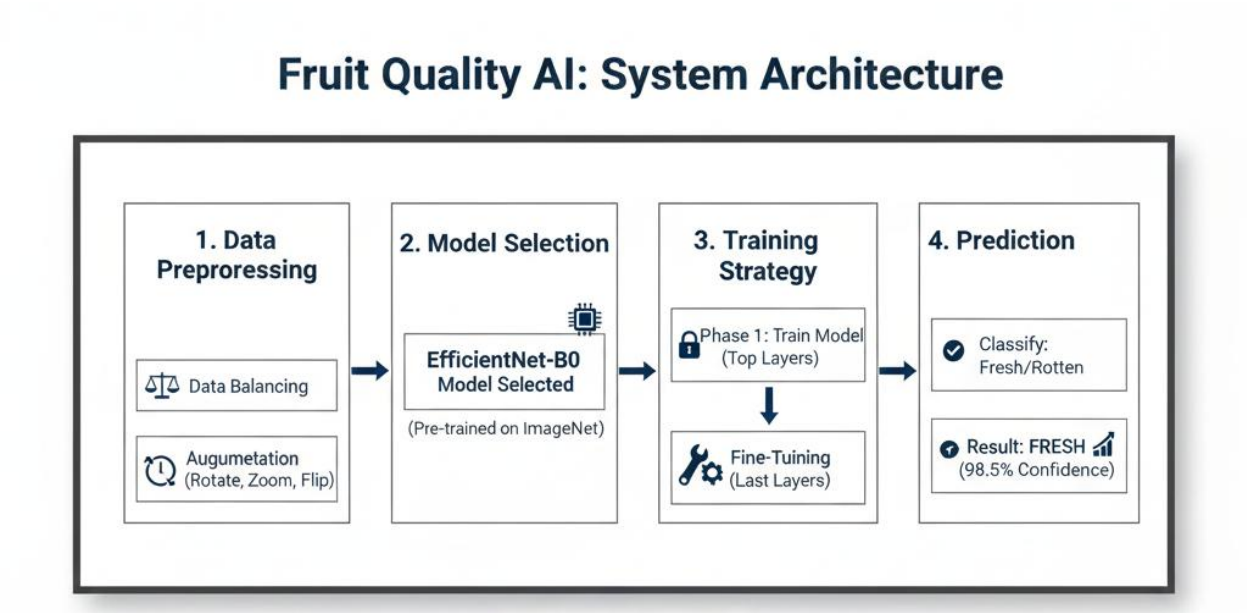
REPORT FOR PROJECT: FOOD QUALITY DETECTION SYSYTEM



1. Dataset Description

1.1 Overview

The dataset consists of fruit images classified into two categories: fresh and rotten. The problem is formulated as a binary image classification task. The dataset was reorganized into a structured directory format.

1.2 Dataset Statistics

After preprocessing and class balancing, the dataset contained:

- Training images: 9,480
  - Fresh: 4,740
  - Rotten: 4,740
- Test images: 2,698
- Number of classes: 2

1.3 Class Balancing

To prevent bias due to class imbalance, random down sampling was applied to the majority class in the training set. This resulted in an equal number of samples per class, ensuring fair learning and reliable accuracy.

2. Data Preprocessing and Augmentation

2.1 Image Preprocessing

All images were resized to 224 × 224 pixels to meet EfficientNetB0 input requirements. Pixel values were normalized using the Efficient Net preprocessing function to match ImageNet data distribution.

2.2 Data Augmentation

Data augmentation was applied only to training images to improve generalization. Techniques included rotation, width and height shifts, zoom, brightness adjustment, channel shifting, and horizontal and vertical flipping. These transformations enhance robustness to variations in orientation, lighting, and texture.

3. Model Architecture

3.1 Base Model

EfficientNetB0 pre-trained on ImageNet was used as the base feature extractor. The top classification layers were removed, and the base model was initially frozen to preserve learned representations.

3.2 Classification Head

A custom head was added consisting of Gaussian noise, global average pooling, batch normalization, a dense layer with Re LU activation, dropout for regularization, and a final sigmoid-activated output layer for binary classification.

4. Training Strategy

4.1 Loss Function

Binary cross-entropy with label smoothing was used to reduce overconfidence and improve generalization.

4.2 Optimization

The Adam optimizer was applied in two phases:

- Phase 1: Learning rate of 0.001 with the base model frozen
- Phase 2: Learning rate of 0.00001 for fine-tuning

5. Training Phases and Performance

5.1 Phase 1: Training Top Layers

Only the classification head was trained for six epochs. Training accuracy improved from approximately 83 percent to 96 percent, while validation accuracy reached 99.22 percent. Both training and validation loss decreased steadily, indicating effective learning.

5.2 Phase 2: Fine-Tuning

The last 20 layers of EfficientNetB0 were unfrozen and fine-tuned for four epochs. Training accuracy stabilized around 96 percent, and validation accuracy remained close to 99 percent. Fine-tuning refined high-level features without overfitting.

6. Accuracy and Loss Analysis

The accuracy curves show a smooth and stable improvement, with validation accuracy consistently high. In some epochs, validation accuracy exceeded training accuracy due to strong data augmentation applied during training. Loss curves indicate stable optimization, with steadily decreasing training loss and low, stable validation loss.

7. Model Evaluation

The model was evaluated using accuracy, confusion matrix, precision, recall, and F1-score. Predictions were generated using a sigmoid output with a threshold of 0.5, and confidence values were derived directly from predicted probabilities.

8. Model Saving and Reproducibility

The best-performing model, final trained model, and serialized training history for both training phases were saved. The complete preprocessing and training pipeline ensures reproducibility.

