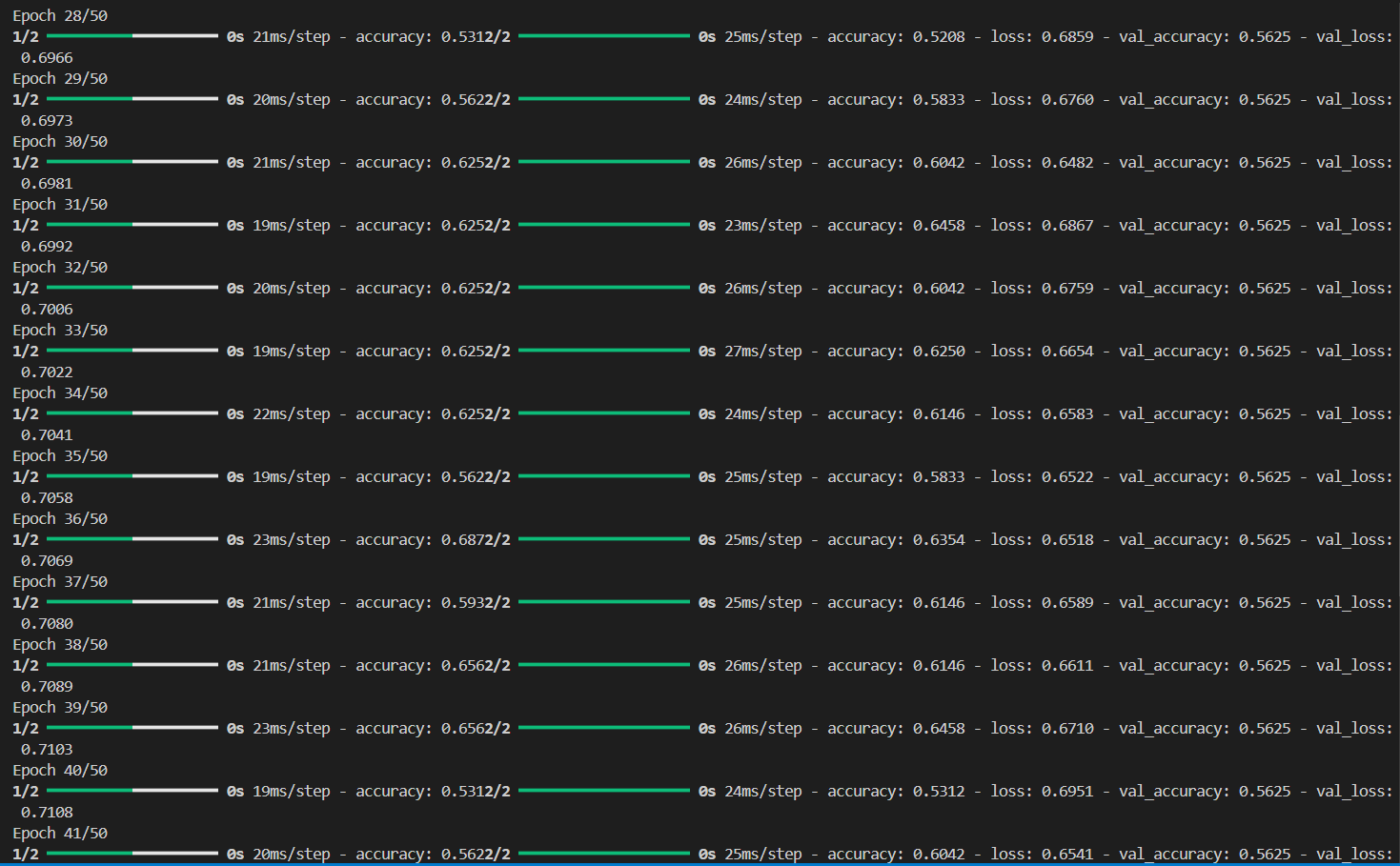
# new\_data = scaler.transform(new\_sensor\_data)  # Replace with actual new sensor data

