**Experiment – 6**

**Aim:** Chatbot using bi-directional LSTMs

**Theory:** LSTM is a type of Recurrent Neural Network (RNN) that is used to process sequential data. Bi-directional LSTMs can capture dependencies in both forward and backward directions, making them useful in applications like chatbots where the meaning of a sentence can depend on the entire context.

**Source Code:**

 import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Sequential

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

# Dummy data (replace with real chatbot data)

input\_data = np.random.randint(1000, size=(100, 10))

output\_data = np.random.randint(2, size=(100, 1))

# Model

model = Sequential()

model.add(Embedding(input\_dim=1000, output\_dim=64))

model.add(Bidirectional(LSTM(64)))

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

# Compile

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

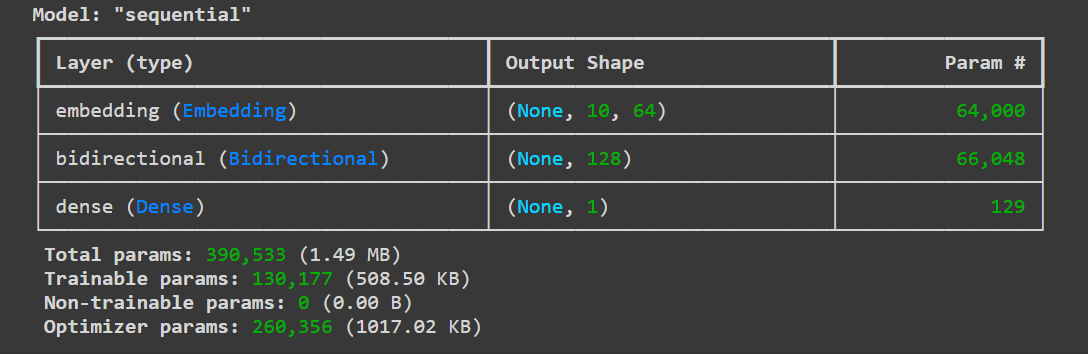
# Train

model.fit(input\_data, output\_data, epochs=10, batch\_size=32)

# Output (Model Summary)

model.summary()

**Output:**

****

**Experiment – 7**

**Aim:** Image classification on MNIST dataset (CNN model with fully connected layer)

**Theory:** CNNs are deep learning models typically used for image processing tasks. They use convolutional layers to capture spatial patterns in images. The fully connected layer at the end maps the extracted features to the output labels (digits 0-9 in MNIST).

**Source Code:**

 import tensorflow as tf

from tensorflow.keras.datasets import mnist

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Load MNIST dataset

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

# Preprocess data

x\_train = x\_train.reshape(-1, 28, 28, 1) / 255.0

x\_test = x\_test.reshape(-1, 28, 28, 1) / 255.0

# Build CNN model

model = Sequential()

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

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dense(10, activation='softmax'))

# Compile and train the model

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

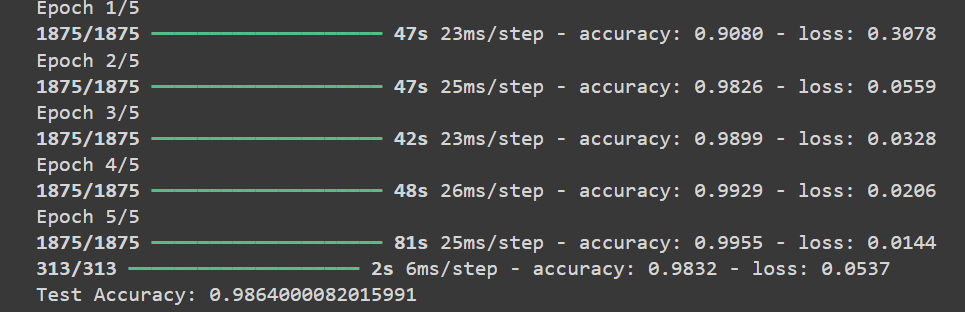
model.fit(x\_train, y\_train, epochs=5, batch\_size=32)

# Evaluate

loss, accuracy = model.evaluate(x\_test, y\_test)

print(f"Test Accuracy: {accuracy}")

**Output:**

****

**Experiment – 8**

**Aim:** Train a sentiment analysis model on IMDB dataset, use RNN layers with LSTM/GRU

**Theory:** Sentiment analysis involves classifying text data into positive or negative sentiment. LSTM and GRU are variants of RNNs that can capture long-term dependencies in text sequences, making them suitable for this task.