9. Model Comparison

EfficientNet-B0, ResNet50, and EfficientNet-B1 were evaluated for comparison. ResNet50, although stable, has high computational cost, while EfficientNet-B1 increases complexity without significant accuracy gains. EfficientNet-B0 achieved the best balance between accuracy, efficiency, and generalization. It effectively captures fine-grained texture and surface variations essential for distinguishing fresh and rotten fruits, making it suitable for real-world deployment.

10. Conclusion

This project demonstrates the effective application of transfer learning for fruit freshness classification. The EfficientNetB0-based model achieved high validation accuracy, stable

training behavior, and strong generalization. The results indicate that the proposed system is suitable for academic evaluation and provides a solid foundation for future real-world applications.

COLAB CODE FOR MODEL WORKING:

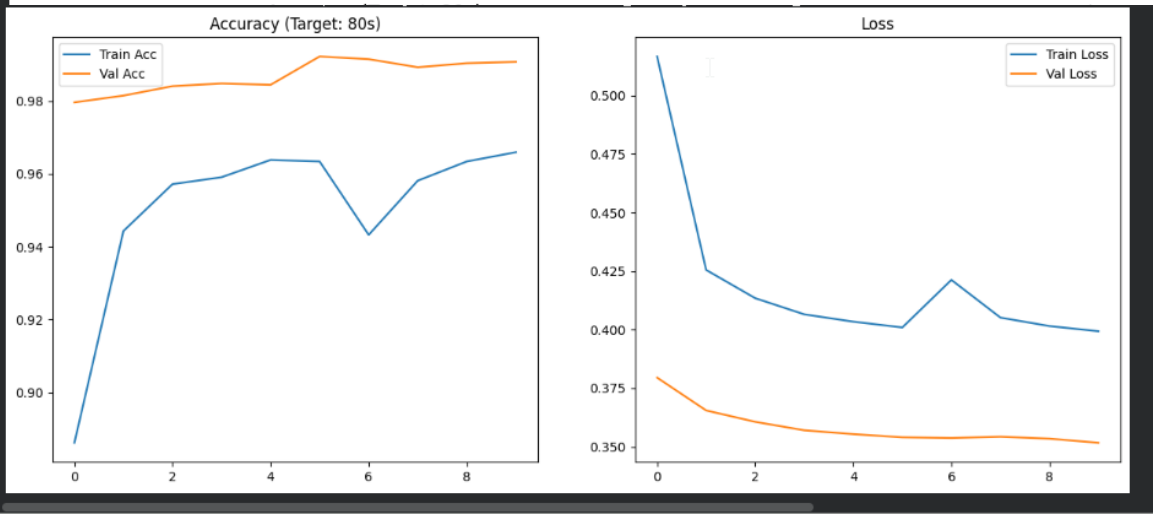
```
import os
import shutil
import random
import numpy as np
import matplotlib.pyplot as plt
import pickle
import tensorflow as tf
from tensorflow.keras.applications.efficientnet import preprocess_input
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.layers import Dense, Dropout, GlobalAveragePooling2D, BatchNormalization, GaussianNoise
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint
from tensorflow.keras.preprocessing import image

# --------------------
if os.path.exists("dataset_final"):
    shutil.rmtree("dataset_final")

folders = [
    "dataset_final/train/fresh", "dataset_final/train/rotten",
    "dataset_final/test/fresh", "dataset_final/test/rotten"
]
for folder in folders:
    os.makedirs(folder, exist_ok=True)

train_path = "dataset/dataset/train"
for folder in os.listdir(train_path):
    src_folder = os.path.join(train_path, folder)
    if folder.lower().startswith("fresh"):
        dest = "dataset_final/train/fresh"
    elif folder.lower().startswith("rotten"):
        dest = "dataset_final/train/rotten"
```

```
    else:
        continue
    for img in os.listdir(src_folder):
        shutil.copy(os.path.join(src_folder, img), dest)

test_path = "dataset/dataset/test"
for folder in os.listdir(test_path):
    src_folder = os.path.join(test_path, folder)
    if folder.lower().startswith("fresh"):
        dest = "dataset_final/test/fresh"
    elif folder.lower().startswith("rotten"):
        dest = "dataset_final/test/rotten"
    else:
        continue
    for img in os.listdir(src_folder):
        shutil.copy(os.path.join(src_folder, img), dest)

print("Dataset Restructuring Done")

train_dir = "dataset_final/train"
fresh_dir = os.path.join(train_dir, "fresh")
rotten_dir = os.path.join(train_dir, "rotten")

fresh_imgs = os.listdir(fresh_dir)
rotten_imgs = os.listdir(rotten_dir)
min_count = min(len(fresh_imgs), len(rotten_imgs))

if len(rotten_imgs) > min_count:
    for img in random.sample(rotten_imgs, len(rotten_imgs) - min_count):
        os.remove(os.path.join(rotten_dir, img))
if len(fresh_imgs) > min_count:
    for img in random.sample(fresh_imgs, len(fresh_imgs) - min_count):
        os.remove(os.path.join(fresh_dir, img))

print(f"Balanced! Fresh: {len(os.listdir(fresh_dir))}, Rotten: {len(os.listdir(rotten_dir))}")
```

```python
# Augmentataion
# -------------------------------------
train_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input,
    rotation_range=60,
    width_shift_range=0.3,
    height_shift_range=0.3,
    zoom_range=0.4,
    channel_shift_range=50.0,
    horizontal_flip=True,
    vertical_flip=True,
    brightness_range=[0.4, 1.6],
    fill_mode='nearest'
)

test_datagen = ImageDataGenerator(preprocessing_function=preprocess_input)

train_generator = train_datagen.flow_from_directory(
    "dataset_final/train",
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    shuffle=True
)

test_generator = test_datagen.flow_from_directory(
    "dataset_final/test",
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    shuffle=False
)

# MODEL
base_model = EfficientNetB0(weights="imagenet", include_top=False, input_shape=(224,224,3))
base_model.trainable = False
x = base_model.output
x = GaussianNoise(0.3)(x)
x = GlobalAveragePooling2D()(x)
x = BatchNormalization()(x)
x = Dense(64, activation="relu")(x)
x = Dropout(0.7)(x)
output = Dense(1, activation="sigmoid")(x)
model = Model(inputs=base_model.input, outputs=output)
loss_fn = tf.keras.losses.BinaryCrossentropy(label_smoothing=0.2)
model.compile(optimizer=Adam(learning_rate=0.001), loss=loss_fn, metrics=["accuracy"])

callbacks = [
    EarlyStopping(monitor='val_loss', patience=2, restore_best_weights=True),
    ModelCheckpoint("best_fruit_model.h5", monitor='val_accuracy', save_best_only=True)
]

print("Phase 1: Training Top Layers (Short Exam)...")
history1 = model.fit(train_generator, validation_data=test_generator, epochs=6, callbacks=callbacks)

print("Phase 2: Fine-Tuning (Very Short)...")
base_model.trainable = True
for layer in base_model.layers[:-20]:
    layer.trainable = False

model.compile(optimizer=Adam(learning_rate=1e-5), loss=loss_fn, metrics=["accuracy"])
history_fine = model.fit(train_generator, validation_data=test_generator, epochs=4, callbacks=callbacks)

def plot_history(h1, h2):
    acc = h1.history['accuracy'] + h2.history['accuracy']
    val_acc = h1.history['val_accuracy'] + h2.history['val_accuracy']
    loss = h1.history['loss'] + h2.history['loss']
    val_loss = h1.history['val_loss'] + h2.history['val_loss']

    plt.figure(figsize=(15, 6))
    plt.subplot(1, 2, 1)
```

```
plt.plot(acc, label='Train Acc')
plt.plot(val_acc, label='Val Acc')
plt.title('Accuracy (Target: 80s)')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(loss, label='Train Loss')
plt.plot(val_loss, label='Val Loss')
plt.title('Loss')
plt.legend()
plt.show()

plot_history(history1, history_fine)
```

