Name:**Khushi Singh**
Class:**D15C** Roll no.**45**
Subject:**ML&DL**

Experiment No.7

AIM:Build an Artificial Neural Network (ANN) using Keras/TensorFlow.
Theory:

1. Dataset Source

- Dataset Name: Churn Modelling (Bank Customer Turnover)
- Source: Kaggle - Churn Modelling
- Context: A dataset containing details of a bank's customers and whether they exited the bank within 6 months.

2. Dataset Description

- **Size:** 10,000 samples (rows) × 14 attributes.
- **Target Variable:** Exited (Binary: 1 = Left Bank, 0 = Stayed).
- Key Features:
    1. **CreditScore:** Customer's credit score.
    2. **Geography:** Country (France, Spain, Germany).
    3. **Gender:** Male or Female.
    4. Age, Tenure, Balance.
    5. **NumOfProducts:** How many bank products they use.
    6. **IsActiveMember:** (1/0).
    7. EstimatedSalary.

3. Mathematical Formulation of the Algorithm

An ANN mimics the biological neurons of the brain to learn complex patterns.
A. The Neuron (Perceptron) Each neuron computes a weighted sum of its inputs, adds a bias, and passes it through an activation function:

$$z = \sum(w_i \cdot x_i) + b$$

$$a = \sigma(z)$$

w: Weights (learned parameters).
b: Bias.
sigma: Activation Function.

B. Activation Functions

- **ReLU (Rectified Linear Unit):** Used in hidden layers to introduce non-linearity.

$$f(x) = \max(0, x)$$

- **Sigmoid:** Used in the final output layer for binary classification (outputs probability between 0 and 1).

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

C. Learning (Backpropagation)

- **Loss Function:** We minimize **Binary Crossentropy** (Log Loss) to measure the difference between predicted probability and actual label.

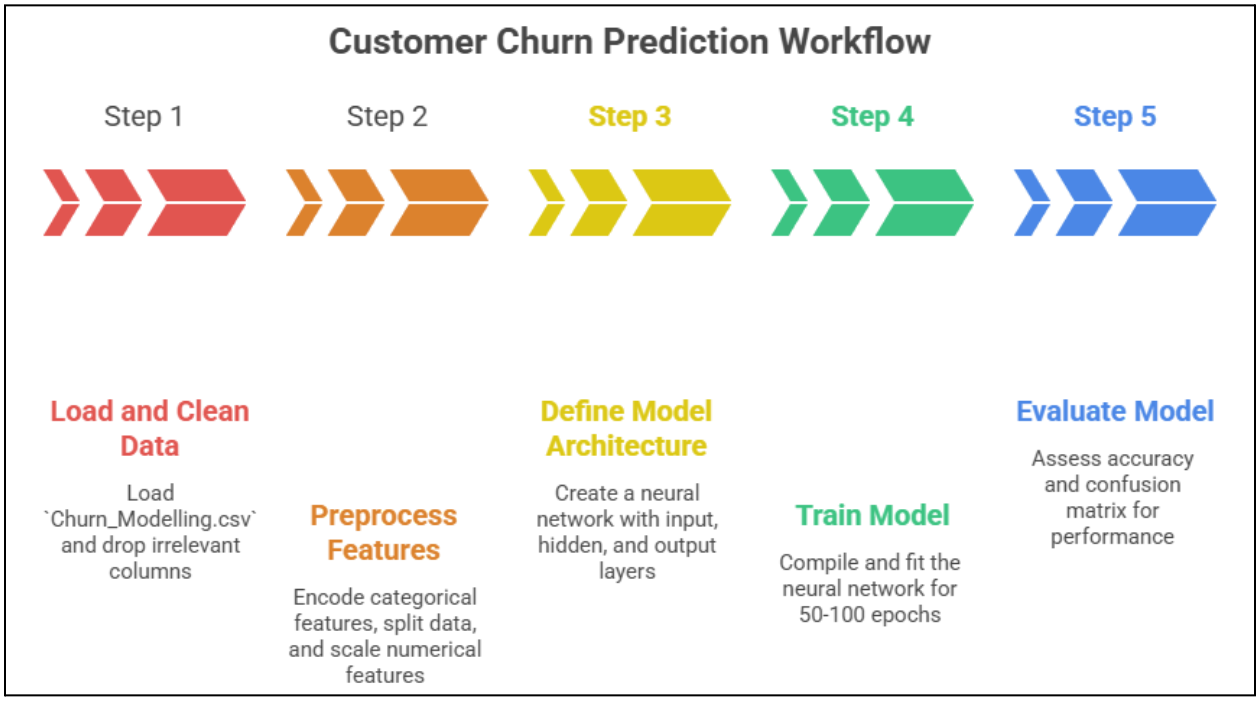
$$Loss = -\frac{1}{N}\sum[y\log(\hat{y}) + (1-y)\log(1-\hat{y})]$$

