import pandas as pd  
from sklearn.model\_selection import train\_test\_split # Changed train\_selection to train\_test\_split  
import matplotlib.pyplot as plt  
from matplotlib import pyplot  
from sklearn.utils import shuffle

**DATASET BALANCING USING PANDAS**

df=pd.read\_csv( 'magicdataset.csv' )  
df

{"summary":"{\n \"name\": \"df\",\n \"rows\": 19020,\n \"fields\": [\n {\n \"column\": \"fLength\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 42.364854942802815,\n \"min\": 4.2835,\n \"max\": 334.177,\n \"num\_unique\_values\": 18643,\n \"samples\": [\n 29.3302,\n 61.2341,\n 40.7017\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"fWidth\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 18.346056295681635,\n \"min\": 0.0,\n \"max\": 256.382,\n \"num\_unique\_values\": 18200,\n \"samples\": [\n 10.5168,\n 22.4704,\n 18.0348\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"fSize\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 0.4725986486893089,\n \"min\": 1.9413,\n \"max\": 5.3233,\n \"num\_unique\_values\": 7228,\n \"samples\": [\n 2.8136,\n 2.5121,\n 3.3903\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"fConc\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 0.18281314722123734,\n \"min\": 0.0131,\n \"max\": 0.893,\n \"num\_unique\_values\": 6410,\n \"samples\": [\n 0.0997,\n 0.1842,\n 0.131\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"fConc1\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 0.11051079890195728,\n \"min\": 0.0003,\n \"max\": 0.6752,\n \"num\_unique\_values\": 4421,\n \"samples\": [\n 0.4651,\n 0.0325,\n 0.1752\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"fAsym\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 59.20606198471514,\n \"min\": -457.9161,\n \"max\": 575.2407,\n \"num\_unique\_values\": 18704,\n \"samples\": [\n -59.0369,\n 31.8588,\n 32.1961\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"fM3Long\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 51.00011801388597,\n \"min\": -331.78,\n \"max\": 238.321,\n \"num\_unique\_values\": 18693,\n \"samples\": [\n -12.7648,\n 18.1689,\n 18.9666\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"fM3Trans\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 20.827438947228195,\n \"min\": -205.8947,\n \"max\": 179.851,\n \"num\_unique\_values\": 18390,\n \"samples\": [\n -39.2878,\n -9.7515,\n -17.2016\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"fAlpha\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 26.103620510358212,\n \"min\": 0.0,\n \"max\": 90.0,\n \"num\_unique\_values\": 17981,\n \"samples\": [\n 8.3289,\n 5.86,\n 1.2801\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"fDist\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 74.73178696313774,\n \"min\": 1.2826,\n \"max\": 495.561,\n \"num\_unique\_values\": 18437,\n \"samples\": [\n 290.884,\n 199.29,\n 171.345\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"class\",\n \"properties\": {\n \"dtype\": \"category\",\n \"num\_unique\_values\": 2,\n \"samples\": [\n \"h\",\n \"g\"\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n }\n ]\n}","type":"dataframe","variable\_name":"df"}

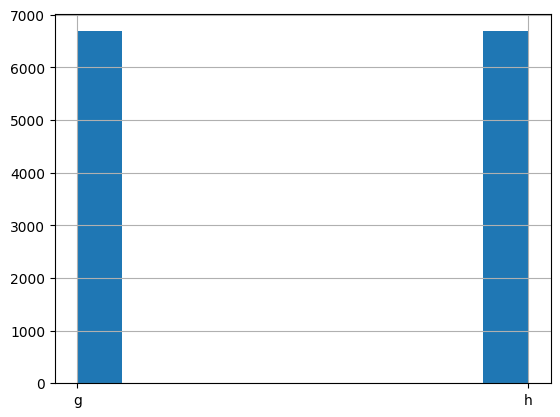
classG=df[df['class']== 'g' ]  
classH=df[df['class']== 'h' ]  
countG, countH = df['class'].value\_counts()

classGUnder = classG.sample(countH)  
newDataset = pd.concat([classGUnder, classH], axis=0)  
newDataset.to\_csv('balanced\_dataset.csv',index=False)  
# df1=pd.read\_csv( 'balanced\_dataset.csv' )  
# df1

# Reading Data without recreating it each time

newDataset = pd.read\_csv('balanced\_dataset.csv')  
newDataset['class'].hist()

<Axes: >



**Data Split**

x = newDataset.drop('class', axis=1) # 1 for column, 0 for index  
y = newDataset['class'] # Remove the trailing comma to avoid creating a tuple  
  
x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3)

**IMPORTING MODELS**

import numpy as np  
import tensorflow as tf  
from tensorflow.keras import layers, models  
from sklearn.preprocessing import StandardScaler

**STANDARDIZE THE FEATURES**

scaler = StandardScaler()  
x\_train = scaler.fit\_transform(x\_train)  
x\_test = scaler.transform(x\_test)

**BUILDING THE MODEL**

We will Build the Neural Network Model using the Keras API in TensorFlow.