**Source Code:**

import tensorflow as tf

from tensorflow.keras.datasets import imdb

from tensorflow.keras.preprocessing import sequence

from tensorflow.keras.models import Sequential

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

# Load IMDB dataset

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

# Preprocess data

x\_train = sequence.pad\_sequences(x\_train, maxlen=500)

x\_test = sequence.pad\_sequences(x\_test, maxlen=500)

# Build LSTM model

model = Sequential()

model.add(Embedding(input\_dim=10000, output\_dim=64, input\_length=500))

model.add(LSTM(64))

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

# Compile and train the model

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

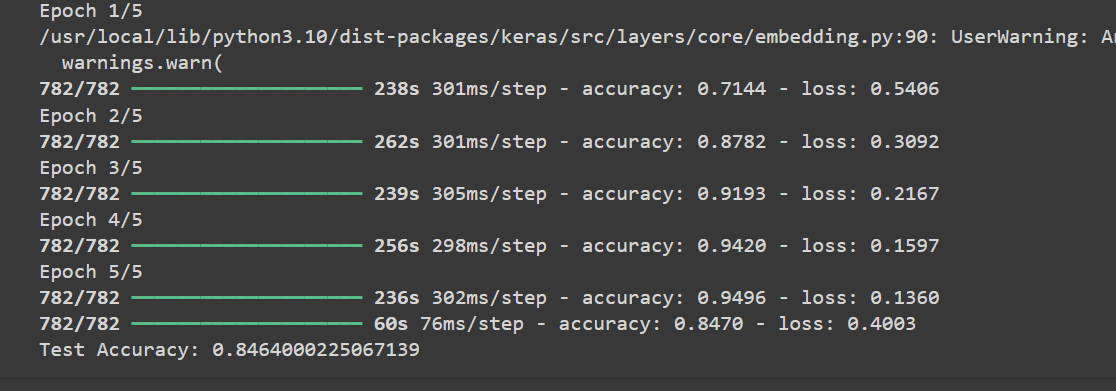
model.fit(x\_train, y\_train, epochs=5, batch\_size=32)

# Evaluate

loss, accuracy = model.evaluate(x\_test, y\_test)

print(f"Test Accuracy: {accuracy}")

**Output:**

****

**Experiment – 9**

**Aim:** Applying the Deep Learning Models in the field of Natural Language Processing

**Theory:** NLP involves processing and understanding human language. Deep learning models such as LSTMs and GRUs are used for tasks like text generation, translation, and sentiment analysis, as they handle sequential data efficiently.

**Source Code:**

 import tensorflow as tf

from tensorflow.keras.preprocessing.text import Tokenizer

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

from tensorflow.keras.models import Sequential

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

# Sample text data (replace with real NLP data)

texts = ["Deep learning is amazing", "Natural Language Processing is fun"]

tokenizer = Tokenizer(num\_words=1000)

tokenizer.fit\_on\_texts(texts)

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

# Pad sequences

data = pad\_sequences(sequences, maxlen=10)

# Build LSTM model for NLP

model = Sequential()

model.add(Embedding(input\_dim=1000, output\_dim=64, input\_length=10))

model.add(LSTM(64))

