National Institute of Electronics and Information Technology, Calicut

PG program in Data Analytics and Artificial Intelligence

Deepfake and Real image classification

Submitted by:
A Athira
Nayeem A
Sanjay P
Swathi P P

Contents

I. Abstract

The Deepfake and Real Image Classification Project aims to develop a robust machine learning model capable of distinguishing between deepfake and authentic images. Leveraging a diverse dataset, this project employs various deep learning techniques and evaluation metrics to achieve accurate classification. Key steps include data preprocessing, model development, performance evaluation, and model deployment. The project concludes with insights and recommendations for stakeholders concerned with combatting the spread of misinformation and preserving digital authenticity.

2. Introduction

In the contemporary digital landscape, the emergence of deepfake technology presents a formidable challenge to the authenticity and reliability of visual content. The Deepfake and Real Image Classification Project addresses this pressing concern by endeavoring to develop a sophisticated deep learning model capable of accurately discerning between fake and real images. Beginning with the meticulous collection of a diverse dataset encompassing fake and real images from various sources, the project proceeds with rigorous data preprocessing to ensure the suitability of the data for model training. The project embarks on model development, utilizing transfer learning to capitalize on pre-trained CNN architectures for efficient model training. Implemented within the program are modular components facilitating seamless data preprocessing, model training, evaluation, and deployment. Subsequent to model training, comprehensive performance evaluation ensues, employing an array of metrics such as accuracy, precision, recall, and ROC curves to assess model efficacy. The ultimate aim is to deploy the best-performing model in real-world scenarios, enabling stakeholders to integrate it into existing systems for automated deepfake detection and classification, thus contributing to the broader efforts to combat misinformation and preserve digital authenticity.

3. Objective

The primary objective is to develop a robust classification model capable of accurately identifying deepfake images. By leveraging advanced deep learning techniques, the project aims to contribute to the ongoing efforts to combat misinformation and preserve digital trustworthiness.

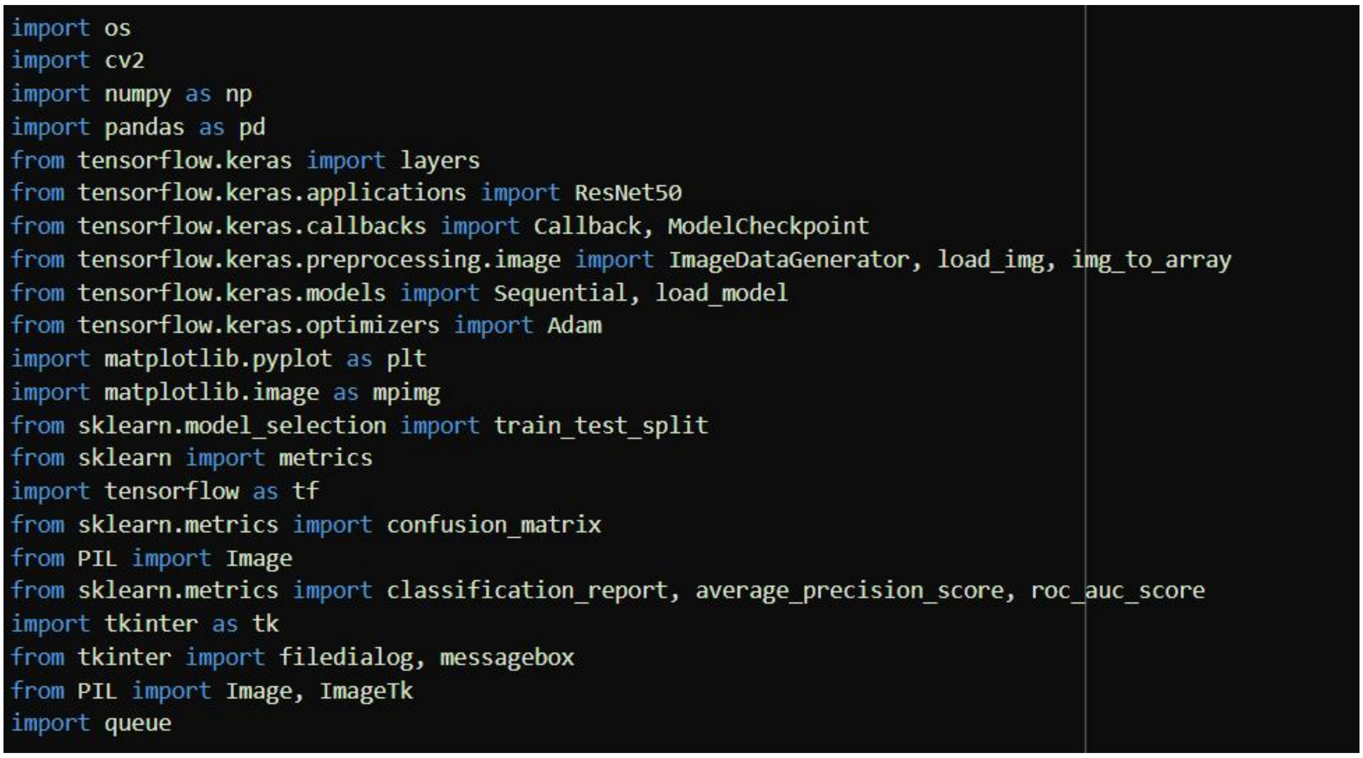
4. Scope of the project

The project encompasses data collection, preprocessing, model development, evaluation, and deployment. It leverages state-of-the-art deep learning architectures and evaluation metrics to assess model performance comprehensively. The insights generated from the project can inform strategies for detecting and mitigating the spread of deepfake content.

5. Procedure

This Python program is designed as an image classifier specifically tailored to discern between **real** and **fake** images. It begins by loading image datasets using libraries such as **cv2**, **os**, and **PIL**, followed by preprocessing using **TensorFlow's ImageDataGenerator** to resize and rescale images. Leveraging the **ResNet50** architecture pretrained on **ImageNet**, the program builds a deep learning model for feature extraction and classification, further enhanced with additional layers. Utilizing **TensorFlow's Keras API**, the model is compiled with appropriate loss functions, optimizers, and evaluation metrics. During training, callbacks monitor progress, while evaluation metrics such as accuracy, precision, recall, and ROC AUC score provide insights into model performance. Moreover, the program features a user-friendly interface built using **Tkinter**, enabling users to browse and select images for classification. Once selected, images undergo preprocessing, and predictions are made using the trained model, with results displayed on the interface. With its modular design and integration capabilities, the program facilitates easy deployment for real-time classification tasks, offering a comprehensive solution to combat the proliferation of deepfake content across various domains.