**MODEL ARCHITECTURE**

Model: "sequential"

* The Model consists of three layers: two hidden layer and one output layer
* The input layer takes in the 10 features from our dataset

model = models.Sequential()  
model.add(layers.Dense(64, activation='relu', input\_shape=(x\_train.shape[1],)))  
model.add(layers.Dense(32, activation='relu'))  
model.add(layers.Dense(1, activation='sigmoid'))

/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: 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)

loss\_functions = {  
 'binary\_crossentropy': 'binary\_crossentropy',  
 'hinge': 'hinge',  
 'focal\_loss' : 'binary\_crossentropy' #Placeholder for focal loss  
 }

def focal\_loss(gamma=2.0, alpha=0.25):  
 def focal\_loss\_fixed(y\_true, y\_pred):  
 epsilon = tf.keras.backend.epsilon()  
 y\_pred = tf.clip\_by\_value(y\_pred, epsilon, 1. - epsilon)  
 cross\_entropy = -y\_true \* tf.math.log(y\_pred)  
 loss = alpha \* tf.pow(1 - y\_pred, gamma) \* cross\_entropy  
 return tf.reduce\_mean(tf.reduce\_sum(loss, axis=1))  
 return focal\_loss\_fixed

**Neural Network using TensorFlow**

import pandas as pd  
import numpy as np  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import StandardScaler  
import tensorflow as tf  
from tensorflow.keras import layers, models  
  
# Load the dataset  
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/magic/magic04.data"  
column\_names = ['fLength', 'fWidth', 'fSize', 'fConc', 'fConc1', 'fAsym',  
 'fM3Long', 'fM3Trans', 'fAlpha', 'fDist', 'class']  
data = pd.read\_csv(url, header=None, names=column\_names)  
  
# Preprocess the data  
data['class'] = data['class'].map({'g': 1, 'h': 0}) # Convert class labels to binary  
X = data.drop('class', axis=1)  
y = data['class']  
  
# Split the dataset into majority and minority classes  
classG = data[data['class'] == 1]  
classH = data[data['class'] == 0]  
  
# Balance the dataset by undersampling the majority class  
countH = len(classH)  
classGUnder = classG.sample(countH)  
newDataset = pd.concat([classGUnder, classH], axis=0)  
  
# Split the balanced dataset  
X = newDataset.drop('class', axis=1)  
y = newDataset['class']  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# Standardize the features  
scaler = StandardScaler()  
X\_train = scaler.fit\_transform(X\_train)  
X\_test = scaler.transform(X\_test)  
  
# Build the model  
model = models.Sequential()  
model.add(layers.Dense(128, activation='relu', input\_shape=(X\_train.shape[1],)))  
model.add(layers.Dense(64, activation='relu'))  
model.add(layers.Dense(32, activation='relu'))  
model.add(layers.Dense(1, activation='sigmoid')) # Output layer for binary classification  
  
# Compile the model with different loss functions  
loss\_functions = {  
 'binary\_crossentropy': 'binary\_crossentropy',  
 'hinge': 'hinge',  
 'focal\_loss': focal\_loss(gamma=2.0, alpha=0.25) # Use the custom focal loss function  
}  
  
# Custom focal loss function  
def focal\_loss(gamma=2.0, alpha=0.25):  
 def focal\_loss\_fixed(y\_true, y\_pred):  
 epsilon = tf.keras.backend.epsilon()  
 y\_pred = tf.clip\_by\_value(y\_pred, epsilon, 1. - epsilon)  
 cross\_entropy = -y\_true \* tf.math.log(y\_pred)  
 loss = alpha \* tf.pow(1 - y\_pred, gamma) \* cross\_entropy  
 return tf.reduce\_mean(tf.reduce\_sum(loss, axis=1))  
 return focal\_loss\_fixed  
  
# Train and evaluate the model with different loss functions  
for loss\_name, loss\_function in loss\_functions.items():  
 print(f"Training with loss function: {loss\_name}")  
  
 model.compile(optimizer='adam', loss=loss\_function, metrics=['accuracy'])  
  
 model.fit(X\_train, y\_train, epochs=10, batch\_size=32, validation\_split=0.2)  
 test\_loss, test\_accuracy = model.evaluate(X\_test, y\_test)  
 print(f"Test accuracy with {loss\_name}: {test\_accuracy:.4f}\n")

