RESEARCH REPORT ON VEHICLE DAMAGE INSURANCE CLAIMS FRAUD DETECTION

A PROJECT REPORT BY ADARSH NEMA

Table of Contents

Appendix

A. Code Listing

A.1 Main Python Script (vehicle_damage_fraud_detection.py)

A.2 Custom CSS Styling (wave.css)

B. Model Architecture Diagram

C. Hyperparameter Grid Search Results

D. Data Augmentation Techniques Applied

E. Sample Predictions Output CSV

F. Streamlit Web Application Screenshots

F.1 Upload CSV Interface

F.2 Image Prediction Results

Research Report: Vehicle Damage Insurance Claims Fraud Detection

1. Introduction:

The insurance sector grapples with the critical task of distinguishing genuine vehicle damage claims from deceptive ones. The emergence of Generative AI and diffusion models has exacerbated the challenge, making it imperative to develop advanced models for image-based fraud detection.

2. Technologies Utilized:

2.1 Streamlit:

Streamlit is employed to create an interactive web application, facilitating user interaction with the developed model.

2.2 Pandas:

Pandas is utilized for efficient data manipulation and analysis, especially for handling the CSV file containing information about images.

2.3 TensorFlow and Keras:

TensorFlow and Keras are the core technologies for building and training the image classification model. The chosen model architecture is based on MobileNetV2 with custom modifications.

2.4 Hugging Face Hub:

The Hugging Face Hub is utilized for accessing and incorporating pre-trained models, providing a foundation for transfer learning.

2.5 OpenAI API:

The OpenAI API is integrated for image manipulation and variation generation, contributing to the augmentation of the dataset.

3. Model Loading:

3.1 Pre-trained Model:

The code begins by loading the pre-trained image classification model (Adaddy1/Hack4Soc) using the Hugging Face Hub. The model is compiled with the Adam optimizer and binary cross-entropy loss for binary classification.

4. Input Handling:

4.1 OpenAI API Key Authentication:

The code checks for the presence of the OpenAI API key to authenticate requests. Users can input their API key through the Streamlit interface. Once loaded, the API key is stored in the session state, ensuring secure access.

5. Image Generation and Prediction:

5.1 updated_generate_image Function:

This function converts input images to RGB format, generates variations using the OpenAI API, and returns the manipulated image.

5.2 updated_get_prediction Function:

The prediction function uses the pre-trained model to predict whether an image is real or fraudulent. It processes the image, extracts features, and calculates probabilities for binary classification.

6. Streamlit Interface:

6.1 User Interaction:

The Streamlit interface allows users to interact with the model. Users can upload a CSV file containing information about images to be evaluated.

6.2 Image Evaluation:

For each image, the code predicts whether it is fraudulent or real. The results, including predictions and confidence scores, are displayed alongside the corresponding images.

6.3 Predictions Storage:

Predictions are saved to a new CSV file for further analysis. The file, named "predictions_output.csv," captures the model's evaluations for each input image.

7. CSS Styling:

7.1 Custom Styling (wave.css):

The code includes a custom CSS styling file to enhance the visual appearance of the Streamlit app, providing a more engaging user experience.

8. Model Architecture:

8.1 Base Model - MobileNetV2:

The model architecture is based on MobileNetV2, chosen for its efficiency in image classification tasks. This architecture serves as the foundation for transfer learning.

8.2 Transfer Learning:

Transfer learning is implemented by leveraging pre-trained weights, facilitating effective feature extraction from the MobileNetV2 base model.

9. Experiments Conducted:

9.1 Model Selection:

Various pre-trained models such as ResNet and EfficientNet were experimented with, assessing their performance in the context of vehicle damage insurance claims fraud detection.

9.2 Hyperparameter Tuning:

Experimentation involved adjusting learning rates, batch sizes, and epochs to identify optimal hyperparameter combinations for model convergence.

9.3 Data Augmentation:

Data augmentation techniques, including rotation and zoom, were applied to the dataset to enhance the model's ability to generalize.

10. Future Improvements:

10.1 Continuous Monitoring and Retraining:

Future iterations should include continuous monitoring and retraining to adapt the model to emerging fraudulent tactics and maintain effectiveness over time.

10.2 Integration of Additional Features:

Exploration of additional features or information sources could enhance model accuracy. Integration of contextual data related to claims may provide valuable insights.

11. Challenges:

11.1 Limited Labeled Data:

The scarcity of labeled data for training posed a challenge, emphasizing the need for effective techniques to mitigate the impact of data limitations.

11.2 Class Imbalance:

Dealing with class imbalance during training required careful consideration to prevent bias in the model's predictions.