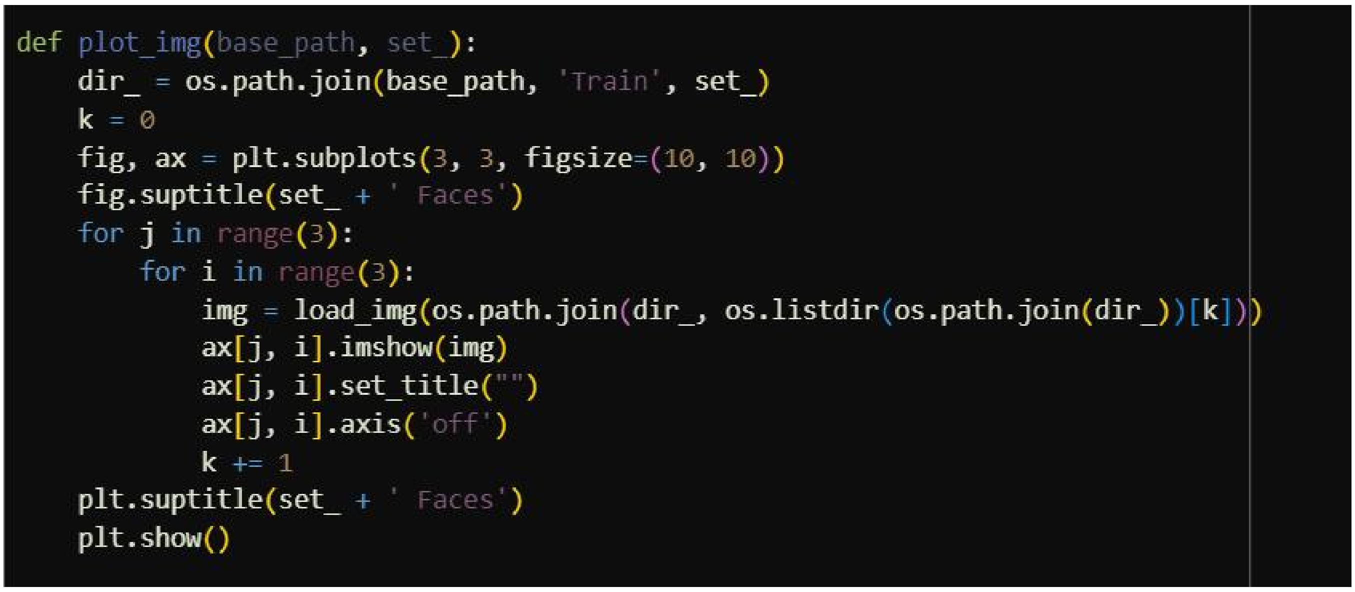
5.1. Importing required libraries



```
import os
import cv2
import numpy as np
import pandas as pd
from tensorflow.keras import layers
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.callbacks import Callback, ModelCheckpoint
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from sklearn.model_selection import train_test_split
from sklearn import metrics
import tensorflow as tf
from sklearn.metrics import confusion_matrix
from PIL import Image
from sklearn.metrics import classification_report, average_precision_score, roc_auc_score
import tkinter as tk
from tkinter import filedialog, messagebox
from PIL import Image, ImageTk
import queue
```

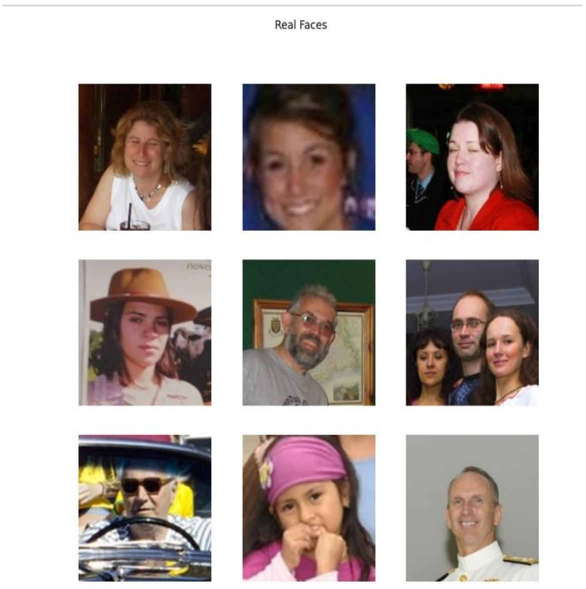
Various libraries are imported for tasks related to computer vision, deep learning, data manipulation, and visualization. Notably, **OpenCV (cv2)**, **NumPy (np)**, **Pandas (pd)**, **TensorFlow (tf)**, **Matplotlib (plt)**, **scikit-learn (sklearn)**, **PIL (Image)** and **Tkinter (tk)** are imported alongside modules for specific functionalities like image preprocessing, model creation, callbacks, and GUI development. The code is geared towards building and evaluating a deep learning model, for image classification tasks, utilizing the **ResNet50** architecture as a backbone. Additionally, it include components for data augmentation, model training, evaluation metrics computation, and GUI-based image input/output.