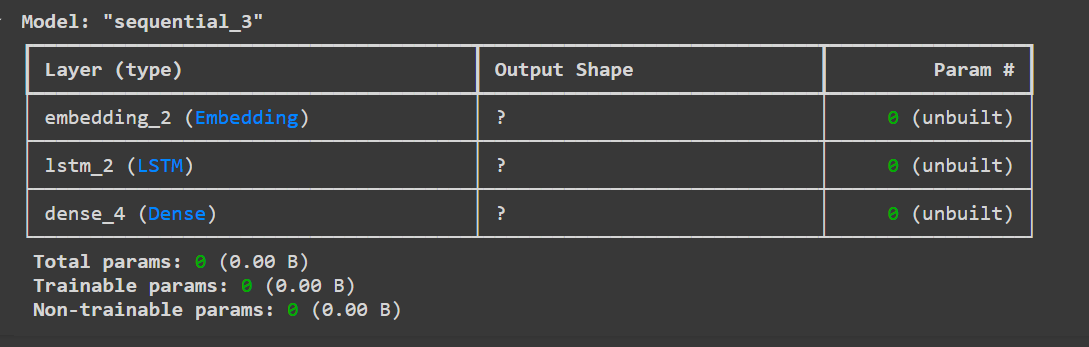
model.add(Dense(1, activation='sigmoid'))

# Compile and train the model

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

model.summary()

**Output:**

****

**Experiment – 10**

**Aim:** Applying the Convolution Neural Network on computer vision problems

**Theory:** CNNs have revolutionized computer vision by enabling accurate detection and classification of objects in images. They use convolutional layers to detect patterns like edges and shapes, followed by pooling and fully connected layers for decision-making.

**Source Code:**

import tensorflow as tf

from tensorflow.keras.datasets import cifar10

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Load CIFAR-10 dataset (for computer vision tasks)

(x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data()

# Preprocess data

x\_train = x\_train.astype('float32') / 255.0

x\_test = x\_test.astype('float32') / 255.0

# Build CNN model

model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dense(10, activation='softmax'))

# Compile and train the model

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

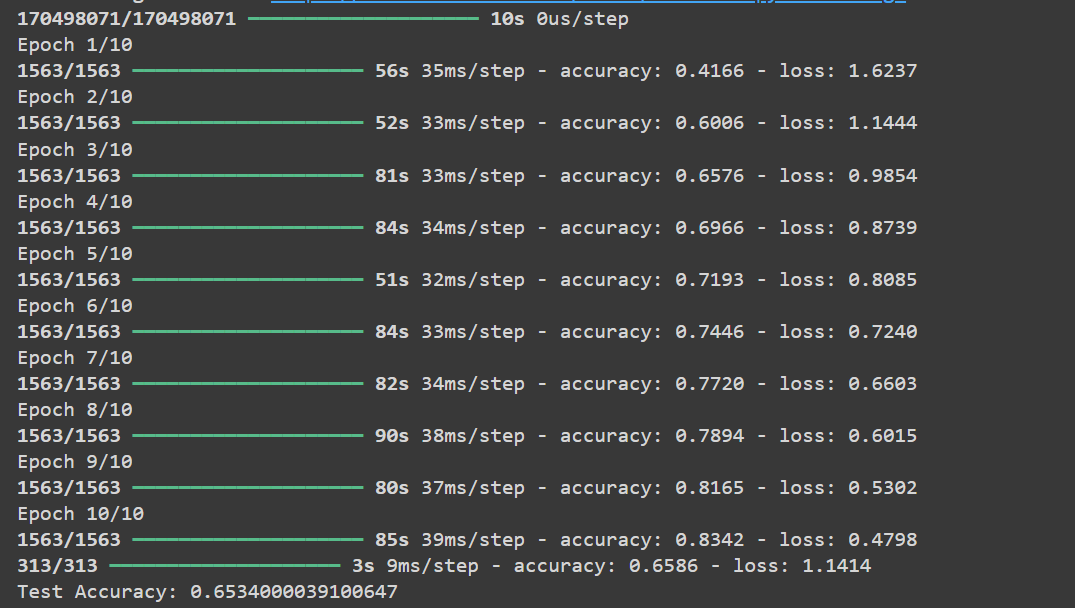
model.fit(x\_train, y\_train, epochs=10, batch\_size=32)

# Evaluate

loss, accuracy = model.evaluate(x\_test, y\_test)

print(f"Test Accuracy: {accuracy}")

**Output:**

****

**Experiment – 11**

**Aim:** Implement Deep Q Networks for CartPole problem where the agent has to balance a pole on a cart.

**Theory:** Deep Q-Networks combine Q-learning with deep neural networks to handle environments with large state spaces. In the CartPole problem, the goal is to balance a pole on a cart by applying forces to the left or right. The network learns the Q-value function to predict the best action for each state.