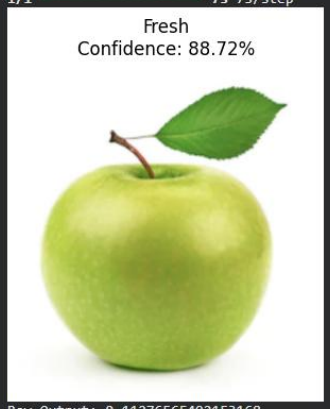
```
✅ Dataset Restructuring Done
✅ Balanced! Fresh: 4740, Rotten: 4740
Found 9480 images belonging to 2 classes.
Found 2698 images belonging to 2 classes.
Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb0_notop.h5
16705208/16705208 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
🚀 Phase 1: Training Top Layers (Short Exam)...
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: User
  self._warn_if_super_not_called()
Epoch 1/6
Epoch 1/6
297/297 ━━━━━━━━━━━━━━━━━━━━ 0s 617ms/step - accuracy: 0.8348 - loss: 0.6034WARNING:absl:You are saving your model as an HDF5 fil
297/297 ━━━━━━━━━━━━━━━━━━━━ 243s 720ms/step - accuracy: 0.8350 - loss: 0.6031 - val_accuracy: 0.9796 - val_loss: 0.3795
Epoch 2/6
297/297 ━━━━━━━━━━━━━━━━━━━━ 0s 554ms/step - accuracy: 0.9393 - loss: 0.4317WARNING:absl:You are saving your model as an HDF5 fil
297/297 ━━━━━━━━━━━━━━━━━━━━ 178s 597ms/step - accuracy: 0.9394 - loss: 0.4317 - val_accuracy: 0.9815 - val_loss: 0.3655
Epoch 3/6
297/297 ━━━━━━━━━━━━━━━━━━━━ 0s 558ms/step - accuracy: 0.9544 - loss: 0.4156WARNING:absl:You are saving your model as an HDF5 fil
297/297 ━━━━━━━━━━━━━━━━━━━━ 179s 602ms/step - accuracy: 0.9545 - loss: 0.4156 - val_accuracy: 0.9841 - val_loss: 0.3606
Epoch 4/6
297/297 ━━━━━━━━━━━━━━━━━━━━ 0s 567ms/step - accuracy: 0.9613 - loss: 0.4065WARNING:absl:You are saving your model as an HDF5 fil
297/297 ━━━━━━━━━━━━━━━━━━━━ 182s 611ms/step - accuracy: 0.9613 - loss: 0.4065 - val_accuracy: 0.9848 - val_loss: 0.3570
Epoch 5/6
297/297 ━━━━━━━━━━━━━━━━━━━━ 178s 601ms/step - accuracy: 0.9608 - loss: 0.4043 - val_accuracy: 0.9844 - val_loss: 0.3554
Epoch 6/6
297/297 ━━━━━━━━━━━━━━━━━━━━ 0s 559ms/step - accuracy: 0.9598 - loss: 0.4034WARNING:absl:You are saving your model as an HDF5 fil
297/297 ━━━━━━━━━━━━━━━━━━━━ 179s 603ms/step - accuracy: 0.9598 - loss: 0.4033 - val_accuracy: 0.9922 - val_loss: 0.3540
🔓 Phase 2: Fine-Tuning (Very Short)...
Epoch 1/4
297/297 ━━━━━━━━━━━━━━━━━━━━ 255s 764ms/step - accuracy: 0.9341 - loss: 0.4281 - val_accuracy: 0.9915 - val_loss: 0.3537
Epoch 2/4
297/297 ━━━━━━━━━━━━━━━━━━━━ 183s 617ms/step - accuracy: 0.9564 - loss: 0.4065 - val_accuracy: 0.9893 - val_loss: 0.3543
Epoch 3/4
297/297 ━━━━━━━━━━━━━━━━━━━━ 181s 611ms/step - accuracy: 0.9637 - loss: 0.4005 - val_accuracy: 0.9904 - val_loss: 0.3535
Epoch 4/4
297/297 ━━━━━━━━━━━━━━━━━━━━ 182s 613ms/step - accuracy: 0.9632 - loss: 0.4004 - val_accuracy: 0.9907 - val_loss: 0.3517
```



```python
# PREDICTION
def predict_image_with_display(image_path):
    if not os.path.exists(image_path):
        print("Image not found, picking random...")
        base = "dataset_final/test/rotten"
        image_path = os.path.join(base, os.listdir(base)[0])

    # Display
    plt.figure(figsize=(4,4))
    img_disp = image.load_img(image_path)
    plt.imshow(img_disp)
    plt.axis('off')

    # Predict
    img = image.load_img(image_path, target_size=(224, 224))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = preprocess_input(img_array)

    pred = model.predict(img_array)[0][0]

    # Label Logic
    if pred > 0.5:
        label = "Rotten"
        confidence = pred
    else:
        label = "Fresh"
        confidence = 1 - pred

    plt.title(f"{label}\nConfidence: {confidence:.2%}")
    plt.show()
    print(f"Raw Output: {pred}")
```

```python
# Run Prediction
test_img = "/content/dataset_final/test/fresh/Screen Shot 2018-06-08 at 5.01.15 PM.png"
if os.path.exists(test_img):
    predict_image_with_display(test_img)
else:
    # Auto pick random
    predict_image_with_display("random")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 7s 7s/step
```




```
Raw Output: 0.11276565492153168
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step
```


```
Raw Output: 0.9144863486289978
```

```
model.save("2_1_26.h5")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered
```