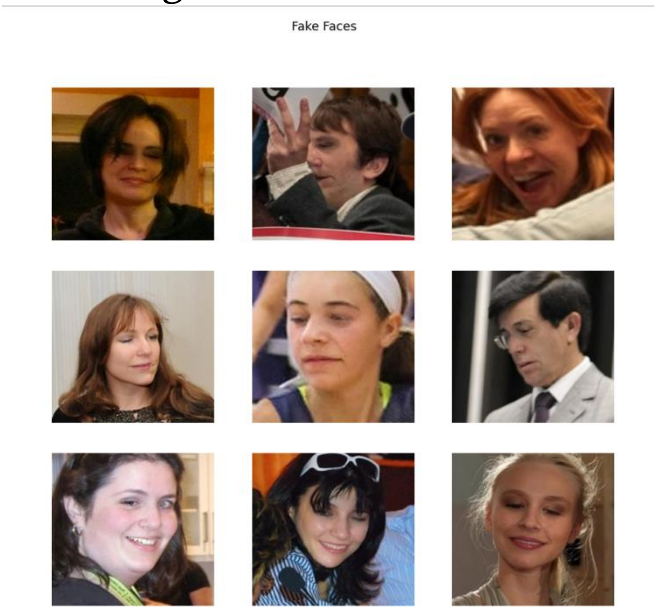
5.2. Plotting images



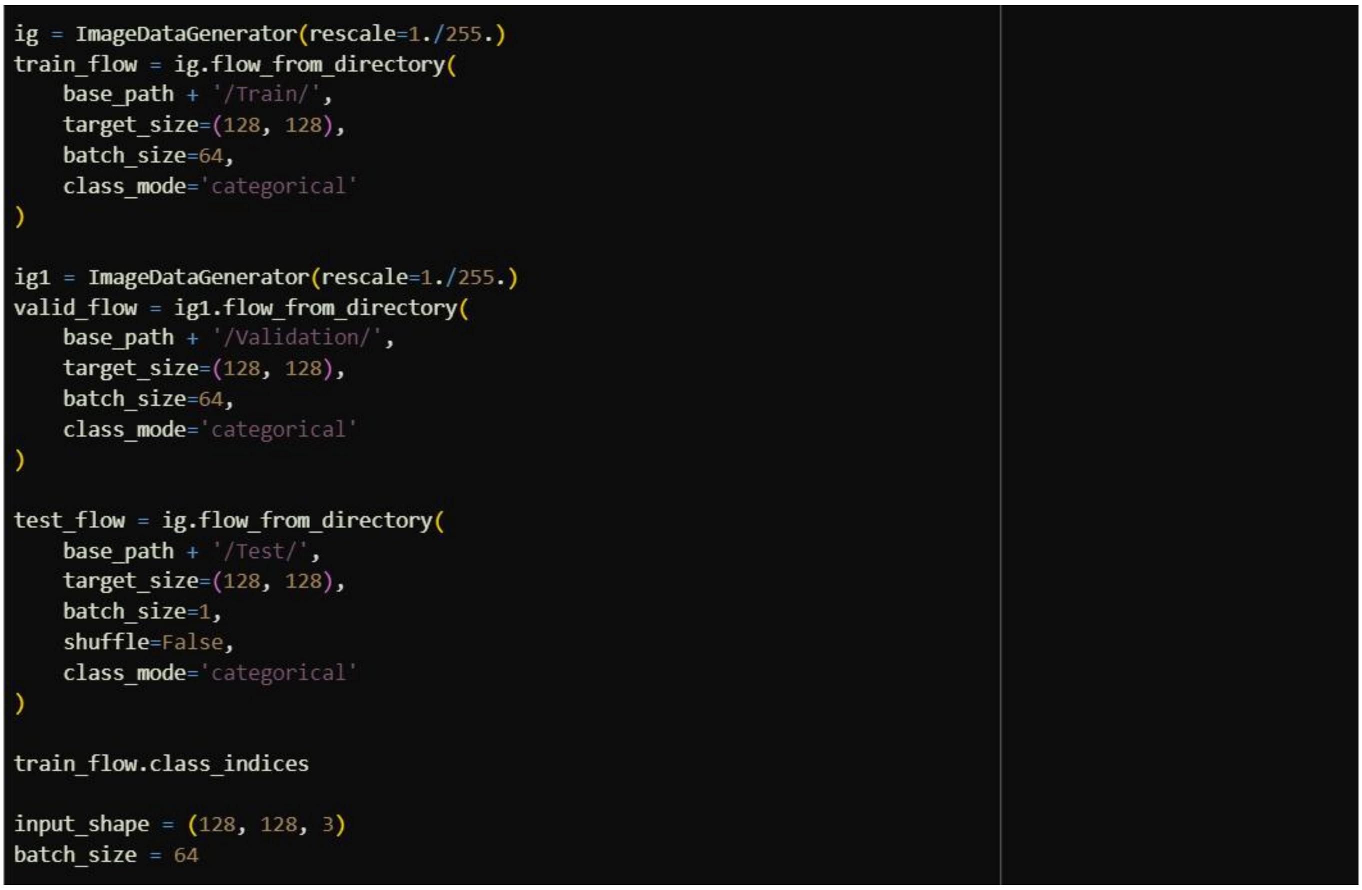
```
def plot_img(base_path, set_):
    dir_ = os.path.join(base_path, 'Train', set_)
    k = 0
    fig, ax = plt.subplots(3, 3, figsize=(10, 10))
    fig.suptitle(set_ + ' Faces')
    for j in range(3):
        for i in range(3):
            img = load_img(os.path.join(dir_, os.listdir(os.path.join(dir_))[k]))
            ax[j, i].imshow(img)
            ax[j, i].set_title("")
            ax[j, i].axis('off')
            k += 1
    plt.suptitle(set_ + ' Faces')
    plt.show()
```

The **plot_img** function is defined to visualize a grid of images from a specified directory (**base_path**) containing **real** and **fake** images under **train** subdirectory. Within the function, a 3x3 grid of subplots is created using **Matplotlib**, with each subplot representing an image. The function iterates through the directory, loading images sequentially and displaying them on the corresponding subplot. This function is useful for inspecting a subset of images from the dataset, aiding in understanding the characteristics and quality of the training data, which is crucial for model development and evaluation in image classification.





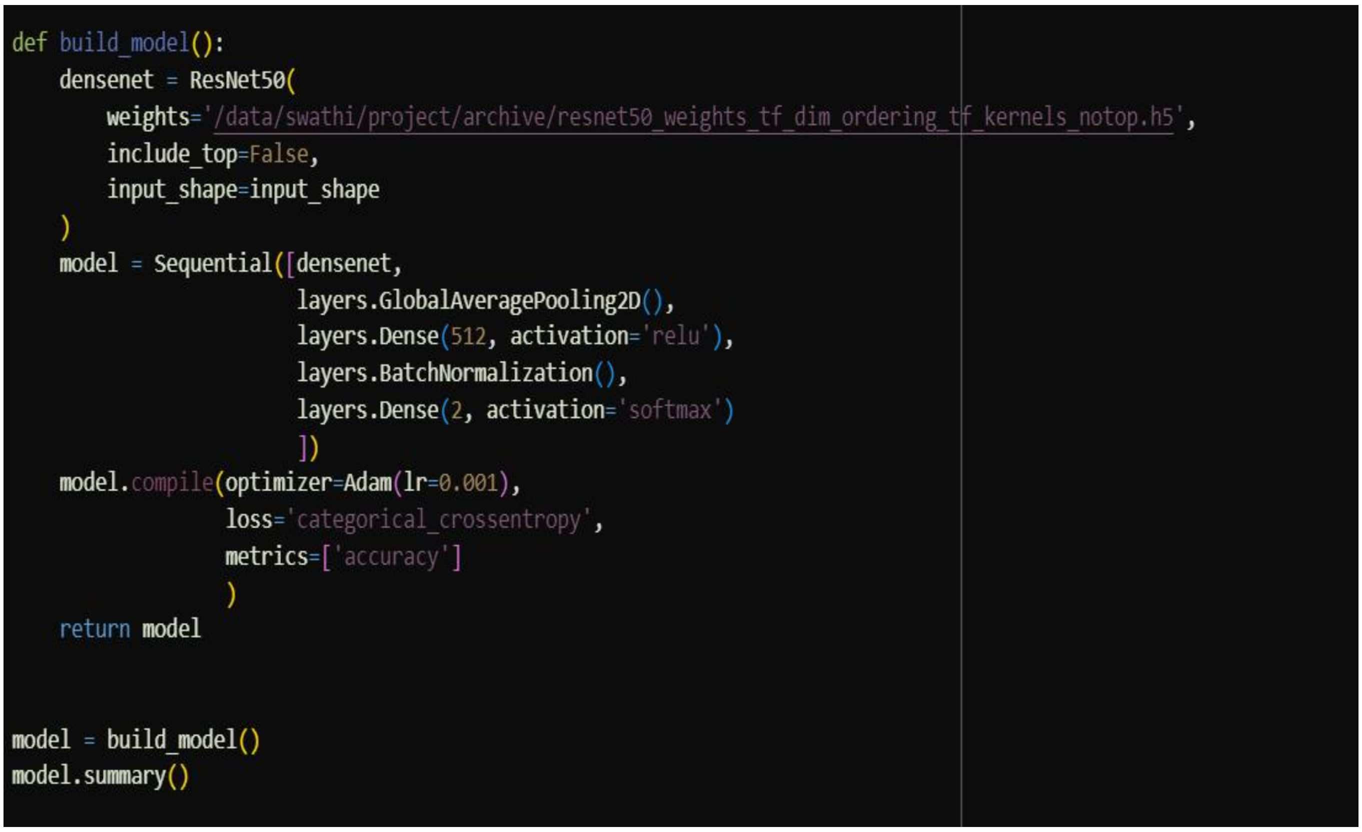
5.3. Data preprocessing



```python
ig = ImageDataGenerator(rescale=1./255.)
train_flow = ig.flow_from_directory(
    base_path + '/Train/',
    target_size=(128, 128),
    batch_size=64,
    class_mode='categorical'
)

ig1 = ImageDataGenerator(rescale=1./255.)
valid_flow = ig1.flow_from_directory(
    base_path + '/Validation/',
    target_size=(128, 128),
    batch_size=64,
    class_mode='categorical'
)

test_flow = ig.flow_from_directory(
    base_path + '/Test/',
    target_size=(128, 128),
    batch_size=1,
    shuffle=False,
    class_mode='categorical'
)

train_flow.class_indices

input_shape = (128, 128, 3)
batch_size = 64
```