Training with loss function: binary\_crossentropy  
Epoch 1/10

/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: 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)

268/268 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.7544 - loss: 0.5082 - val\_accuracy: 0.8318 - val\_loss: 0.3788  
Epoch 2/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8338 - loss: 0.3745 - val\_accuracy: 0.8332 - val\_loss: 0.3731  
Epoch 3/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8417 - loss: 0.3629 - val\_accuracy: 0.8379 - val\_loss: 0.3522  
Epoch 4/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8477 - loss: 0.3500 - val\_accuracy: 0.8430 - val\_loss: 0.3443  
Epoch 5/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8502 - loss: 0.3388 - val\_accuracy: 0.8421 - val\_loss: 0.3436  
Epoch 6/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8565 - loss: 0.3300 - val\_accuracy: 0.8458 - val\_loss: 0.3385  
Epoch 7/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8577 - loss: 0.3274 - val\_accuracy: 0.8397 - val\_loss: 0.3447  
Epoch 8/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8558 - loss: 0.3278 - val\_accuracy: 0.8500 - val\_loss: 0.3402  
Epoch 9/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8584 - loss: 0.3253 - val\_accuracy: 0.8411 - val\_loss: 0.3459  
Epoch 10/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8563 - loss: 0.3179 - val\_accuracy: 0.8472 - val\_loss: 0.3339  
84/84 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8569 - loss: 0.3325  
Test accuracy with binary\_crossentropy: 0.8524  
  
Training with loss function: hinge  
Epoch 1/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8446 - loss: 0.6705 - val\_accuracy: 0.8383 - val\_loss: 0.6671  
Epoch 2/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8524 - loss: 0.6569 - val\_accuracy: 0.8425 - val\_loss: 0.6600  
Epoch 3/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8578 - loss: 0.6468 - val\_accuracy: 0.8491 - val\_loss: 0.6539  
Epoch 4/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8550 - loss: 0.6431 - val\_accuracy: 0.8402 - val\_loss: 0.6643  
Epoch 5/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8581 - loss: 0.6409 - val\_accuracy: 0.8486 - val\_loss: 0.6556  
Epoch 6/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8609 - loss: 0.6311 - val\_accuracy: 0.8458 - val\_loss: 0.6579  
Epoch 7/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8569 - loss: 0.6507 - val\_accuracy: 0.8425 - val\_loss: 0.6593  
Epoch 8/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8576 - loss: 0.6404 - val\_accuracy: 0.8383 - val\_loss: 0.6645  
Epoch 9/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8619 - loss: 0.6365 - val\_accuracy: 0.8505 - val\_loss: 0.6528  
Epoch 10/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8567 - loss: 0.6432 - val\_accuracy: 0.8430 - val\_loss: 0.6590  
84/84 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8364 - loss: 0.6631  
Test accuracy with hinge: 0.8419  
  
Training with loss function: focal\_loss  
Epoch 1/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7456 - loss: 0.0174 - val\_accuracy: 0.5332 - val\_loss: 7.2481e-16  
Epoch 2/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.5275 - loss: 8.4542e-07 - val\_accuracy: 0.5322 - val\_loss: 4.5793e-16  
Epoch 3/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.5503 - loss: 9.6561e-08 - val\_accuracy: 0.5322 - val\_loss: 3.8910e-16  
Epoch 4/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.5276 - loss: 3.3382e-08 - val\_accuracy: 0.5322 - val\_loss: 3.4315e-16  
Epoch 5/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.5245 - loss: 8.9874e-08 - val\_accuracy: 0.5322 - val\_loss: 3.1287e-16  
Epoch 6/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.5238 - loss: 6.7383e-09 - val\_accuracy: 0.5318 - val\_loss: 2.9286e-16  
Epoch 7/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.5298 - loss: 2.2124e-08 - val\_accuracy: 0.5318 - val\_loss: 2.6395e-16  
Epoch 8/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5315 - loss: 7.9460e-09 - val\_accuracy: 0.5318 - val\_loss: 2.4525e-16  
Epoch 9/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5355 - loss: 7.3973e-09 - val\_accuracy: 0.5318 - val\_loss: 2.2933e-16  
Epoch 10/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5271 - loss: 5.5966e-09 - val\_accuracy: 0.5318 - val\_loss: 2.1530e-16  
84/84 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5261 - loss: 7.5497e-12  
Test accuracy with focal\_loss: 0.5247

import pandas as pd  
import numpy as np  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import StandardScaler  
from sklearn.metrics import classification\_report  
import tensorflow as tf  
from tensorflow.keras import layers, models  
  