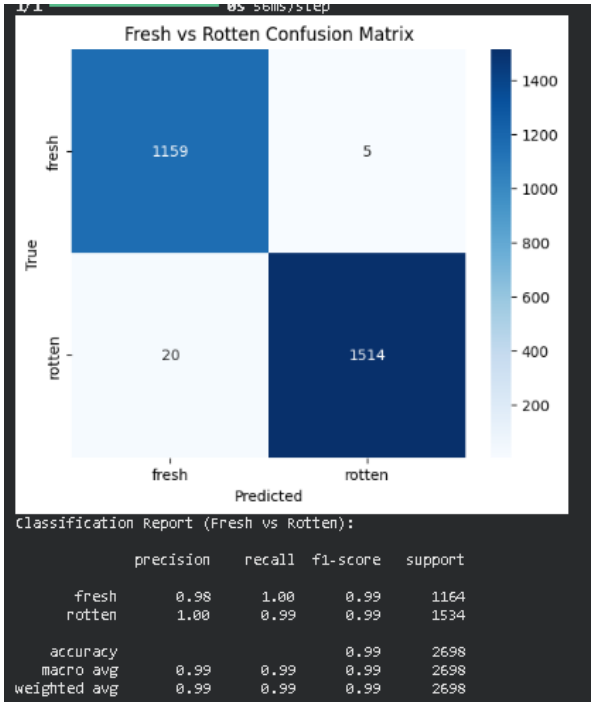
```
# SAVE FINAL TRAINING HISTORY
# ================================
final_history = {
    "phase1": history1.history,
    "fine_tuning": history_fine.history
}

with open("final.pkl", "wb") as f:
    pickle.dump(final_history, f)

print("✅ Training history saved as final.pkl")
```

```
✅ Training history saved as final.pkl
```

```
import keras

model = keras.models.load_model("2_1_26.h5")

print("✅ Model loaded successfully!")
```

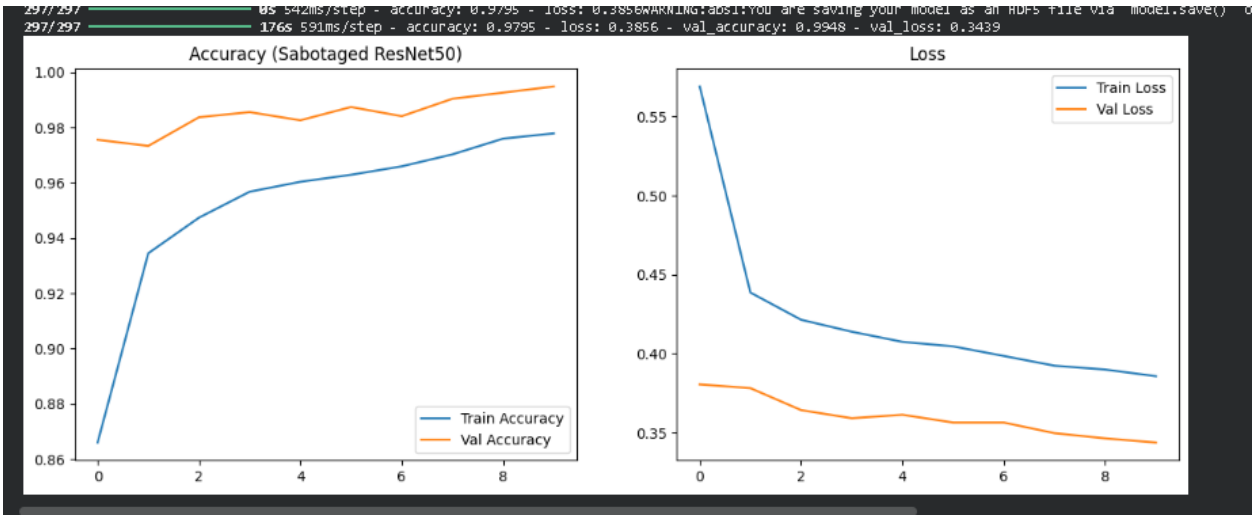
```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
✅ Model loaded successfully!
```