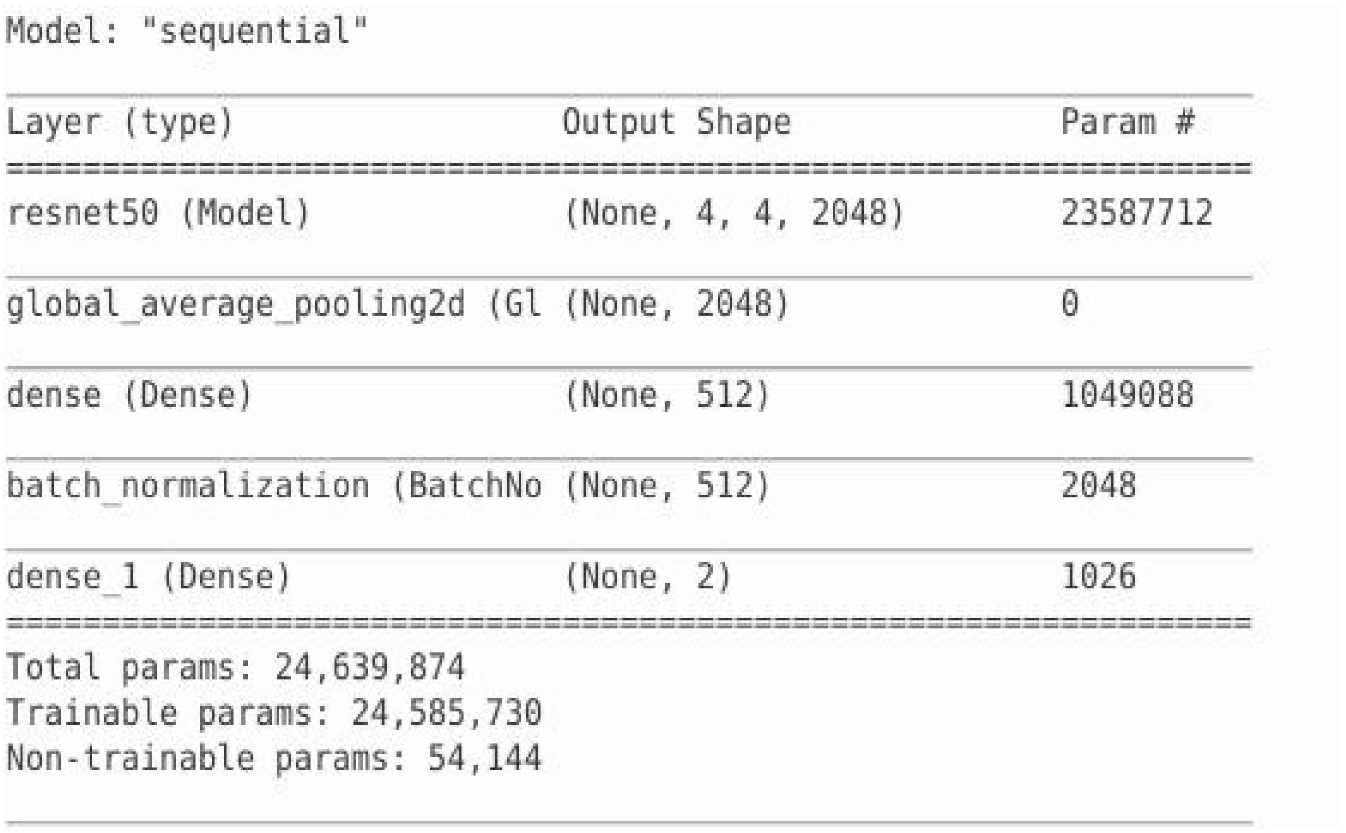
This function is used for training a deep learning model for image classification tasks using **TensorFlow** and **Keras**. It leverages data augmentation and preprocessing techniques provided by **ImageDataGenerator** to enhance model generalization. By splitting the data into training, validation, and testing sets, it ensures proper evaluation of the model's performance.

5.4. Model building



```
def build_model():
    densenet = ResNet50(
        weights='/data/swathi/project/archive/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5',
        include_top=False,
        input_shape=input_shape
    )
    model = Sequential([densenet,
                        layers.GlobalAveragePooling2D(),
                        layers.Dense(512, activation='relu'),
                        layers.BatchNormalization(),
                        layers.Dense(2, activation='softmax')
                        ])
    model.compile(optimizer=Adam(lr=0.001),
                  loss='categorical_crossentropy',
                  metrics=['accuracy']
                  )
    return model


model = build_model()
model.summary()
```