# Load the dataset  
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/magic/magic04.data"  
column\_names = ['fLength', 'fWidth', 'fSize', 'fConc', 'fConc1', 'fAsym',  
 'fM3Long', 'fM3Trans', 'fAlpha', 'fDist', 'class']  
data = pd.read\_csv(url, header=None, names=column\_names)  
  
# Preprocess the data  
data['class'] = data['class'].map({'g': 1, 'h': 0}) # Convert class labels to binary  
X = data.drop('class', axis=1)  
y = data['class']  
  
# Split the dataset into majority and minority classes  
classG = data[data['class'] == 1]  
classH = data[data['class'] == 0]  
  
# Balance the dataset by undersampling the majority class  
countH = len(classH)  
classGUnder = classG.sample(countH)  
newDataset = pd.concat([classGUnder, classH], axis=0)  
  
# Split the balanced dataset  
X = newDataset.drop('class', axis=1)  
y = newDataset['class']  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# Standardize the features  
scaler = StandardScaler()  
X\_train = scaler.fit\_transform(X\_train)  
X\_test = scaler.transform(X\_test)  
  
# Custom focal loss function  
def focal\_loss(gamma=2.0, alpha=0.25):  
 def focal\_loss\_fixed(y\_true, y\_pred):  
 epsilon = tf.keras.backend.epsilon()  
 y\_pred = tf.clip\_by\_value(y\_pred, epsilon, 1. - epsilon)  
 cross\_entropy = -y\_true \* tf.math.log(y\_pred)  
 loss = alpha \* tf.pow(1 - y\_pred, gamma) \* cross\_entropy  
 return tf.reduce\_mean(tf.reduce\_sum(loss, axis=1))  
 return focal\_loss\_fixed  
  
# Build the model  
model = models.Sequential()  
model.add(layers.Dense(128, activation='relu', input\_shape=(X\_train.shape[1],)))  
model.add(layers.Dense(64, activation='relu'))  
model.add(layers.Dense(32, activation='relu'))  
model.add(layers.Dense(1, activation='sigmoid')) # Output layer for binary classification  
  
# Compile the model with different loss functions  
loss\_functions = {  
 'binary\_crossentropy': 'binary\_crossentropy',  
 'hinge': 'hinge',  
 'focal\_loss': focal\_loss(gamma=2.0, alpha=0.25) # Use the custom focal loss function  
}  
  
# Train and evaluate the model with different loss functions  
for loss\_name, loss\_function in loss\_functions.items():  
 print(f"Training with loss function: {loss\_name}")  
  
 model.compile(optimizer='adam', loss=loss\_function, metrics=['accuracy'])  
  
 model.fit(X\_train, y\_train, epochs=10, batch\_size=32, validation\_split=0.2)  
 test\_loss, test\_accuracy = model.evaluate(X\_test, y\_test)  
 print(f"Test accuracy with {loss\_name}: {test\_accuracy:.4f}")  
  
 # Get predictions  
 y\_pred\_prob = model.predict(X\_test)  
 y\_pred = (y\_pred\_prob > 0.5).astype(int) # Convert probabilities to binary predictions  
  
 # Calculate precision, recall, and F1 score  
 report = classification\_report(y\_test, y\_pred, target\_names=['Class H (0)', 'Class G (1)'], output\_dict=True)  
  
 precision = report['Class G (1)']['precision']  
 recall = report['Class G (1)']['recall']  
 f1\_score = report['Class G (1)']['f1-score']  
  
 print(f"Precision for Class G (1): {precision:.4f}")  
 print(f"Recall for Class G (1): {recall:.4f}")  
 print(f"F1 Score for Class G (1): {f1\_score:.4f}\n")

Training with loss function: binary\_crossentropy  
Epoch 1/10

/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: 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)

268/268 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.7190 - loss: 0.5397 - val\_accuracy: 0.8196 - val\_loss: 0.3831  
Epoch 2/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8262 - loss: 0.3867 - val\_accuracy: 0.8336 - val\_loss: 0.3575  
Epoch 3/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8455 - loss: 0.3588 - val\_accuracy: 0.8350 - val\_loss: 0.3622  
Epoch 4/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8429 - loss: 0.3508 - val\_accuracy: 0.8463 - val\_loss: 0.3341  
Epoch 5/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8500 - loss: 0.3381 - val\_accuracy: 0.8472 - val\_loss: 0.3329  
Epoch 6/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8504 - loss: 0.3273 - val\_accuracy: 0.8477 - val\_loss: 0.3323  
Epoch 7/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8540 - loss: 0.3291 - val\_accuracy: 0.8561 - val\_loss: 0.3268  
Epoch 8/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8525 - loss: 0.3316 - val\_accuracy: 0.8598 - val\_loss: 0.3230  
Epoch 9/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8556 - loss: 0.3262 - val\_accuracy: 0.8523 - val\_loss: 0.3240  
Epoch 10/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8566 - loss: 0.3191 - val\_accuracy: 0.8533 - val\_loss: 0.3412  
84/84 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8396 - loss: 0.3604  
Test accuracy with binary\_crossentropy: 0.8382  
84/84 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step  
Precision for Class G (1): 0.8484  
Recall for Class G (1): 0.8209  
F1 Score for Class G (1): 0.8344  
  