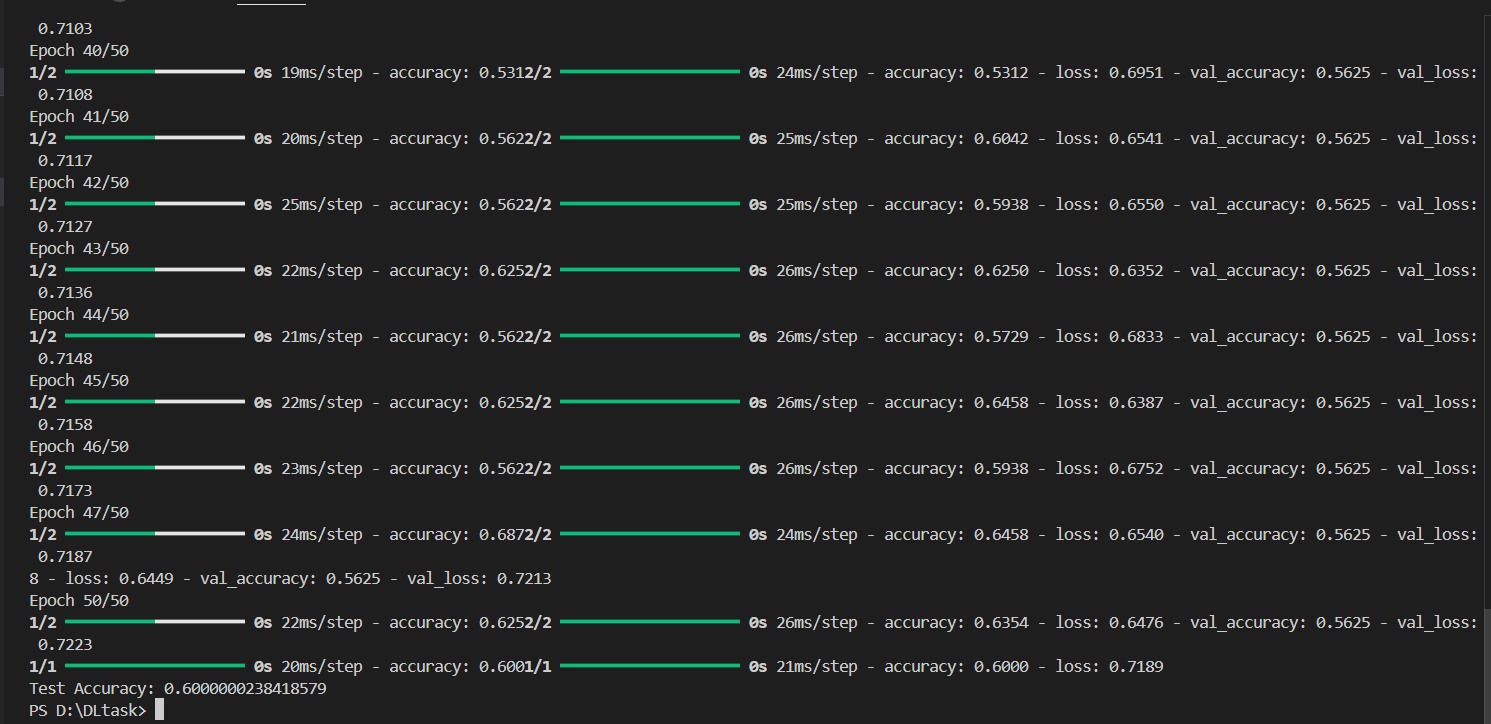
# predictions = model.predict(new\_data)

OUTPUT:









2.Create an autoencoder that enhances the resolution of low-

resolution images. The model should learn to generate high-

resolution images from their low-resolution counterparts.

import tensorflow as tf

from tensorflow.keras.layers import Conv2D, Conv2DTranspose, Input, ReLU, BatchNormalization

from tensorflow.keras.models import Model

# Autoencoder model

def build\_autoencoder(input\_shape):

    # Encoder

    inputs = Input(shape=input\_shape)