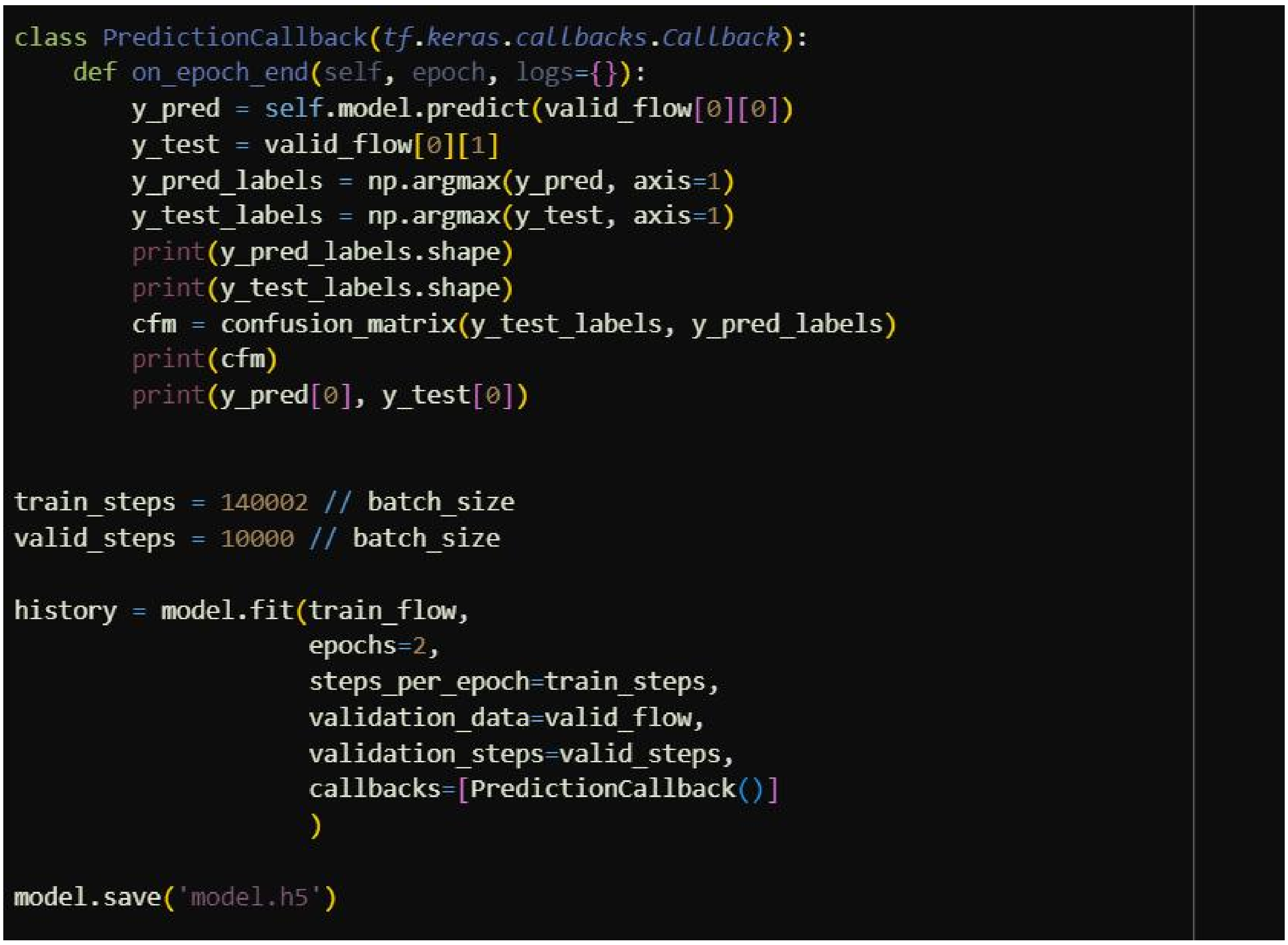
The **build_model** function constructs a **convolutional neural network (CNN)** for image classification, specifically utilizing the **ResNet50** architecture as a feature extractor. The ResNet50 model is initialized with pre-trained weights. These weights are crucial as they contain learned patterns and features from extensive training on a large dataset. The function then creates a **Sequential** model consisting of layers from ResNet50 followed by a **Global Average Pooling layer** to reduce the spatial dimensions of the feature maps. Subsequently, a fully connected layer with **512 units** and **ReLU activation** is added, followed by **Batch Normalization** to improve model stability and convergence. Finally, a **Dense layer** with **2 units** (for binary classification) and **softmax activation** is appended to output class probabilities. The model is compiled using the **Adam optimizer** with a learning rate of 0.001, **categorical cross-entropy loss** function, and **accuracy** as the evaluation metric. This architecture effectively leverages the powerful feature extraction capabilities of ResNet50 while incorporating additional layers to adapt the learned features for the specific classification task of distinguishing between two classes. The model summary provides a concise overview of its architecture, including the number of parameters and output shapes of each layer, aiding in understanding its complexity and potential performance.



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
resnet50 (Model)             (None, 4, 4, 2048)        23587712  
_________________________________________________________________
global_average_pooling2d (Gl (None, 2048)              0         
_________________________________________________________________
dense (Dense)                (None, 512)               1049088   
_________________________________________________________________
batch_normalization (BatchNo (None, 512)               2048      
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 1026      
=================================================================
Total params: 24,639,874
Trainable params: 24,585,730
Non-trainable params: 54,144
_________________________________________________________________
```

The **model.summary()** function in Keras provides a concise overview of the architecture of the neural network model that has been built. It displays information about the layers, their output shapes, and the number of parameters in each layer.

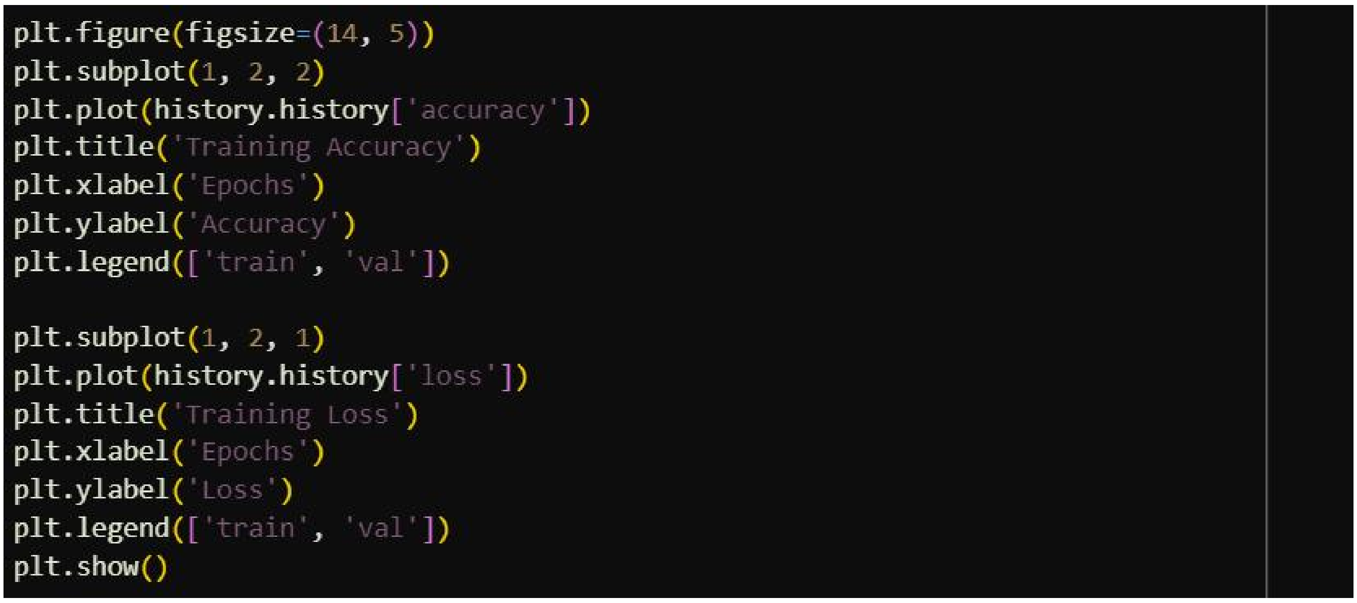
5.5. Model training



```
class PredictionCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        y_pred = self.model.predict(valid_flow[0][0])
        y_test = valid_flow[0][1]
        y_pred_labels = np.argmax(y_pred, axis=1)
        y_test_labels = np.argmax(y_test, axis=1)
        print(y_pred_labels.shape)
        print(y_test_labels.shape)
        cfm = confusion_matrix(y_test_labels, y_pred_labels)
        print(cfm)
        print(y_pred[0], y_test[0])


train_steps = 140002 // batch_size
valid_steps = 10000 // batch_size

history = model.fit(train_flow,
                    epochs=2,
                    steps_per_epoch=train_steps,
                    validation_data=valid_flow,
                    validation_steps=valid_steps,
                    callbacks=[PredictionCallback()]
                    )

