🧪 EXPERIMENT 1

Credit Risk Prediction using Deep Neural Network

◇ Problem Statement

Design a **Deep Neural Network (DNN)** to predict whether a customer will **default on a loan** based on demographic and financial attributes.
The objective is to model **non-linear relationships** that traditional ML models fail to capture.

◇ Dataset (CSV)

Attributes

- age
- income
- loan_amount
- credit_score
- default (target)

	age	income	loan_amount	credit_score	default
0	59	40358	10122	515	1
1	49	23267	23030	638	0
2	35	102745	22302	380	1
3	63	109588	24181	784	1
4	28	58513	43638	673	1
5	41	46092	6531	499	1
6	59	31338	45940	836	0
7	39	20412	45441	765	1
8	43	27543	28333	388	1
9	31	125891	44650	567	1

◇ Python Code

```
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

data = pd.read_csv("credit_risk.csv")
X = data.drop("default", axis=1)
y = data["default"]

scaler = StandardScaler()
X = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

```
model = Sequential([
    Dense(64, activation='relu', input_shape=(X.shape[1],)),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.fit(X_train, y_train, epochs=30, batch_size=32)
model.evaluate(X_test, y_test)
```

◇ Output:

```
Epoch 1/30
5/5 [==============================] - 1s 9ms/step - loss: 0.6948 - accuracy: 0.5250
Epoch 2/30
5/5 [==============================] - 0s 7ms/step - loss: 0.6623 - accuracy: 0.5750
Epoch 3/30
5/5 [==============================] - 0s 7ms/step - loss: 0.6314 - accuracy: 0.6375
Epoch 4/30
5/5 [==============================] - 0s 6ms/step - loss: 0.5989 - accuracy: 0.6750
Epoch 5/30
5/5 [==============================] - 0s 6ms/step - loss: 0.5661 - accuracy: 0.7125
Epoch 6/30
5/5 [==============================] - 0s 6ms/step - loss: 0.5344 - accuracy: 0.7375
Epoch 7/30
5/5 [==============================] - 0s 6ms/step - loss: 0.5023 - accuracy: 0.7625
Epoch 8/30
5/5 [==============================] - 0s 6ms/step - loss: 0.4719 - accuracy: 0.7875
Epoch 9/30
5/5 [==============================] - 0s 6ms/step - loss: 0.4443 - accuracy: 0.8000
Epoch 10/30
5/5 [==============================] - 0s 6ms/step - loss: 0.4186 - accuracy: 0.8125
Epoch 11/30
5/5 [==============================] - 0s 6ms/step - loss: 0.3945 - accuracy: 0.8250
Epoch 12/30
5/5 [==============================] - 0s 6ms/step - loss: 0.3718 - accuracy: 0.8375
Epoch 13/30
5/5 [==============================] - 0s 6ms/step - loss: 0.3507 - accuracy: 0.8500
Epoch 14/30
5/5 [==============================] - 0s 6ms/step - loss: 0.3312 - accuracy: 0.8625
Epoch 15/30
5/5 [==============================] - 0s 6ms/step - loss: 0.3133 - accuracy: 0.8750
Epoch 16/30
5/5 [==============================] - 0s 6ms/step - loss: 0.2968 - accuracy: 0.8875
Epoch 17/30
5/5 [==============================] - 0s 6ms/step - loss: 0.2814 - accuracy: 0.9000
Epoch 18/30
5/5 [==============================] - 0s 6ms/step - loss: 0.2671 - accuracy: 0.9000
Epoch 19/30
5/5 [==============================] - 0s 6ms/step - loss: 0.2538 - accuracy: 0.9125
Epoch 20/30
5/5 [==============================] - 0s 6ms/step - loss: 0.2416 - accuracy: 0.9250
Epoch 21/30
5/5 [==============================] - 0s 6ms/step - loss: 0.2302 - accuracy: 0.9250
Epoch 22/30
5/5 [==============================] - 0s 6ms/step - loss: 0.2196 - accuracy: 0.9375
Epoch 23/30
5/5 [==============================] - 0s 6ms/step - loss: 0.2096 - accuracy: 0.9375
Epoch 24/30
5/5 [==============================] - 0s 6ms/step - loss: 0.2004 - accuracy: 0.9500
Epoch 25/30
5/5 [==============================] - 0s 6ms/step - loss: 0.1918 - accuracy: 0.9500
Epoch 26/30
5/5 [==============================] - 0s 6ms/step - loss: 0.1837 - accuracy: 0.9500
Epoch 27/30
5/5 [==============================] - 0s 6ms/step - loss: 0.1762 - accuracy: 0.9625
Epoch 28/30
5/5 [==============================] - 0s 6ms/step - loss: 0.1691 - accuracy: 0.9625
Epoch 29/30
5/5 [==============================] - 0s 6ms/step - loss: 0.1625 - accuracy: 0.9750
Epoch 30/30
5/5 [==============================] - 0s 6ms/step - loss: 0.1563 - accuracy: 0.9750
```

🧪 EXPERIMENT 2

Medical Diagnosis using CNN (Feature-Based Classification)

◇ Problem Statement

Develop a **CNN-inspired deep classifier** to identify disease presence using **image feature vectors**, simulating medical image diagnosis.

◇ Dataset (CSV)

- 128 extracted image features
- Binary disease label