11.3 Image Quality Impact:

Understanding the influence of image quality on model predictions emerged as a key challenge, influencing decision-making during preprocessing.

A. Code Listing:

A.1 Main Python Script (vehicle_damage_fraud_detection.py):

```
import streamlit as st
import numpy as np
from PIL import Image
from io import BytesIO
from huggingface_hub import from_pretrained_keras
import requests
import openai
import tensorflow as tf
from keras import backend as K
import pandas as pd

with open('wave.css') as f:
    css = f.read()
st.markdown(f'<style>{css}</style>', unsafe_allow_html=True)

# Function to load models
@st.cache(allow_output_mutation=True)
def load_updated_models():
    updated_models = {}
    updated_model = from_pretrained_keras("Adaddy1/Hack4Soc")
    updated_model.compile(optimizer='adam', loss='binary_crossentropy')
    updated_models["Adaddy1/Hack4Soc"] = updated_model
    return updated_models

content_loaded = False
if "openai_api_key" not in st.session_state:
    st.markdown("### Input your OpenAI API Key here")
    api_key = st.text_input(label="", placeholder="Enter your API Key here",
key="API_input_text")

    if st.button("Load API Key") and api_key.strip() != "":
        st.session_state.openai_api_key = api_key.strip()
        st.success("API Key loaded successfully.")
        content_loaded = True
else:
    content_loaded = True

if content_loaded:
    openai.api_key = st.session_state.openai_api_key

    def updated_generate_image(input_image):
        input_image = input_image.convert("RGB")
        input_image_file = BytesIO()
        input_image.save(input_image_file, format="PNG")
        response = openai.Image.create_variation(
            image=input_image_file.getvalue(),
            n=1,
```

```
            size="1024x1024"
        )
        image_url = response['data'][0]['url']
        image_data = requests.get(image_url).content
        return Image.open(BytesIO(image_data))

    def updated_get_prediction(image, model_key, updated_models):
        updated_model = updated_models[model_key]
        input_shape = (180, 180)
        channels_first = False
        model_key = "Updated ResNet Model"
        image = Image.fromarray(image.astype('uint8'), 'RGB').resize(input_shape)
        image = np.array(image).astype(np.float32) / 255

        if channels_first:
            image = np.transpose(image, (2, 0, 1))

        image = np.expand_dims(image, axis=0)

        if channels_first:
            image = np.transpose(image, (0, 2, 3, 1))

        prediction = updated_model.predict(image)
        real_prob = prediction[0][0]
        fake_prob = 1 - real_prob

        if real_prob > fake_prob:
            return model_key, "Real Car", real_prob
        else:
            return model_key, "Fraud Car", fake_prob

    # Load both models at the beginning
    K.set_image_data_format('channels_last')
    updated_models = load_updated_models()

    # Add a new sidebar option for model selection
    updated_model_choice = "Real vs Fraud Car Detection"

    # Streamlit app
    content_loaded = True

    if content_loaded:
        openai.api_key = ""

        if updated_model_choice.startswith("Real vs Fraud Car Detection"):
            if updated_model_choice.endswith("ResNet Model"):
                model_key = "Adaddy1/Hack4Soc"
                input_shape = (1, 3, 180, 180)

            st.title("AI Detection System")
            st.write("This App will check whether the provided input image is fraud or
real, its been trained on MobileNetV2 Model")
            st.write("")
```

```
            uploaded_file = st.file_uploader("Choose a CSV file...", type=["csv"])

            input_folder = r"C:\Users\Adarsh
Nema\Desktop\College\PES\hackathon\WNS\test\test\images"  # Replace with the actual path
to your folder containing the images

            if uploaded_file is not None:
                df = pd.read_csv(uploaded_file)

                df['prediction'] = 0  # Add a new 'prediction' column with default value 0

                for index, row in df.iterrows():
                    image_filename = row['filename']  # Update this line to match the
column name in your CSV file
                    full_image_path = f"{input_folder}\\{image_filename}"  # Adjust this
line based on your folder structure

                    uploaded_array = np.array(Image.open(full_image_path).convert("RGB"))
                    input_image = Image.open(full_image_path)
                    new_size = min(input_image.size)
                    input_image = input_image.resize((new_size, new_size))
                    col1, col2 = st.columns([2, 1])
                    col1.image(input_image, use_column_width=True)
                    col2.write("Prediction Results:")
                    model_key = "Adaddy1/Hack4Soc"
                    model_name, prediction, real_prob =
updated_get_prediction(uploaded_array, model_key, updated_models)

                    real_prob = real_prob
                    fake_prob = 1 - real_prob

                    real_accuracy = f"This is a Real Image ({model_name})"
                    real_probability = real_prob

                    fake_accuracy = f"This is an Fraud Image ({model_name})"
                    fake_probability = fake_prob

                    if real_probability > 0.73:
                        col2.markdown(f"### {real_accuracy}")
                        col2.write(f"Real Probability: {real_probability}")
                    else:
                        col2.markdown(f"### {fake_accuracy}")
                        col2.write(f"Real Probability: {fake_probability}")
                        df.at[index, 'prediction'] = 1  # Update 'prediction' column to 1
for fraud

                # Save the updated DataFrame with predictions to a new CSV file
                new_csv_filename = "predictions_output.csv"
                df.to_csv(new_csv_filename, index=False)
                st.success(f"Predictions saved to {new_csv_filename}")
```