model.save('model.h5')
```

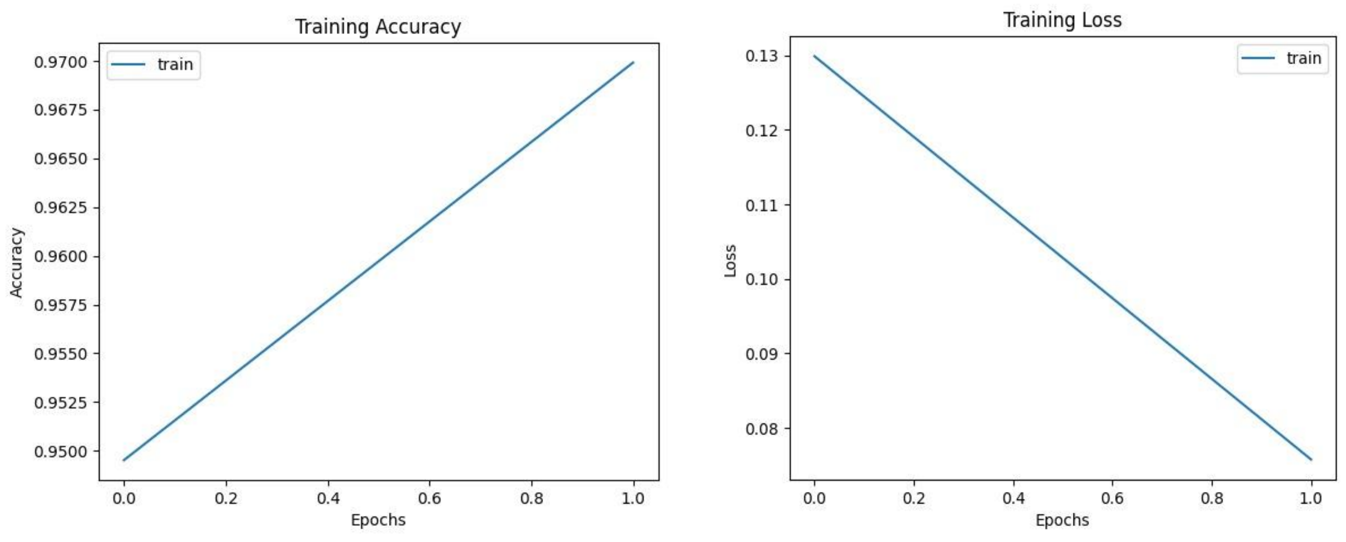
TensorFlow/Keras callback named **PredictionCallback** that is designed to be executed at the end of each training epoch. Its purpose is to perform predictions on the validation dataset and compute metrics such as confusion matrix and print predictions and actual labels for the first batch of validation data. This callback allows us to monitor the model's performance on the validation dataset during training by computing metrics like the confusion matrix and printing predictions and actual labels for the validation data. This can be helpful for diagnosing issues and evaluating the model's progress.

5.6. Model evaluation

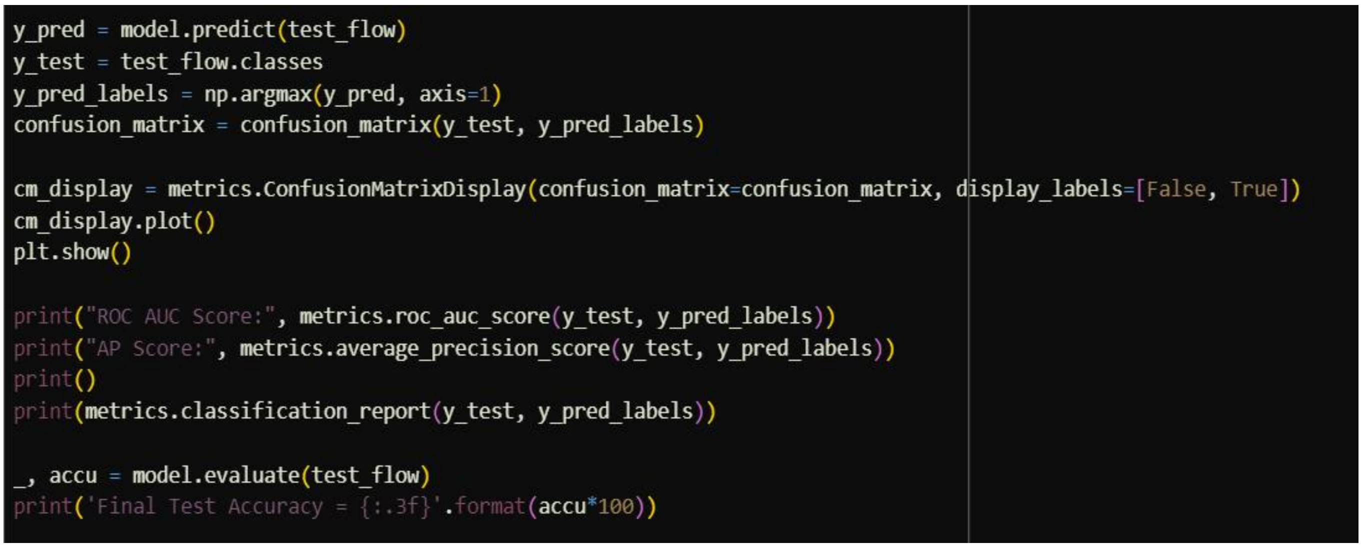


```
plt.figure(figsize=(14, 5))
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'])
plt.title('Training Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(['train', 'val'])