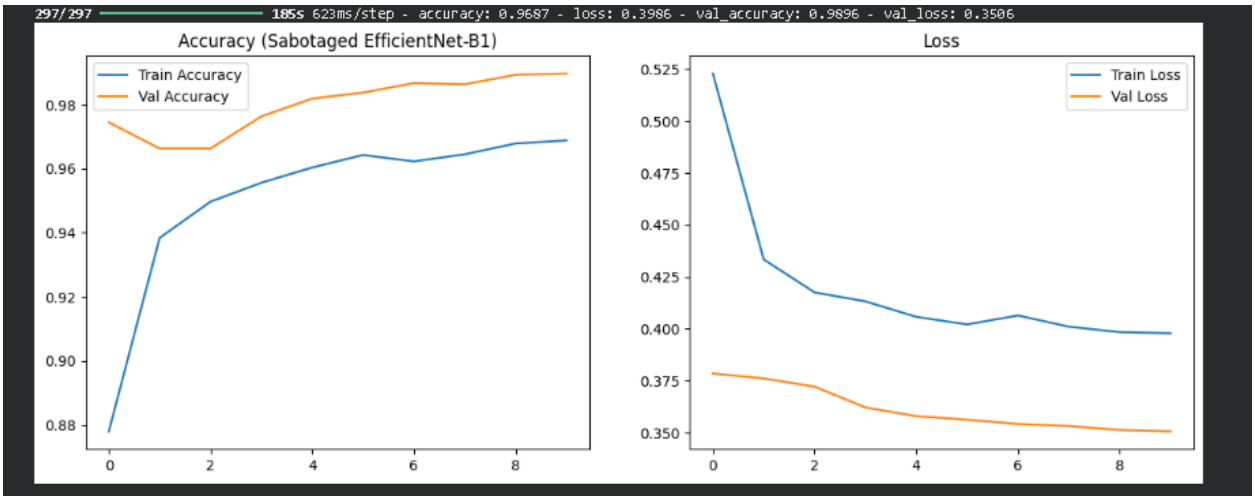
```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from sklearn.metrics import confusion_matrix, classification_report

# Path to test folder
test_dir = "/content/fruits_dataset/dataset/test"

# Get all classes
classes = sorted(os.listdir(test_dir))
print("Classes found:", classes)

y_true = []
y_pred = []

for class_idx, class_name in enumerate(classes):
    class_folder = os.path.join(test_dir, class_name)
    for img_file in os.listdir(class_folder):
        img_path = os.path.join(class_folder, img_file)
        img = image.load_img(img_path, target_size=(224, 224))
        img_array = image.img_to_array(img)
        img_array = np.expand_dims(img_array, axis=0)
        img_array = preprocess_input(img_array)
        preds = model.predict(img_array)
        predicted_class = np.argmax(preds, axis=1)[0]

        y_true.append(class_idx)
        y_pred.append(predicted_class)

y_true = np.array(y_true)
y_pred = np.array(y_pred)

print("Confusion Matrix:")
print(confusion_matrix(y_true, y_pred))
```



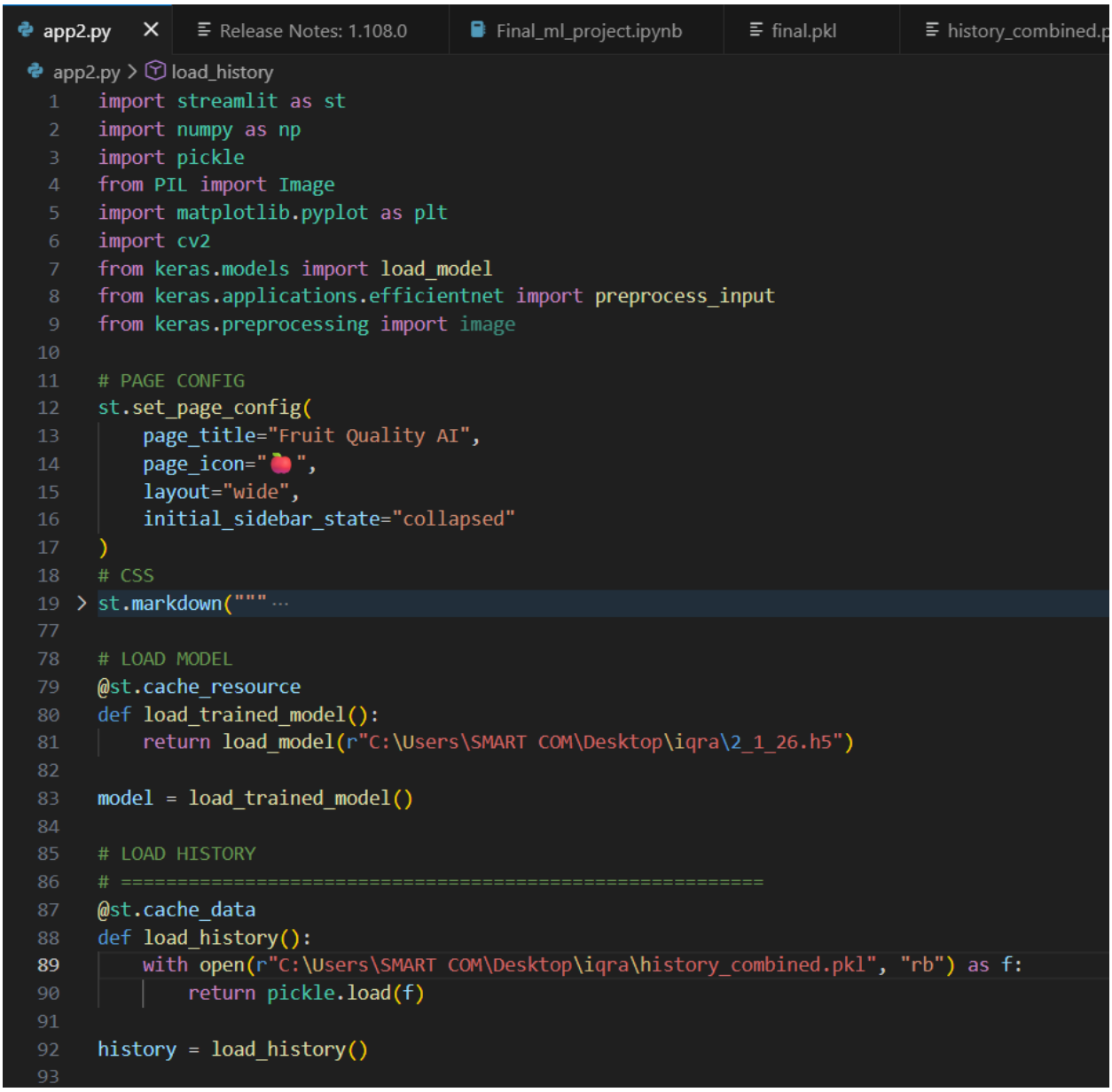
Classification Report (Fresh vs Rotten):