**Source Code:**

import gym

import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

# Initialize environment

env = gym.make('CartPole-v1')

state\_size = env.observation\_space.shape[0]

action\_size = env.action\_space.n

# Build Deep Q-Network

model = Sequential()

model.add(Dense(24, input\_dim=state\_size, activation='relu'))

model.add(Dense(24, activation='relu'))

model.add(Dense(action\_size, activation='linear'))

model.compile(loss='mse', optimizer=tf.keras.optimizers.Adam(learning\_rate=0.001))

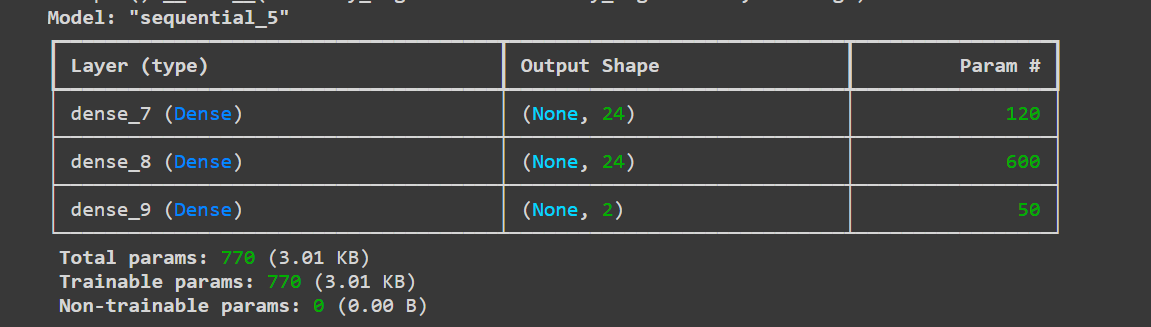
# Placeholder for training code: collecting experiences, updating Q-values, etc.

# This would involve implementing the experience replay and Q-learning update rule.

# Output: Model Summary

model.summary()

**Output:**

****

**Experiment – 12**

**Aim:** Demonstrate the application of transfer learning using Cartpole dataset and MountainCar dataset.

**Theory:** Transfer learning allows a model trained on one problem to be used in another related problem by fine-tuning. In this case, a model trained on CartPole can be adapted for the MountainCar problem, transferring knowledge about control systems between tasks.

**Source Code:**

import gym

import numpy as np

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

# Load CartPole environment and pre-train model (assuming pre-training is done)

env\_cartpole = gym.make('CartPole-v1')

state\_size = env\_cartpole.observation\_space.shape[0]

action\_size = env\_cartpole.action\_space.n

# Create a model for CartPole

model\_cartpole = Sequential()

model\_cartpole.add(Dense(24, input\_dim=state\_size, activation='relu'))

model\_cartpole.add(Dense(24, activation='relu'))

model\_cartpole.add(Dense(action\_size, activation='linear'))