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'])
plt.title('Training Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(['train', 'val'])
plt.show()
```

We visualize the training progress of a deep learning model using matplotlib. It creates a figure with two subplots: one for **training accuracy** and the other for **training loss**.

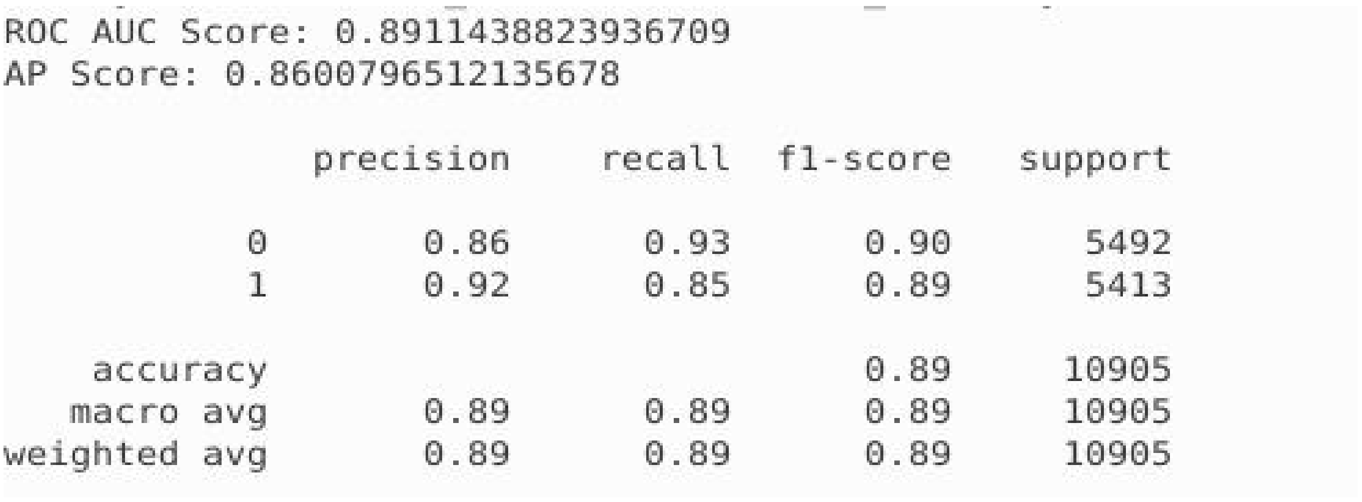


These plots provide a concise visualization of the model's performance during training, allowing users to monitor changes in accuracy and loss over successive epochs.



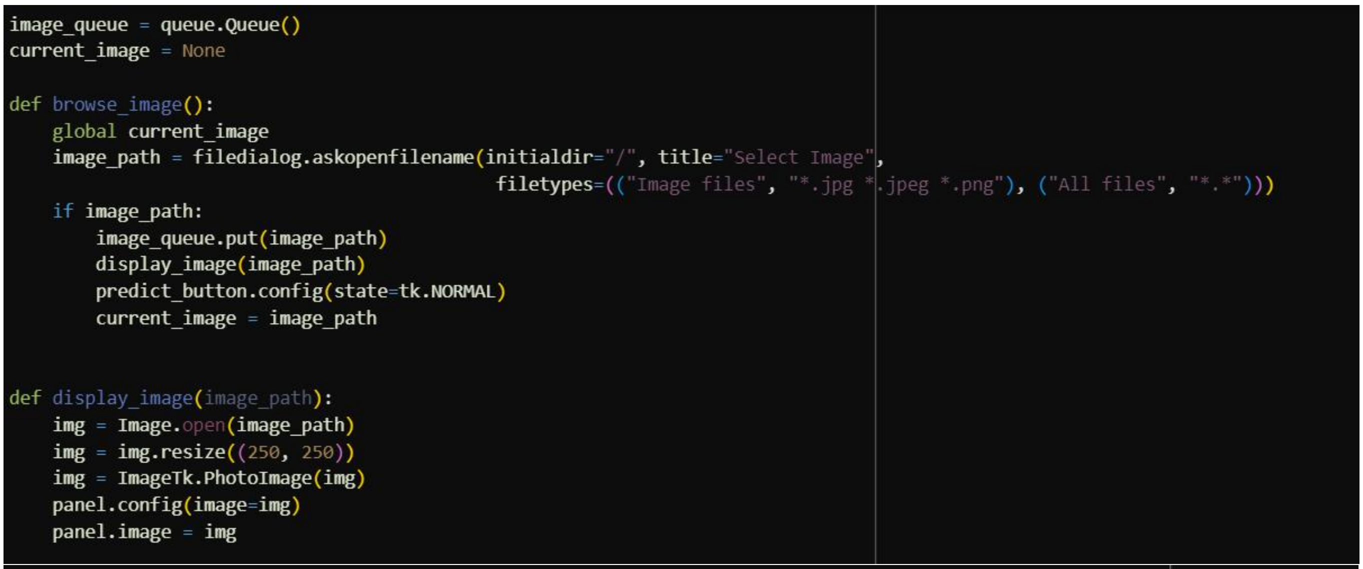
```
y_pred = model.predict(test_flow)
y_test = test_flow.classes
y_pred_labels = np.argmax(y_pred, axis=1)
confusion_matrix = confusion_matrix(y_test, y_pred_labels)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix=confusion_matrix, display_labels=[False, True])
cm_display.plot()
plt.show()

print("ROC AUC Score:", metrics.roc_auc_score(y_test, y_pred_labels))
print("AP Score:", metrics.average_precision_score(y_test, y_pred_labels))
print()
print(metrics.classification_report(y_test, y_pred_labels))

_, accu = model.evaluate(test_flow)
print('Final Test Accuracy = {:.3f}'.format(accu*100))
```

We perform a comprehensive analysis of the model's performance on the test dataset, including visualizing the **confusion matrix** and reporting various classification metrics such as **ROC AUC score**, **AP score**, and a detailed **classification report**. Additionally, we print the final test accuracy of the model.

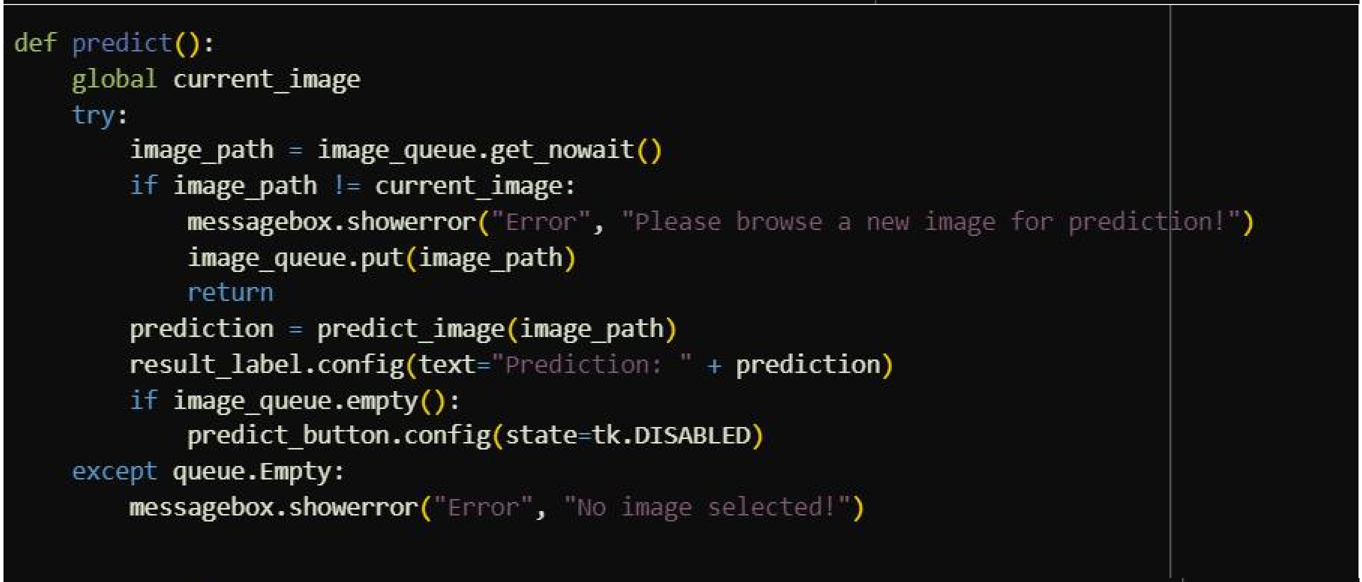


```
ROC AUC Score: 0.8911438823936709
AP Score: 0.8600796512135678

              precision    recall  f1-score   support

           0       0.86      0.93      0.90      5492
           1       0.92      0.85      0.89      5413