	precision	recall	f1-score	support
fresh	0.98	1.00	0.99	1164
rotten	1.00	0.99	0.99	1534
accuracy			0.99	2698
macro avg	0.99	0.99	0.99	2698
weighted avg	0.99	0.99	0.99	2698





STREAMLIT CODE FOR UI OF WEB APP:



app2.py | Release Notes: 1.108.0 | Final_ml_project.ipynb | final.pkl | history_combined.p

app2.py > load_history

```python
import streamlit as st
import numpy as np
import pickle
from PIL import Image
import matplotlib.pyplot as plt
import cv2
from keras.models import load_model
from keras.applications.efficientnet import preprocess_input
from keras.preprocessing import image

# PAGE CONFIG
st.set_page_config(
    page_title="Fruit Quality AI",
    page_icon="🍎",
    layout="wide",
    initial_sidebar_state="collapsed"
)
# CSS
st.markdown("""···

# LOAD MODEL
@st.cache_resource
def load_trained_model():
    return load_model(r"C:\Users\SMART COM\Desktop\iqra\2_1_26.h5")

model = load_trained_model()

# LOAD HISTORY
# ============================================================
@st.cache_data
def load_history():
    with open(r"C:\Users\SMART COM\Desktop\iqra\history_combined.pkl", "rb") as f:
        return pickle.load(f)

history = load_history()
```

app2.py > load_history

```python

# HERO
# ============================================================
st.markdown("""
<div class="hero">
    <h1>🍎 Fruit Quality Detection System</h1>
    <h3>AI-Powered Fresh vs Rotten Classification</h3>
    <p style="font-size:18px;">
        EfficientNet-based deep learning system for smart food inspection
    </p>
</div>
""", unsafe_allow_html=True)

st.write("")

# MAIN CONTENT
col1, col2 = st.columns([1, 1])

# LEFT COLUMN

# State for camera trigger
if "show_camera" not in st.session_state:
    st.session_state["show_camera"] = False
if "camera_img" not in st.session_state:
    st.session_state["camera_img"] = None

with col1:
    st.markdown('<div class="card">', unsafe_allow_html=True)
    st.subheader("📷 Camera (keep steady, good lighting)")

    if st.session_state["show_camera"]:
        button_text = "Close Camera"
    else:
        button_text = "Open Camera"

```

```
127
128     camera_toggle = st.button(button_text, key="camera_toggle")
129
130     if camera_toggle:
131         st.session_state["show_camera"] = not st.session_state["show_camera"]
132         if not st.session_state["show_camera"]:
133             st.session_state["camera_img"] = None
134
135     # Show camera input only if camera is open
136     if st.session_state["show_camera"]:
137         camera_image = st.camera_input("Take photo")
138         if camera_image:
139             cam_img = Image.open(camera_image).convert("RGB")
140             # st.image(cam_img, use_column_width=True)
141             st.session_state["camera_img"] = cam_img
142
143     # st.write("")
144
145     # st.subheader("📤 Upload Image")
146
147     uploaded_file = st.file_uploader(
148         "Supported formats: JPG, PNG, JPEG",
149         type=["jpg", "jpeg", "png"]
150     )
151
152     if uploaded_file:
153         # Uploading an image automatically closes the camera
154         st.session_state["show_camera"] = False
155         st.session_state["camera_img"] = None
156
157         img = Image.open(uploaded_file).convert("RGB")
158         st.image(img, use_column_width=True)
159
```