# Now apply the same architecture to MountainCar

env\_mountaincar = gym.make('MountainCar-v0')

state\_size\_mc = env\_mountaincar.observation\_space.shape[0]

action\_size\_mc = env\_mountaincar.action\_space.n

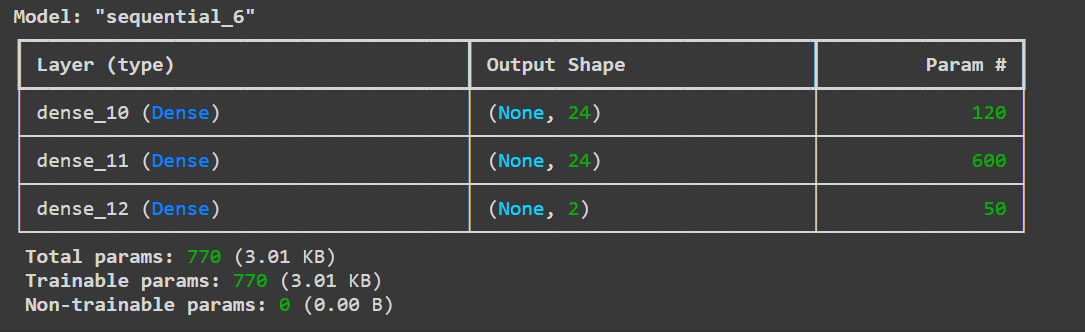
# Fine-tune CartPole model to MountainCar

model\_cartpole.compile(loss='mse', optimizer='adam')

# Transfer learning can involve freezing layers or retraining certain layers on MountainCar data.

model\_cartpole.summary()

**Output:**

****

**Experiment – 13**

**Aim:** Choose any corpus available on the internet freely. For the corpus, for each document, count how many times each stop word occurs and find out which are the most frequently occurring stop words. Further, calculate the term frequency and inverse document frequency as the motivation behind this is basically to find out how important a document is to a given query.

**Theory:** Stop words are common words like "the", "is", "in", which are filtered out in text processing. TF-IDF measures the importance of a word in a document relative to a corpus. It reduces the weight of common words (high frequency) and highlights significant ones (low frequency across documents).

**Source Code:**

 import nltk

nltk.download('stopwords')

from sklearn.feature\_extraction.text import TfidfVectorizer

from nltk.corpus import stopwords

# Sample corpus

corpus = [

    "The brown fox jumps over the lazy dog.",

    "The quick brown fox jumped over the lazy dog."

]

# Stop words

stop\_words = list(stopwords.words('english'))

# Count stop word occurrences

stop\_word\_count = {word: 0 for word in stop\_words}

for doc in corpus:

    for word in doc.lower().split():

        if word in stop\_word\_count:

            stop\_word\_count[word] += 1

# TF-IDF calculation

vectorizer = TfidfVectorizer(stop\_words=stop\_words)

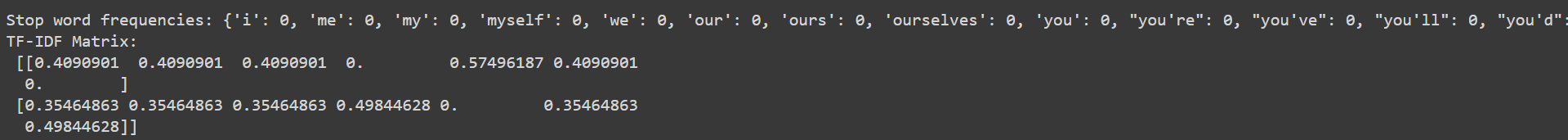
tfidf\_matrix = vectorizer.fit\_transform(corpus)

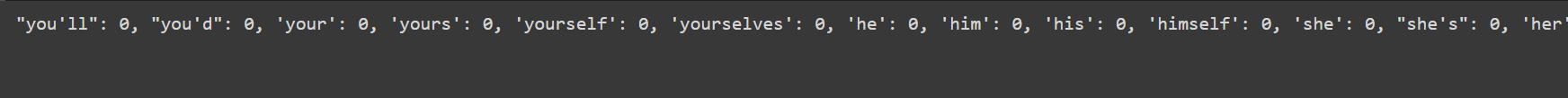
# Output stop word frequencies and TF-IDF scores

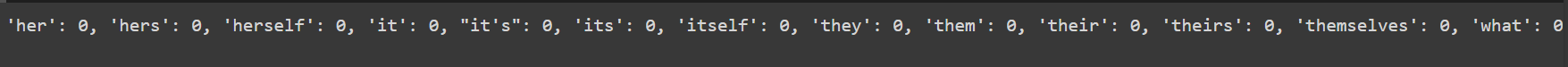
print("Stop word frequencies:", stop\_word\_count)