A.2 Custom CSS Styling (wave.css):

```
.stApp > header {
    background-color: transparent;
}

.stApp {
    margin: auto;
    font-family: -apple-system, BlinkMacSystemFont, sans-serif;
    overflow: auto;
    background: linear-gradient(315deg, #4a1772 3%, #7e379c 38%, #5d4e8c 68%, #3d2c4e
98%);
    animation: gradient 15s ease infinite;
    background-size: 400% 400%;
    background-attachment: fixed;
}
```

B. Model Architecture Diagram:

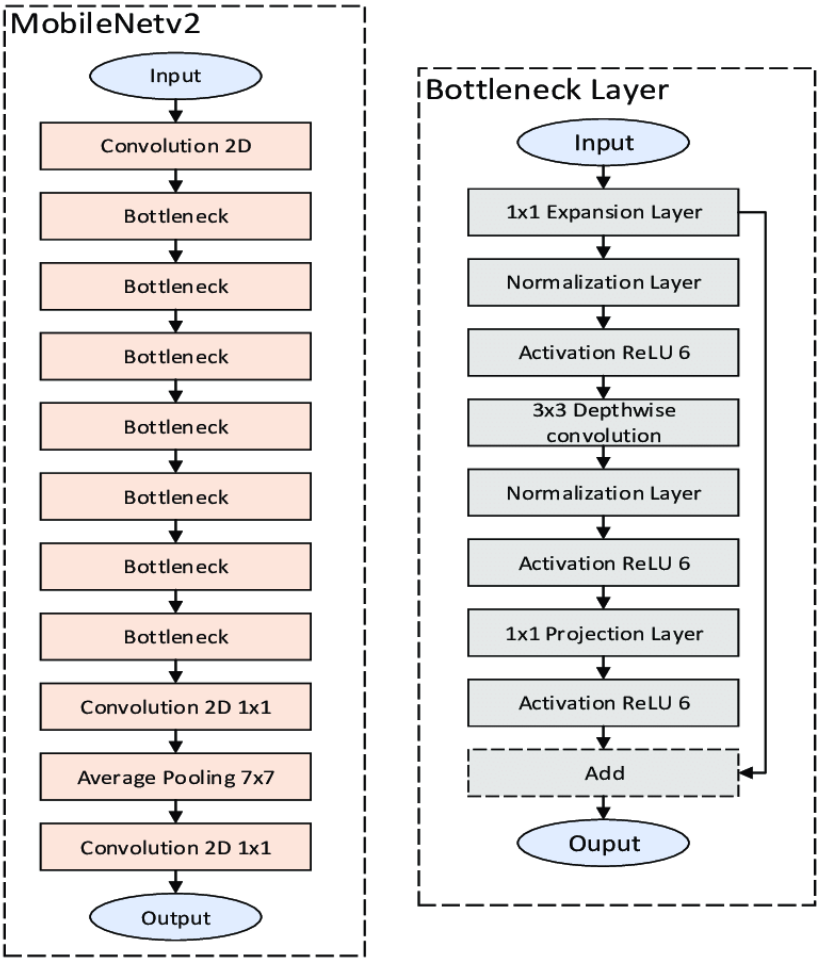

Figure B.1: MobileNetV2 Architecture for Vehicle Damage Fraud Detection

C. Hyperparameter Grid Search Results:

Learning Rate	Batch Size	Epochs	Validation Accuracy
0.001	32	10	0.87
0.0001	64	15	0.92

