CODE FILE:

The model will train using both text and image data and evaluate its performance. The structure of the code follows the previously discussed methodology, where you:

1. Collect data (text and image data for Instagram).
2. Preprocess data (clean text, resize images).
3. Use LSTM for text-based spam detection and CNN for image-based spam detection.
4. Combine both models for a hybrid spam detection system.
5. Evaluate and test the model.

For this example, I will provide a basic framework, but you should customize it with your dataset and additional functions as needed.

**Project Structure**

* **Dataset**: You should have a dataset with Instagram posts containing text (comments, captions) and images (for promotional or spam-related images). Use a collection of labeled data (spam or not spam).
* **Preprocessing**: Text data will be preprocessed using tokenization, padding, and embedding. Image data will be resized and normalized.
* **Model**: A hybrid model combining CNN for image classification and LSTM for text classification.
* **Evaluation**: Evaluate the model using accuracy, precision, recall, and F1 score.

Code:

import numpy as np

import pandas as pd

import tensorflow as tf

from tensorflow.keras.models import Model

from tensorflow.keras.layers import LSTM, Embedding, Dense, Dropout, Conv2D, MaxPooling2D, Flatten, Input

from tensorflow.keras.preprocessing.text import Tokenizer

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

from tensorflow.keras.applications import VGG16

from tensorflow.keras.optimizers import Adam

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

from sklearn.metrics import accuracy\_score, classification\_report

import cv2

import os

# Data Loading and Preprocessing

# Assuming you have a dataset in CSV format with 'text' and 'image\_path' columns

data = pd.read\_csv('instagram\_data.csv') # Replace with your dataset path

texts = data['text']

labels = data['label'] # 1 for spam, 0 for not spam

image\_paths = data['image\_path']

# Preprocess Text Data

max\_words = 10000 # Maximum number of words to use in Tokenizer

max\_sequence\_length = 100 # Maximum length of text sequences

embedding\_dim = 100 # Dimension for word embeddings

tokenizer = Tokenizer(num\_words=max\_words)

tokenizer.fit\_on\_texts(texts)

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

X\_text = pad\_sequences(sequences, maxlen=max\_sequence\_length)

# Preprocess Image Data

image\_size = (224, 224) # Resize image to fit VGG16 input size

X\_images = []

for img\_path in image\_paths:

img = cv2.imread(img\_path)

img = cv2.resize(img, image\_size)

img = img / 255.0 # Normalize image

X\_images.append(img)

X\_images = np.array(X\_images)

# Train-Test Split

X\_text\_train, X\_text\_test, X\_images\_train, X\_images\_test, y\_train, y\_test = train\_test\_split(

X\_text, X\_images, labels, test\_size=0.2, random\_state=42

)

# Build the Hybrid Model (Text + Image)

# Text Model (LSTM)

text\_input = Input(shape=(max\_sequence\_length,))

embedding = Embedding(input\_dim=max\_words, output\_dim=embedding\_dim, input\_length=max\_sequence\_length)(text\_input)

x\_text = LSTM(128, dropout=0.2)(embedding)

# Image Model (CNN - using VGG16 pre-trained model)

image\_input = Input(shape=(224, 224, 3))

vgg\_model = VGG16(weights='imagenet', include\_top=False, input\_shape=(224, 224, 3))

vgg\_model.trainable = False # Freeze VGG16 layers

x\_image = vgg\_model(image\_input)

x\_image = Flatten()(x\_image)

x\_image = Dense(128, activation='relu')(x\_image)

# Combine Text and Image Models

combined = tf.keras.layers.concatenate([x\_text, x\_image])

combined = Dense(128, activation='relu')(combined)

combined = Dropout(0.5)(combined)

output = Dense(1, activation='sigmoid')(combined)

# Create the Final Model

model = Model(inputs=[text\_input, image\_input], outputs=output)

# Compile the Model

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

# Train the Model

history = model.fit(

[X\_text\_train, X\_images\_train], y\_train,

epochs=10, batch\_size=32,

validation\_data=([X\_text\_test, X\_images\_test], y\_test)

)

# Evaluate the Model

y\_pred = model.predict([X\_text\_test, X\_images\_test])

y\_pred = (y\_pred > 0.5) # Convert probabilities to binary predictions

# Accuracy and Evaluation

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {accuracy:.4f}")

print(classification\_report(y\_test, y\_pred))

# Save the Model

model.save('instagram\_spam\_detection\_model.h5')

# Visualization: Plot Training and Validation Loss

import matplotlib.pyplot as plt

plt.plot(history.history['loss'], label='Train Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.legend()

plt.show()

### # You can visualize accuracy in a similar manner Explanation of Code:

1. **Data Loading and Preprocessing**:
   * **Text Preprocessing**: The Tokenizer is used to convert text data into sequences of integers. These sequences are padded to ensure uniform length across all input texts.
   * **Image Preprocessing**: Images are read from file paths, resized to fit the input shape of the VGG16 model, and normalized by scaling pixel values to the range [0, 1].
2. **Model Architecture**:
   * **Text Model (LSTM)**: A simple LSTM network is used to process text input. The LSTM layer captures the sequential nature of the text and provides a rich feature representation of the text data.
   * **Image Model (CNN)**: The pre-trained VGG16 model is used for image classification. It is frozen (its weights are not updated during training) to avoid overfitting, and its output is processed through a dense layer.
   * **Hybrid Model**: The outputs from both the text and image models are concatenated and passed through a final dense layer to predict whether the content is spam or not.
3. **Training and Evaluation**:
   * The model is trained on both text and image data. It uses binary cross-entropy as the loss function (since it's a binary classification problem) and accuracy as the evaluation metric.
   * After training, the model is evaluated using accuracy, precision, recall, and F1 score, which provide insights into the performance of the model.
4. **Results Visualization**: The training and validation loss are plotted to observe the convergence of the model.

**Customization:**

1. **Dataset**: The dataset (instagram\_data.csv) should include columns for text (Instagram post text) and image\_path (path to the image associated with the post).
2. **Training Configuration**: You can tweak hyperparameters such as the number of epochs, batch size, and model architecture based on your dataset and computational resources.
3. **Model Performance**: To improve model performance, you can fine-tune VGG16, use a more sophisticated text processing model (like BERT), or adjust the hybrid model architecture.

**Conclusion:**

This Python-based Instagram spam detection system is a complete pipeline that combines both text and image data for spam detection. By leveraging deep learning techniques (LSTM for text and CNN for images), it provides a robust solution for detecting spam posts on Instagram. This project is a solid foundation for further optimization, such as incorporating real-time learning and improving model generalization.