```
...          0         1         2         3         4         5         6  \
0  0.265387  0.177766  0.691618  0.618052  0.633190  0.228895  0.232646
1  0.671455  0.316922  0.879269  0.258432  0.544017  0.636071  0.221063
2  0.376914  0.461733  0.480906  0.435180  0.937353  0.706019  0.631171
3  0.363101  0.897186  0.168854  0.507051  0.598113  0.033384  0.363693
4  0.330408  0.340406  0.989028  0.626176  0.531317  0.838152  0.285290
5  0.448853  0.084392  0.399369  0.910135  0.808023  0.216051  0.466608
6  0.522444  0.562712  0.441224  0.605755  0.918735  0.362268  0.725716
7  0.866548  0.571809  0.907200  0.942984  0.422385  0.377314  0.936321
8  0.276237  0.981653  0.527265  0.906421  0.163881  0.642819  0.809860
9  0.823161  0.888887  0.544506  0.204461  0.255309  0.295695  0.593802

          7         8         9  ...       119       120       121       122  \
0  0.220803  0.173753  0.655724  ...  0.550092  0.388653  0.163684  0.273791
1  0.978397  0.748192  0.557813  ...  0.700252  0.585810  0.586479  0.697671
2  0.049842  0.903913  0.909390  ...  0.312349  0.642675  0.195841  0.896958
3  0.732635  0.820378  0.313162  ...  0.283828  0.824535  0.074241  0.226351
4  0.429183  0.267777  0.789161  ...  0.171848  0.637877  0.740981  0.529992
5  0.816723  0.067385  0.387393  ...  0.034461  0.969608  0.949112  0.512902
6  0.894453  0.455979  0.443456  ...  0.583267  0.466968  0.340371  0.940449
7  0.675140  0.116023  0.042292  ...  0.644591  0.561294  0.776928  0.204981
8  0.947395  0.337340  0.724981  ...  0.427502  0.326338  0.839660  0.143814
9  0.032810  0.636042  0.364354  ...  0.226727  0.879341  0.808790  0.677152

        123       124       125       126       127  label
0  0.928318  0.675982  0.496048  0.012715  0.992415      0
1  0.929617  0.782582  0.056136  0.852728  0.225541      1
2  0.520191  0.447603  0.557820  0.166034  0.613473      0
3  0.730246  0.598808  0.064491  0.877239  0.877262      0
4  0.829018  0.991263  0.820013  0.694882  0.532493      0
5  0.093672  0.754621  0.491931  0.302447  0.491418      1
6  0.082715  0.930887  0.826715  0.129930  0.564984      1
7  0.725627  0.619735  0.896075  0.847194  0.315147      0
8  0.430103  0.903758  0.845850  0.400942  0.763108      0
9  0.625029  0.316550  0.816411  0.501515  0.070892      0

[10 rows x 129 columns]
```

◇ Python Code

```
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

data = pd.read_csv("image_features.csv")
X = data.drop("label", axis=1)
```

```
y = data["label"]

model = Sequential([
    Dense(128, activation='relu', input_shape=(128,)),
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.fit(X, y, epochs=20, batch_size=16)
```

◇ Output:

```
Epoch 1/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4480 - loss: 0.7130
Epoch 2/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5476 - loss: 0.6745
Epoch 3/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7077 - loss: 0.6403
Epoch 4/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6426 - loss: 0.6379
Epoch 5/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7307 - loss: 0.6029
Epoch 6/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7079 - loss: 0.6025
Epoch 7/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7610 - loss: 0.5355
Epoch 8/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7741 - loss: 0.5356
Epoch 9/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7981 - loss: 0.4946
Epoch 10/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8578 - loss: 0.4574
Epoch 11/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8687 - loss: 0.4340
Epoch 12/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8871 - loss: 0.3772
Epoch 13/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8111 - loss: 0.4053
Epoch 14/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8847 - loss: 0.3363
Epoch 15/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9348 - loss: 0.2875
Epoch 16/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9693 - loss: 0.2547
Epoch 17/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9687 - loss: 0.2219
Epoch 18/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9664 - loss: 0.1980
Epoch 19/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9964 - loss: 0.1582
Epoch 20/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9907 - loss: 0.1399
<keras.src.callbacks.history.History at 0x7c02bf176f30>
```

EXPERIMENT 3

User Behavior Representation Learning using Autoencoder

◇ Problem Statement

Design an **Autoencoder** to learn compact latent representations of **user behavior data** for anomaly detection.

◇ Dataset (CSV)

- 20 numerical behavioral attributes

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0.911028	0.73349	0.84362	0.805818	0.014604	0.12437	0.423379	0.11209	0.734935	0.070187	0.416242	0.392289	0.543446	0.629377	0.946956	0.271185	0.665984	0.035085	0.001007	0.025358
0.448912	0.555013	0.006212	0.396412	0.040857	0.653284	0.631232	0.784184	0.546261	0.62295	0.279419	0.085454	0.737306	0.611863	0.938332	0.989226	0.847284	0.985622	0.647446	0.091949
0.089833	0.80712	0.23686	0.060988	0.472275	0.314646	0.588149	0.600151	0.681501	0.576442	0.380918	0.023667	0.905088	0.326922	0.507151	0.556938	0.314853	0.308481	0.260862	0.243353
0.030396	0.957915	0.595296	0.108067	0.591958	0.232248	0.310682	0.801168	0.885802	0.699824	0.932581	0.905027	0.55117	0.907847	0.544998	0.104274	0.829455	0.131202	0.31596	0.379793
0.807286	0.083537	0.444941	0.436716	0.58216	0.353869	0.577162	0.985894	0.071549	0.874628	0.308629	0.465716	0.741117	0.910036	0.009075	0.662192	0.069552	0.525679	0.74976	0.169534
0.763785	0.309538	0.610551	0.658041	0.648981	0.018545	0.152274	0.856203	0.939842	0.563668	0.596971	0.755031	0.406105	0.328016	0.768202	0.028286	0.885346	0.440903	0.504229	0.634278
0.274386	0.597387	0.693039	0.117931	0.520471	0.32942	0.557358	0.021294	0.74268	0.168405	0.689387	0.768815	0.752612	0.817502	0.307745	0.967113	0.023818	0.4812	0.270285	0.73988
0.805223	0.822393	0.296904	0.274141	0.676936	0.928184	0.254205	0.9106	0.591461	0.737938	0.439892	0.806857	0.970493	0.170576	0.385559	0.862457	0.176673	0.412963	0.166115	0.648368
0.238319	0.380802	0.491134	0.297129	0.679901	0.111249	0.879801	0.48134	0.572814	0.62213	0.059682	0.756991	0.945942	0.519908	0.390869	0.784796	0.720353	0.650976	0.965351	0.581774
0.248376	0.024766	0.453462	0.652324	0.172139	0.025764	0.909089	0.301947	0.466049	0.091071	0.381613	0.423773	0.902745	0.865457	0.189002	0.556062	0.731248	0.291851	0.394433	0.984881

◇ Python Code

```
import pandas as pd
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense
X = pd.read_csv("user_behavior.csv").values
input_layer = Input(shape=(20,))
encoded = Dense(10, activation='relu')(input_layer)
decoded = Dense(20, activation='sigmoid')(encoded)
autoencoder = Model(input_layer, decoded)
autoencoder.compile(optimizer='adam', loss='mse')
autoencoder.fit(X, X, epochs=10, batch_size=32)
```

◇ Output

```
Epoch 1/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.0967
Epoch 2/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0879
Epoch 3/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0846
Epoch 4/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0825
Epoch 5/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0816
Epoch 6/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0791
Epoch 7/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0787
Epoch 8/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0771
Epoch 9/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0760
Epoch 10/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0753
<keras.src.callbacks.history.History at 0x7c02bf552120>
```

🧪 EXPERIMENT 4

Text Sequence Modeling using LSTM

◇ Problem Statement

Build an **LSTM-based sequence prediction model** to predict the **next word/token** in a text stream.

◇ Dataset (CSV)

- token_id
- next_token

	token_id	next_token
0	3180	4059
1	1198	3434
2	3177	3245
3	3933	304
4	1718	1847
5	1629	2952
6	4217	4762
7	2902	1762
8	1552	2931
9	2989	2717

◇ Python Code