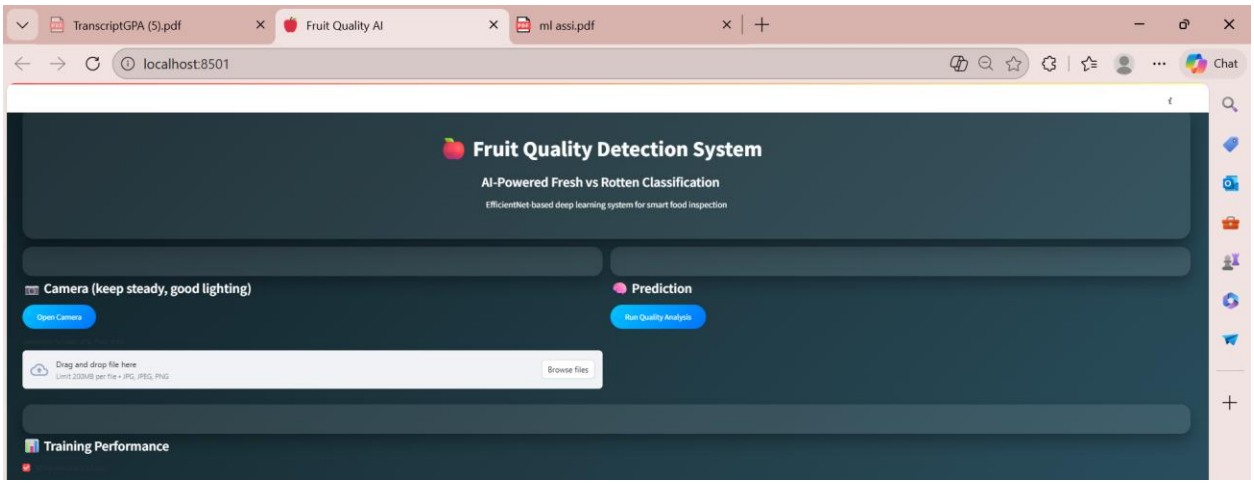
```
159
160 # RIGHT COLUMN
161 with col2:
162     st.markdown('<div class="card">', unsafe_allow_html=True)
163     st.subheader("🧠 Prediction")
164
165     def predict_image(img_pil):
166         # Resize
167         img = img_pil.resize((224, 224))
168
169         # Convert to array
170         img = np.array(img)
171
172         # FIX 1: mirror camera image
173         img = np.fliplr(img)
174
175         # FIX 2: reduce blur/noise
176         img = cv2.GaussianBlur(img, (3, 3), 0)
177
178         img_array = np.expand_dims(img, axis=0)
179         img_array = preprocess_input(img_array)
180
181         pred = model.predict(img_array)[0][0]
182
183         if pred > 0.5:
184             return "Rotten", pred
185         else:
186             return "Fresh", 1 - pred
187
```

```python
    if (uploaded_file or "camera_img" in st.session_state) and st.button("Run Quality Analysis"):
        input_img = img if uploaded_file else st.session_state["camera_img"]

        label, confidence = predict_image(input_img)

        if confidence < 0.65:
            st.warning("⚠️ Low confidence. Retake image with better lighting & plain background.")

        if label == "Fresh":
            st.markdown(
                f"<div class='result-fresh'>✅ FRESH<br>Confidence: {confidence:.2%}</div>",
                unsafe_allow_html=True
            )
        else:
            st.markdown(
                f"<div class='result-rotten'>❌ ROTTEN<br>Confidence: {confidence:.2%}</div>",
                unsafe_allow_html=True
            )

        st.progress(float(confidence))

    st.markdown('</div>', unsafe_allow_html=True)
```

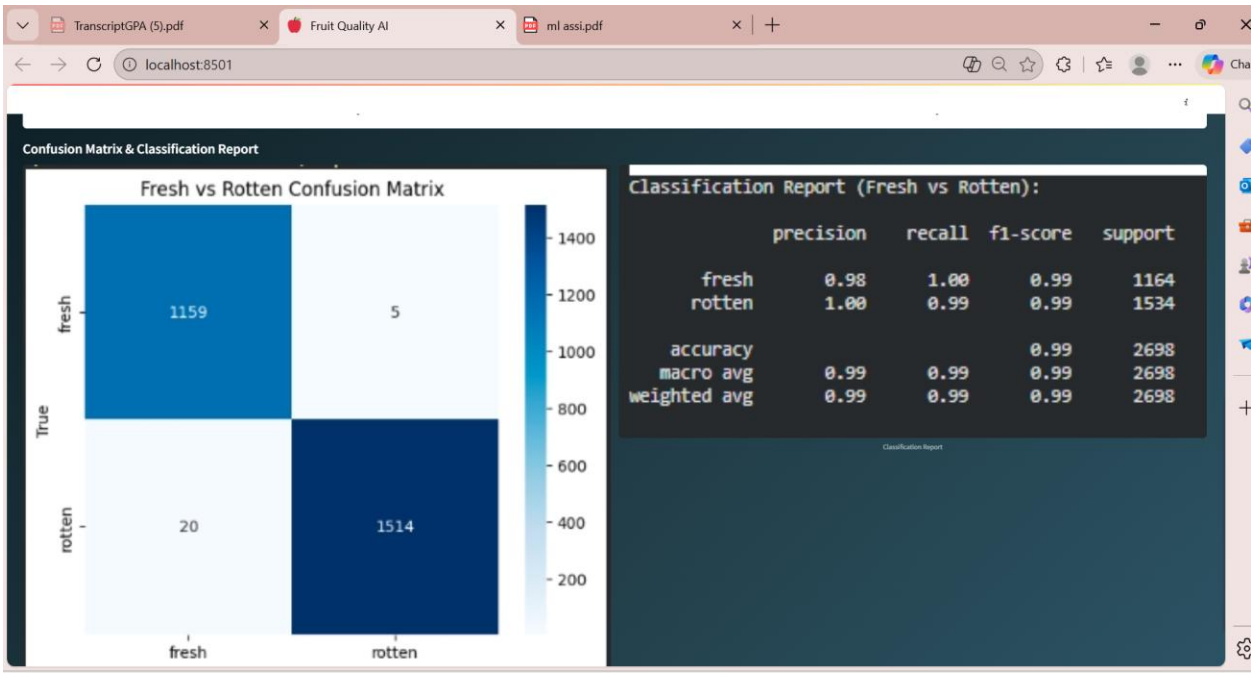
```python
# PERFORMANCE
st.write("")
st.markdown('<div class="card">', unsafe_allow_html=True)
st.subheader("📊 Training Performance")


if st.checkbox("Show Accuracy & Loss"):
    h1 = history["h1"]
    h2 = history["h2"]

    acc = h1['accuracy'] + h2['accuracy']
    val_acc = h1['val_accuracy'] + h2['val_accuracy']
    loss = h1['loss'] + h2['loss']
    val_loss = h1['val_loss'] + h2['val_loss']

    epochs_phase1 = len(h1['accuracy'])
    epochs_phase2 = len(h2['accuracy'])
    total_epochs = epochs_phase1 + epochs_phase2
    epochs = range(1, total_epochs + 1)

    fig, ax = plt.subplots(1, 2, figsize=(10, 3))

    ax[0].plot(epochs, acc, label="Train Acc", marker='o')
    ax[0].plot(epochs, val_acc, label="Val Acc", marker='o')
    ax[0].axvline(x=epochs_phase1, color='r', linestyle='--', label='Fine-tuning Start')
    ax[0].set_title("Accuracy")
    ax[0].set_xlabel("Epoch")
    ax[0].set_ylabel("Accuracy")
    ax[0].legend()
    ax[0].grid(True)

    ax[1].plot(epochs, loss, label="Train Loss", marker='o')
    ax[1].plot(epochs, val_loss, label="Val Loss", marker='o')
    ax[1].axvline(x=epochs_phase1, color='r', linestyle='--', label='Fine-tuning Start')
```