    # Encoding layers

    x = Conv2D(64, (3, 3), strides=2, padding='same')(inputs)

    x = BatchNormalization()(x)

    x = ReLU()(x)

    x = Conv2D(128, (3, 3), strides=2, padding='same')(x)

    x = BatchNormalization()(x)

    x = ReLU()(x)

    x = Conv2D(256, (3, 3), strides=2, padding='same')(x)

    x = BatchNormalization()(x)

    x = ReLU()(x)

    # Decoder

    x = Conv2DTranspose(256, (3, 3), strides=2, padding='same')(x)

    x = BatchNormalization()(x)

    x = ReLU()(x)

    x = Conv2DTranspose(128, (3, 3), strides=2, padding='same')(x)

    x = BatchNormalization()(x)

    x = ReLU()(x)

    x = Conv2DTranspose(64, (3, 3), strides=2, padding='same')(x)

    x = BatchNormalization()(x)

    x = ReLU()(x)

    # Output layer: Reconstruct high-resolution image

    outputs = Conv2DTranspose(3, (3, 3), activation='sigmoid', padding='same')(x)

    # Autoencoder model

    autoencoder = Model(inputs, outputs)

    return autoencoder

# Set input shape (e.g., low-resolution 32x32x3 image)

input\_shape = (32, 32, 3)

autoencoder = build\_autoencoder(input\_shape)

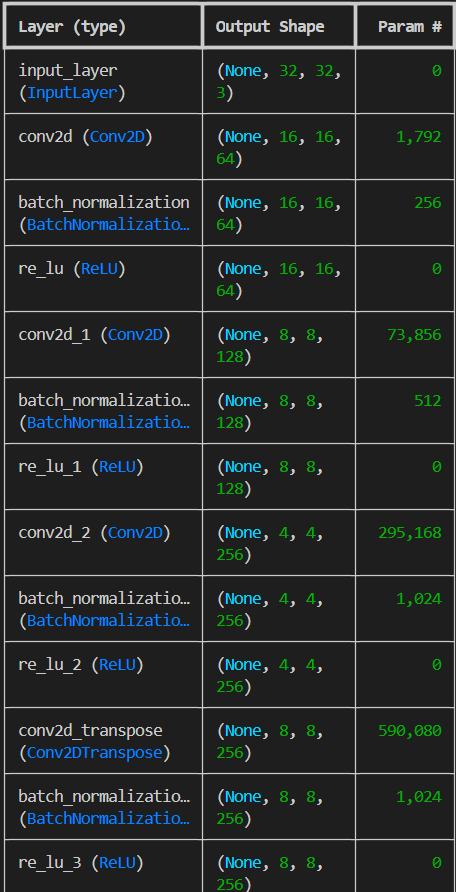
# Compile the model

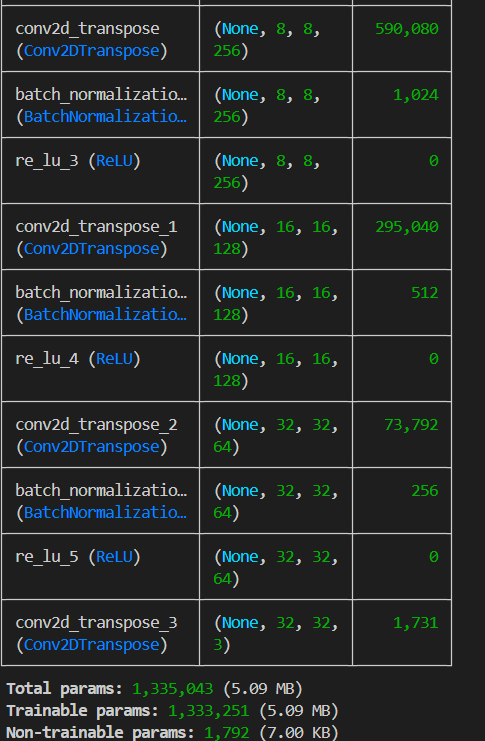
autoencoder.compile(optimizer='adam', loss='mse')

# Model summary

autoencoder.summary()

OUTPUT:





2.