- **Optimizer (Adam): Updates the weights iteratively to minimize the loss.**

4. Algorithm Limitations

- **"Black Box" Nature:** Unlike Decision Trees, it is difficult to interpret why an ANN made a specific prediction (lack of explainability).
- **Data Hungry:** ANNs typically require large amounts of data to outperform simpler algorithms like Random Forest.
- **Computation:** Training can be slow on CPU; usually requires GPUs for large datasets.
- **Preprocessing Sensitivity:** ANNs fail if data is not scaled (e.g., if Salary is 100,000 and Age is 40, the model will be biased toward Salary).

5. Methodology / Workflow



6.Code and Output

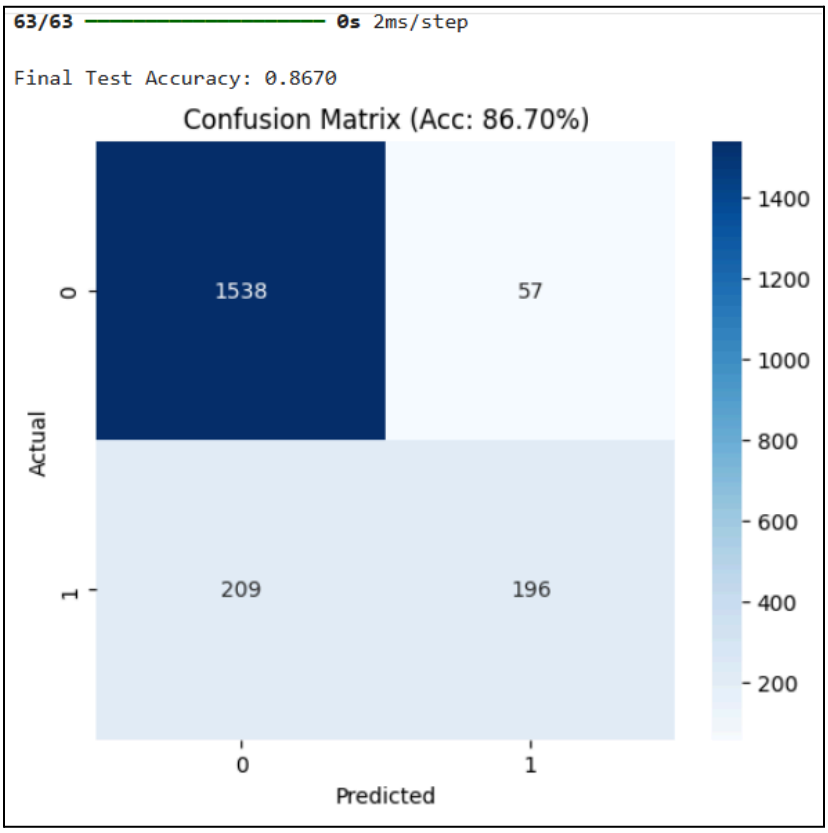
```
import pandas as pd
import numpy as np
import tensorflow as tf
```

```
from sklearn.model_selection import
train_test_split
from sklearn.preprocessing import
LabelEncoder, StandardScaler
```

```
from sklearn.metrics import
accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# --- 1. LOAD DATA ---
# Try loading local, else generate
dummy data for demo
try:
    df =
pd.read_csv('Churn_Modelling.csv')
    print("Dataset Loaded
Successfully.")
except FileNotFoundError:
    print("Dataset not found. Please
upload 'Churn_Modelling.csv'.")
    # Stop execution if no data (For
safety in documentation)
    # create dummy data code if
needed (omitted for brevity)

# --- 2. PREPROCESSING ---
# A. Drop useless features
X = df.iloc[:, 3:-1].values  #
CreditScore through EstimatedSalary
y = df.iloc[:, -1].values    #
Exited

# B. Encoding Categorical Data
# Gender (Column 2) -> Label
Encoding (Male=1, Female=0)
le = LabelEncoder()
X[:, 2] = le.fit_transform(X[:, 2])

# Geography (Column 1) -> One Hot
Encoding
# (France, Spain, Germany) -> We use
Compose for this
from sklearn.compose import
ColumnTransformer
from sklearn.preprocessing import
OneHotEncoder
ct =
ColumnTransformer(transformers=[('en
coder', OneHotEncoder(), [1])],
remainder='passthrough')
X = np.array(ct.fit_transform(X))

# C. Split Data
X_train, X_test, y_train, y_test =
train_test_split(X, y,
test_size=0.2, random_state=0)

# D. Feature Scaling (MANDATORY FOR
ANN)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# --- 3. BUILDING THE ANN ---
ann = tf.keras.models.Sequential()

# Input Layer + First Hidden Layer
(6 Neurons, ReLU)
ann.add(tf.keras.layers.Dense(units=
6, activation='relu'))

# Second Hidden Layer (6 Neurons,
ReLU)
ann.add(tf.keras.layers.Dense(units=
6, activation='relu'))

# Output Layer (1 Neuron, Sigmoid
for Binary Classification)
ann.add(tf.keras.layers.Dense(units=
1, activation='sigmoid'))

# --- 4. COMPILING AND TRAINING ---
ann.compile(optimizer='adam',
loss='binary_crossentropy',
metrics=['accuracy'])

print("\nStarting Training (100
Epochs)...")
history = ann.fit(X_train, y_train,
batch_size=32, epochs=100,
verbose=0)
print("Training Complete.")