    accuracy                           0.89     10905
   macro avg       0.89      0.89      0.89     10905
weighted avg       0.89      0.89      0.89     10905
```

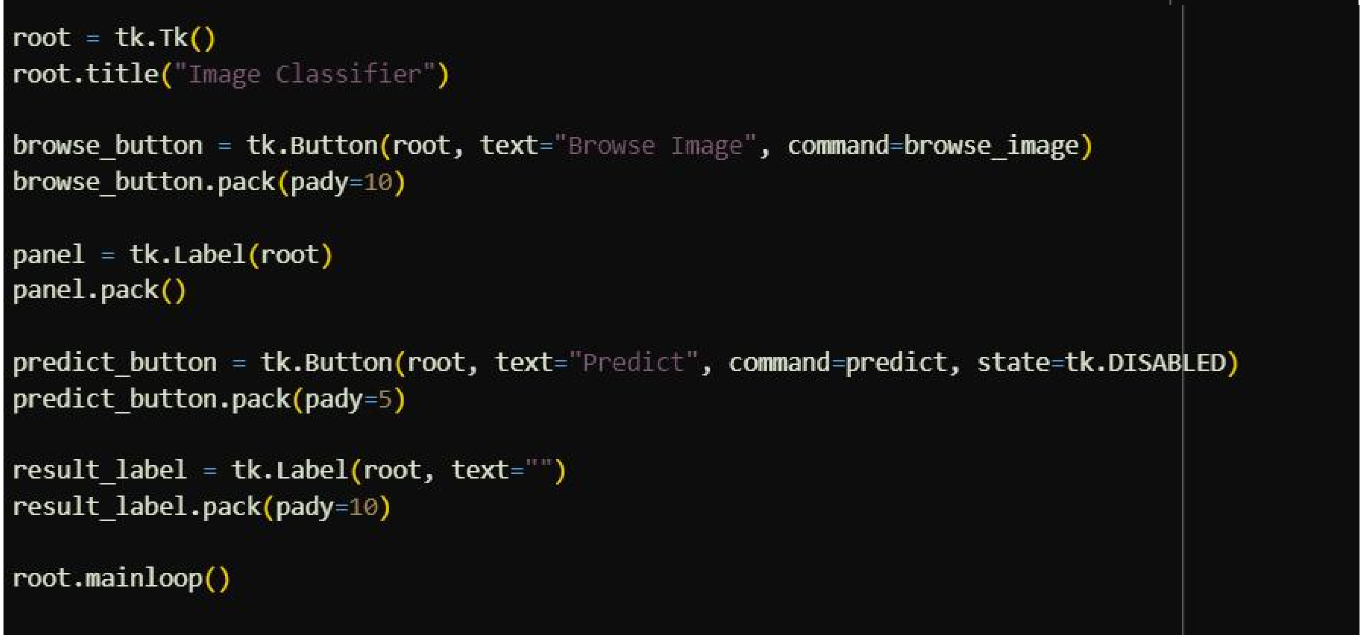
5.7. Tkinter interface



```
image_queue = queue.Queue()
current_image = None

def browse_image():
    global current_image
    image_path = filedialog.askopenfilename(initialdir="/", title="Select Image",
                                            filetypes=(("Image files", "*.jpg *.jpeg *.png"), ("All files", "*.*")))
    if image_path:
        image_queue.put(image_path)
        display_image(image_path)
        predict_button.config(state=tk.NORMAL)
        current_image = image_path


def display_image(image_path):
    img = Image.open(image_path)
    img = img.resize((250, 250))
    img = ImageTk.PhotoImage(img)
    panel.config(image=img)
    panel.image = img
```

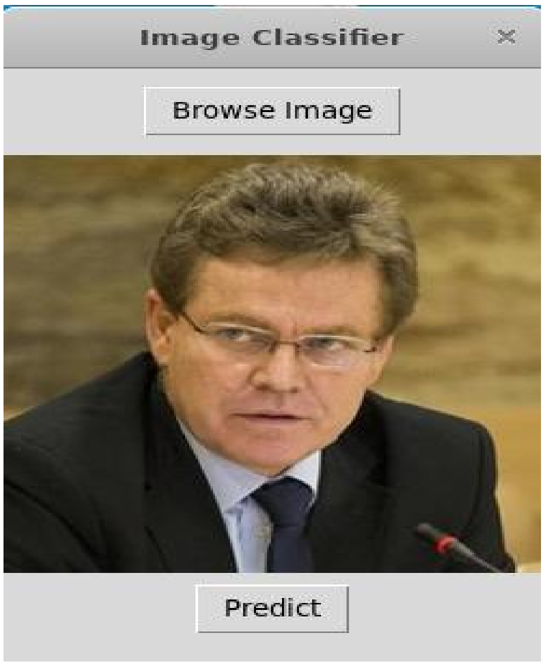


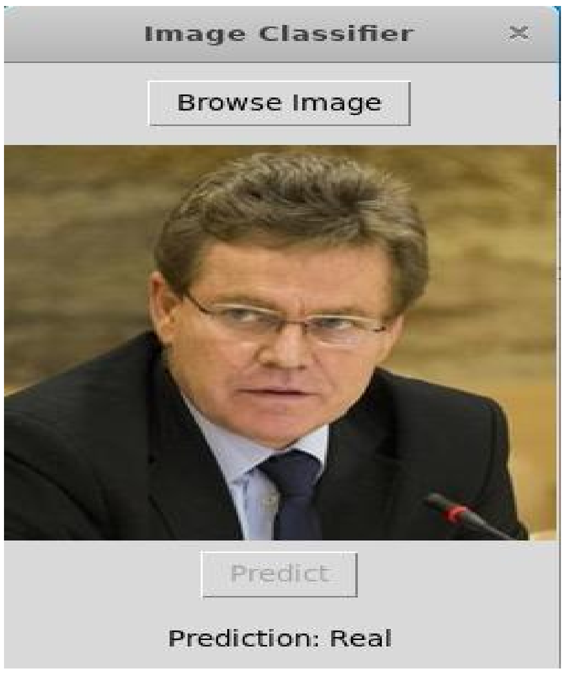
```
def predict():
    global current_image
    try:
        image_path = image_queue.get_nowait()
        if image_path != current_image:
            messagebox.showerror("Error", "Please browse a new image for prediction!")
            image_queue.put(image_path)
            return
        prediction = predict_image(image_path)
        result_label.config(text="Prediction: " + prediction)
        if image_queue.empty():
            predict_button.config(state=tk.DISABLED)
    except queue.Empty:
        messagebox.showerror("Error", "No image selected!")
```



```
root = tk.Tk()
root.title("Image Classifier")

browse_button = tk.Button(root, text="Browse Image", command=browse_image)
browse_button.pack(pady=10)

panel = tk.Label(root)
panel.pack()

predict_button = tk.Button(root, text="Predict", command=predict, state=tk.DISABLED)
predict_button.pack(pady=5)

result_label = tk.Label(root, text="")
result_label.pack(pady=10)

root.mainloop()
```

We create a simple **graphical user interface (GUI)** for an image classification application using **Tkinter**, a standard GUI toolkit for Python. The interface allows users to browse for an image file, which is then displayed in a panel within the window. Upon selecting an image, users can click the "Predict" button to initiate the prediction process using a function named **predict_image**. The prediction result is then displayed in a label below the panel. Error messages are displayed if no image is selected or if an incorrect image is chosen. Overall, the interface provides a user-friendly way to interact with the image classifier, enabling users to quickly browse images and obtain predictions with ease.





Prediction: Real

7. Conclusion

The deepfake and real classification project employing deep learning techniques presents a pivotal advancement in addressing the growing concerns surrounding manipulated media content. By leveraging sophisticated neural network architectures such as convolutional neural networks (CNNs), the project aims to accurately distinguish between authentic and artificially generated content. Through extensive training on diverse datasets encompassing both real and synthetic media samples, the model learns to identify subtle visual artifacts and inconsistencies indicative of deepfake manipulation. The utilization of transfer learning, where pre-trained models like ResNet-50 are fine-tuned on the target dataset, further enhances the model's ability to generalize across different domains and improve classification performance. Additionally, techniques such as data augmentation, ensemble learning, and adversarial training contribute to robustness against various forms of manipulation and improve the model's resilience in real-world scenarios. The project's outcomes hold significant implications for combatting the spread of misinformation, safeguarding the integrity of digital media, and empowering users with tools to discern authentic content from manipulated counterparts in an era marked by the proliferation of deepfake technology.