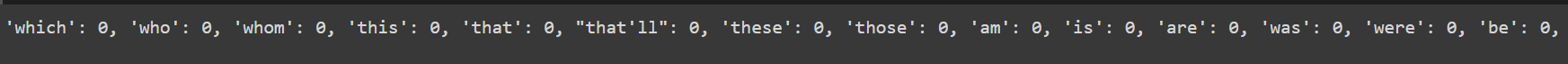
print("TF-IDF Matrix:\n", tfidf\_matrix.toarray())

**Output:**

****

****

****

****

**Experiment – 14**

**Aim:** Write the python code to develop Spam Filter using NLP.

**Theory:** A spam filter classifies emails or text messages as either spam or not spam. NLP techniques such as tokenization, vectorization, and machine learning models (e.g., Naive Bayes, SVM) are used to distinguish between normal and spam content based on text patterns.

**Source Code:**

 from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.naive\_bayes import MultinomialNB

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

# Sample dataset

emails = ["Hey, wanna grab lunch?", "Win $1000 now!", "Your package has shipped.", "Limited time offer, claim your prize!"]

labels = [0, 1, 0, 1]  # 0 = Not Spam, 1 = Spam

# Vectorize emails

vectorizer = CountVectorizer()

X = vectorizer.fit\_transform(emails)

# Split into train and test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, labels, test\_size=0.2)

# Train Naive Bayes classifier

model = MultinomialNB()

model.fit(X\_train, y\_train)

# Evaluate the model

accuracy = model.score(X\_test, y\_test)

print(f"Spam Filter Accuracy: {accuracy}")

**Output:**

****

**Experiment – 15**

**Aim:** Demonstrate any one application of generative adversarial network (GAN).

**Theory:** GANs consist of two neural networks, a generator and a discriminator, which compete in a zero-sum game. The generator creates synthetic data, while the discriminator distinguishes between real and generated data. GANs are used in image generation, style transfer, and more.

**Source Code:**

 import tensorflow as tf

from tensorflow.keras.layers import Dense, LeakyReLU, Reshape, Flatten

from tensorflow.keras.models import Sequential

import numpy as np

# Generator

def build\_generator():

    model = Sequential()

    model.add(Dense(128, input\_dim=100))

    model.add(LeakyReLU(alpha=0.01))

    model.add(Dense(784, activation='tanh'))

    model.add(Reshape((28, 28, 1)))

    return model

# Discriminator

def build\_discriminator():

    model = Sequential()

    model.add(Flatten(input\_shape=(28, 28, 1)))

    model.add(Dense(128))

    model.add(LeakyReLU(alpha=0.01))

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

    return model

# Create GAN

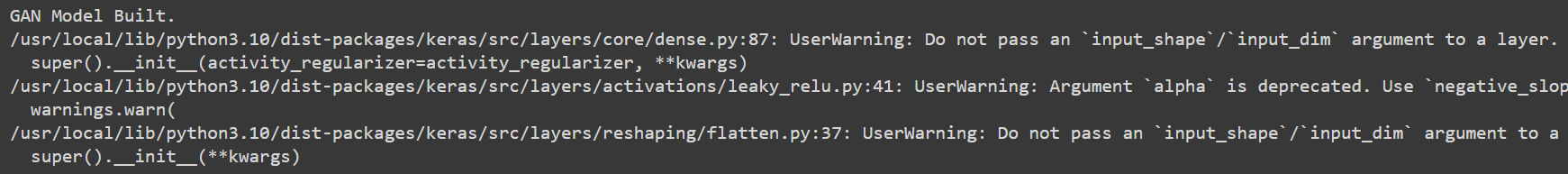
generator = build\_generator()

discriminator = build\_discriminator()

# Compile GAN (placeholder for real training process)

print("GAN Model Built.")

**Output:**

****