Training with loss function: hinge  
Epoch 1/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8534 - loss: 0.6677 - val\_accuracy: 0.8444 - val\_loss: 0.6601  
Epoch 2/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8544 - loss: 0.6458 - val\_accuracy: 0.8439 - val\_loss: 0.6590  
Epoch 3/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8460 - loss: 0.6568 - val\_accuracy: 0.8383 - val\_loss: 0.6643  
Epoch 4/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8560 - loss: 0.6419 - val\_accuracy: 0.8416 - val\_loss: 0.6586  
Epoch 5/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8588 - loss: 0.6368 - val\_accuracy: 0.8481 - val\_loss: 0.6559  
Epoch 6/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8588 - loss: 0.6519 - val\_accuracy: 0.8383 - val\_loss: 0.6634  
Epoch 7/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8525 - loss: 0.6432 - val\_accuracy: 0.8477 - val\_loss: 0.6560  
Epoch 8/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8672 - loss: 0.6270 - val\_accuracy: 0.8477 - val\_loss: 0.6563  
Epoch 9/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8657 - loss: 0.6323 - val\_accuracy: 0.8495 - val\_loss: 0.6548  
Epoch 10/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8581 - loss: 0.6460 - val\_accuracy: 0.8346 - val\_loss: 0.6687  
84/84 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8303 - loss: 0.6712  
Test accuracy with hinge: 0.8352  
84/84 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step  
Precision for Class G (1): 0.8289  
Recall for Class G (1): 0.8420  
F1 Score for Class G (1): 0.8354  
  
Training with loss function: focal\_loss  
Epoch 1/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7438 - loss: 0.0164 - val\_accuracy: 0.5696 - val\_loss: 1.7174e-04  
Epoch 2/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.5743 - loss: 4.4009e-07 - val\_accuracy: 0.5654 - val\_loss: 1.4567e-04  
Epoch 3/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.5619 - loss: 3.7090e-08 - val\_accuracy: 0.5654 - val\_loss: 1.4494e-04  
Epoch 4/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.5643 - loss: 1.4793e-08 - val\_accuracy: 0.5654 - val\_loss: 1.4414e-04  
Epoch 5/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.5642 - loss: 4.2458e-08 - val\_accuracy: 0.5654 - val\_loss: 1.4337e-04  
Epoch 6/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.5636 - loss: 1.9525e-08 - val\_accuracy: 0.5654 - val\_loss: 1.4257e-04  
Epoch 7/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.5695 - loss: 1.1986e-08 - val\_accuracy: 0.5650 - val\_loss: 1.4176e-04  
Epoch 8/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5717 - loss: 5.8084e-09 - val\_accuracy: 0.5650 - val\_loss: 1.4097e-04  
Epoch 9/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.5684 - loss: 5.0746e-09 - val\_accuracy: 0.5640 - val\_loss: 1.4020e-04  
Epoch 10/10  
268/268 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.5673 - loss: 1.7621e-09 - val\_accuracy: 0.5640 - val\_loss: 1.3955e-04  
84/84 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5626 - loss: 3.6021e-06  
Test accuracy with focal\_loss: 0.5609  
84/84 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step  
Precision for Class G (1): 0.5308  
Recall for Class G (1): 1.0000  
F1 Score for Class G (1): 0.6935

import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.metrics import confusion\_matrix  
  
# Assuming y\_test and y\_pred are already defined from your previous code  
# y\_pred contains the binary predictions from the model  
  
# Create confusion matrix  
cm = confusion\_matrix(y\_test, y\_pred)  
  
# Plotting the confusion matrix  
plt.figure(figsize=(8, 6))  
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Class H (0)', 'Class G (1)'], yticklabels=['Class H (0)', 'Class G (1)'])  
plt.title('Confusion Matrix')  
plt.xlabel('Predicted Label')  
plt.ylabel('True Label')  
plt.show()