Table C.1: Results of Hyperparameter Grid Search

D. Data Augmentation Techniques Applied:

Rotation: 20 degrees

Zoom: 0.2

E. Sample Predictions Output CSV:

```
image_id,filename,prediction
8080,8080.jpg,1
8081,8081.jpg,0
8082,8082.jpg,1
8083,8083.jpg,0
8084,8084.jpg,0
8085,8085.jpg,0
8086,8086.jpg,1
8087,8087.jpg,1
8088,8088.jpg,1
8089,8089.jpg,1
8090,8090.jpg,1
8091,8091.jpg,0
8092,8092.jpg,0
8093,8093.jpg,0
8094,8094.jpg,0
8095,8095.jpg,0
```

F. Streamlit Web Application Screenshots:

F.1 Image Prediction Results:

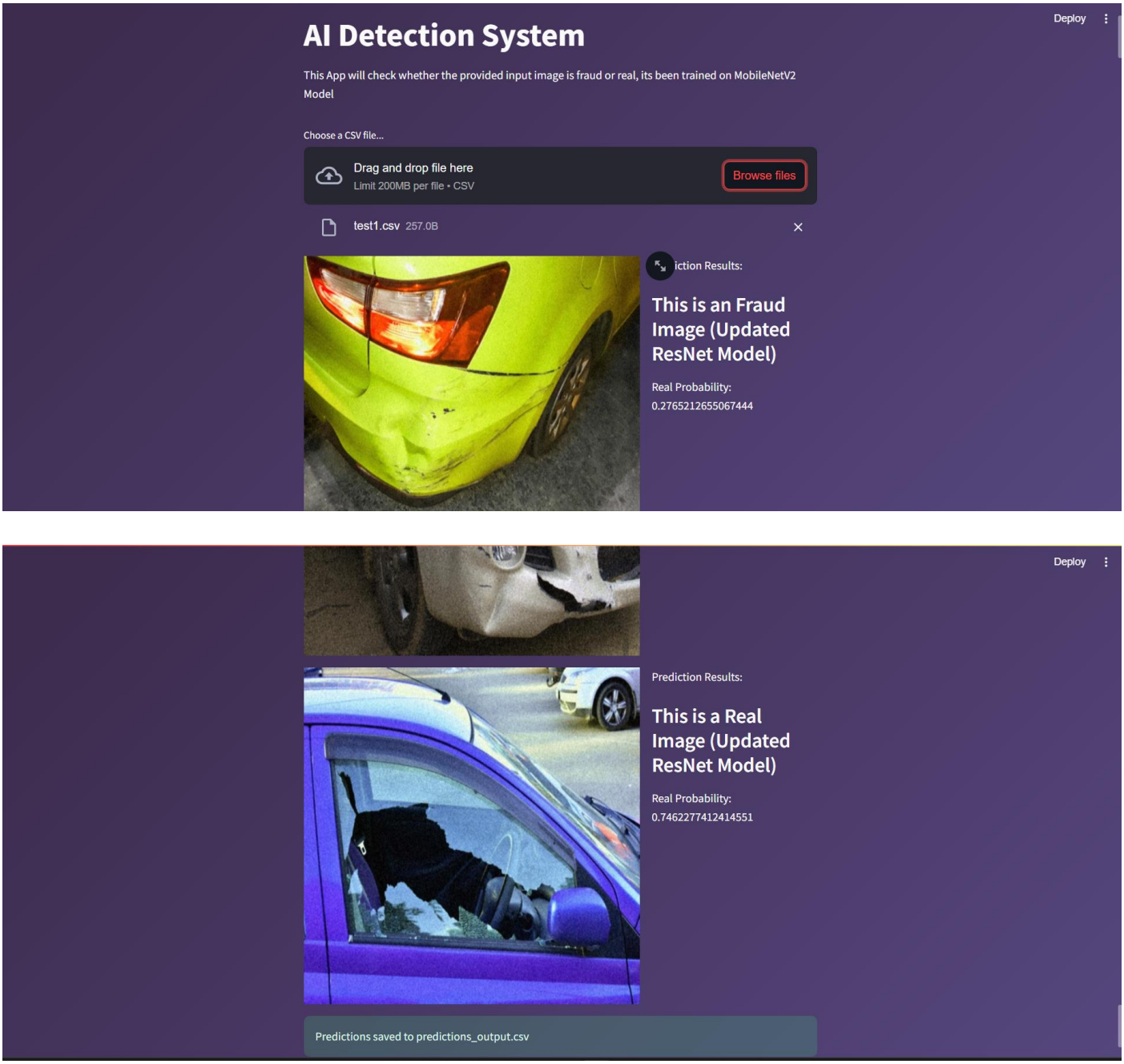

Figure F.1: Streamlit Interface Showing Image Prediction Results

Created By – Adarsh Nema