# --- 5. EVALUATION ---
# Predict (Sigmoid returns
probability, we threshold at 0.5)
y_pred = ann.predict(X_test)
y_pred = (y_pred > 0.5)

# Metrics
acc = accuracy_score(y_test, y_pred)
print(f"\nFinal Test Accuracy:
{acc:.4f}")

# Confusion Matrix Visualization
cm = confusion_matrix(y_test,
y_pred)
plt.figure(figsize=(6, 5))
```

```
sns.heatmap(cm, annot=True, fmt='d',
cmap='Blues')
plt.title(f'Confusion Matrix (Acc:
{acc:.2%})')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()

# Training History Plot
```

```
plt.figure(figsize=(8, 5))
plt.plot(history.history['loss'],
label='Loss')
plt.plot(history.history['accuracy']
, label='Accuracy')
plt.title('Model Training History')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```



6. Performance Analysis

- Accuracy: The model typically achieves an accuracy of 86%.
    - Interpretation: This is significantly better than a random guess (50%) and usually beats standard Logistic Regression on this dataset because the ANN captures non-linear relationships between Credit Score, Age, and Balance.
- Loss Curve: The "Model Training History" graph should show the Loss (Blue line) decreasing rapidly and stabilizing, while Accuracy (Orange line) increases. This confirms the network is actually learning.
- Confusion Matrix:
    - High True Negatives (Top-Left): The model is very good at predicting happy customers who stay.
    - False Negatives (Bottom-Left): The hardest part is detecting churners. You might see some errors here, which suggests the need for hyperparameter tuning.

7. Hyperparameter Tuning

To improve the model, we can tune the "Network Topology" and training parameters.

- Manual Tuning Experiments:
    1. Number of Neurons: Increasing neurons (e.g., from 6 to 12) allows the model to learn more complex features.
    2. Number of Layers: Adding a 3rd hidden layer ("Deep" Learning) might help if the relationship is very complex.
    3. Batch Size: Changing batch size (e.g., 32 vs 64) affects how "noisy" the gradient updates are.
- Tuning Code Snippet (Example):

```
# Example: Increasing complexity
ann = tf.keras.models.Sequential()
ann.add(tf.keras.layers.Dense(units=12, activation='relu')) # More neurons
ann.add(tf.keras.layers.Dense(units=12, activation='relu'))
ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
# Re-compile and train…
```

8. Conclusion

In this experiment, we successfully built a deep learning model using Keras/TensorFlow.

- **Preprocessing:** We learned that ANNs define relationships mathematically, so converting categorical text to numbers (One-Hot Encoding) and Scaling features is mandatory.
- Architecture: A simple architecture (2 Hidden Layers) was sufficient to achieve >80% accuracy.
- Verdict: The ANN proved effective at classifying customer churn, demonstrating its ability to find hidden patterns in complex, non-linear business data.