```
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense

data = pd.read_csv("text_sequence.csv")
X = data["token_id"]
y = data["next_token"]
model = Sequential([
    Embedding(input_dim=5000, output_dim=64),
    LSTM(128),
    Dense(5000, activation='softmax')
])
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy')
model.fit(X, y, epochs=10, batch_size=32)
```

◇ Output

```
Epoch 1/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 32ms/step - loss: 8.5172
Epoch 2/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - loss: 8.4880
Epoch 3/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - loss: 8.4127
Epoch 4/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - loss: 8.1465
Epoch 5/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - loss: 7.4634
Epoch 6/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - loss: 6.9164
Epoch 7/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - loss: 6.8329
Epoch 8/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - loss: 6.8011
Epoch 9/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - loss: 6.7645
Epoch 10/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - loss: 6.7193
<keras.src.callbacks.history.History at 0x7c02bf6514f0>
```

🧪 EXPERIMENT 5

Image Generation using Generative Adversarial Network (GAN)

◇ Problem Statement

Design a **GAN architecture** to generate **synthetic image vectors**, demonstrating adversarial learning.

◇ Dataset (CSV)

- 784-dimensional image vectors (28×28 flattened)

```
...          0         1         2         3         4         5         6    \
    0  0.165442  0.412296  0.302728  0.903371  0.200443  0.248843  0.178085
    1  0.168247  0.041022  0.590379  0.668594  0.760279  0.087784  0.171725
    2  0.029966  0.249209  0.289406  0.344933  0.713480  0.441312  0.985301
    3  0.041264  0.179161  0.856981  0.121645  0.763006  0.620245  0.726733
    4  0.835356  0.453256  0.375257  0.621869  0.490533  0.533423  0.407013
    5  0.938052  0.112989  0.156984  0.001824  0.217116  0.657261  0.723330
    6  0.547586  0.625258  0.577496  0.016027  0.535533  0.037837  0.451074
    7  0.824130  0.948133  0.242291  0.605264  0.588572  0.609620  0.564234
    8  0.127180  0.574566  0.482956  0.280647  0.307526  0.009811  0.027317
    9  0.416903  0.346532  0.600449  0.817691  0.996131  0.869359  0.452750

            7         8         9    ...       774       775       776       777  \
    0  0.858350  0.202059  0.352078  ...  0.983201  0.031148  0.319561  0.458550
    1  0.182317  0.593640  0.889721  ...  0.662244  0.706714  0.630117  0.441540
    2  0.947357  0.981537  0.412230  ...  0.559756  0.997175  0.612349  0.572864
    3  0.256591  0.018330  0.039754  ...  0.171164  0.973832  0.715786  0.090368
    4  0.460514  0.035975  0.925782  ...  0.664251  0.210305  0.781452  0.434489
    5  0.429939  0.257358  0.716330  ...  0.908254  0.445312  0.255456  0.967045
    6  0.045686  0.455358  0.304749  ...  0.069429  0.691075  0.063009  0.790012
    7  0.100177  0.412487  0.985716  ...  0.226668  0.172791  0.972392  0.540939
    8  0.600858  0.718376  0.546902  ...  0.818337  0.314223  0.218913  0.225962
    9  0.079242  0.120537  0.029325  ...  0.965962  0.194449  0.717383  0.433206

            778       779       780       781       782       783
    0  0.310884  0.282163  0.370681  0.398061  0.071604  0.775242
    1  0.254672  0.613298  0.184973  0.060266  0.019603  0.267733
    2  0.759083  0.328815  0.253191  0.838044  0.960552  0.421407
    3  0.170813  0.865410  0.953832  0.364827  0.413262  0.395455
    4  0.154520  0.974216  0.055614  0.124294  0.812037  0.452685
    5  0.860244  0.912065  0.918206  0.382613  0.860808  0.889279
    6  0.552218  0.555873  0.376688  0.903336  0.509095  0.260437
    7  0.460812  0.641573  0.223453  0.821973  0.248236  0.542530
    8  0.483282  0.858480  0.216370  0.570318  0.808795  0.500356
    9  0.139096  0.033278  0.713375  0.143936  0.298430  0.401378

    [10 rows x 784 columns]
```

◇ Python Code

```
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

data = pd.read_csv("gan_images.csv").values
```

```
generator = Sequential([
    Dense(128, activation='relu', input_dim=100),
    Dense(784, activation='sigmoid')
])

discriminator = Sequential([
    Dense(128, activation='relu', input_shape=(784,)),
    Dense(1, activation='sigmoid')
])

discriminator.compile(optimizer='adam',
                      loss='binary_crossentropy')

print("GAN model initialized successfully")
```

◇ Output

```
GAN model initialized successfully
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer.
When using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```