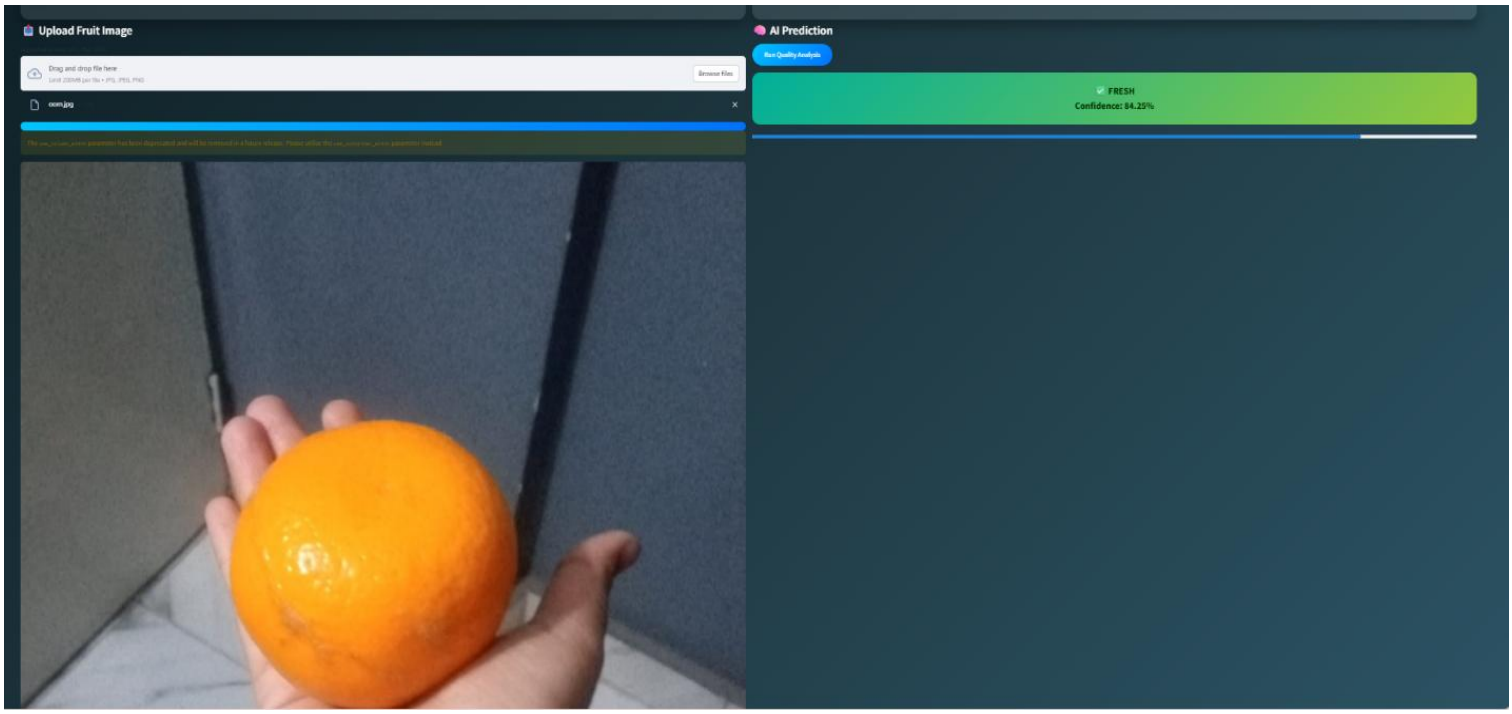
```
    ax[1].set_title("Loss")
    ax[1].set_xlabel("Epoch")
    ax[1].set_ylabel("Loss")
    ax[1].legend()
    ax[1].grid(True)

    st.pyplot(fig)

    st.subheader("Confusion Matrix & Classification Report")

    col1, col2 = st.columns(2)

    with col1:
        cm_img = Image.open("cm.png")
        st.image(cm_img, caption="Confusion Matrix", use_container_width=True)

    with col2:
        cr_img = Image.open("cr.png")
        st.image(cr_img, caption="Classification Report", use_container_width=True)

st.markdown('</div>', unsafe_allow_html=True)

# FOOTER
st.markdown("""
<p style="text-align:center; opacity:0.7;">
© 2025 • AI Fruit Quality System
</p>
""", unsafe_allow_html=True)
```

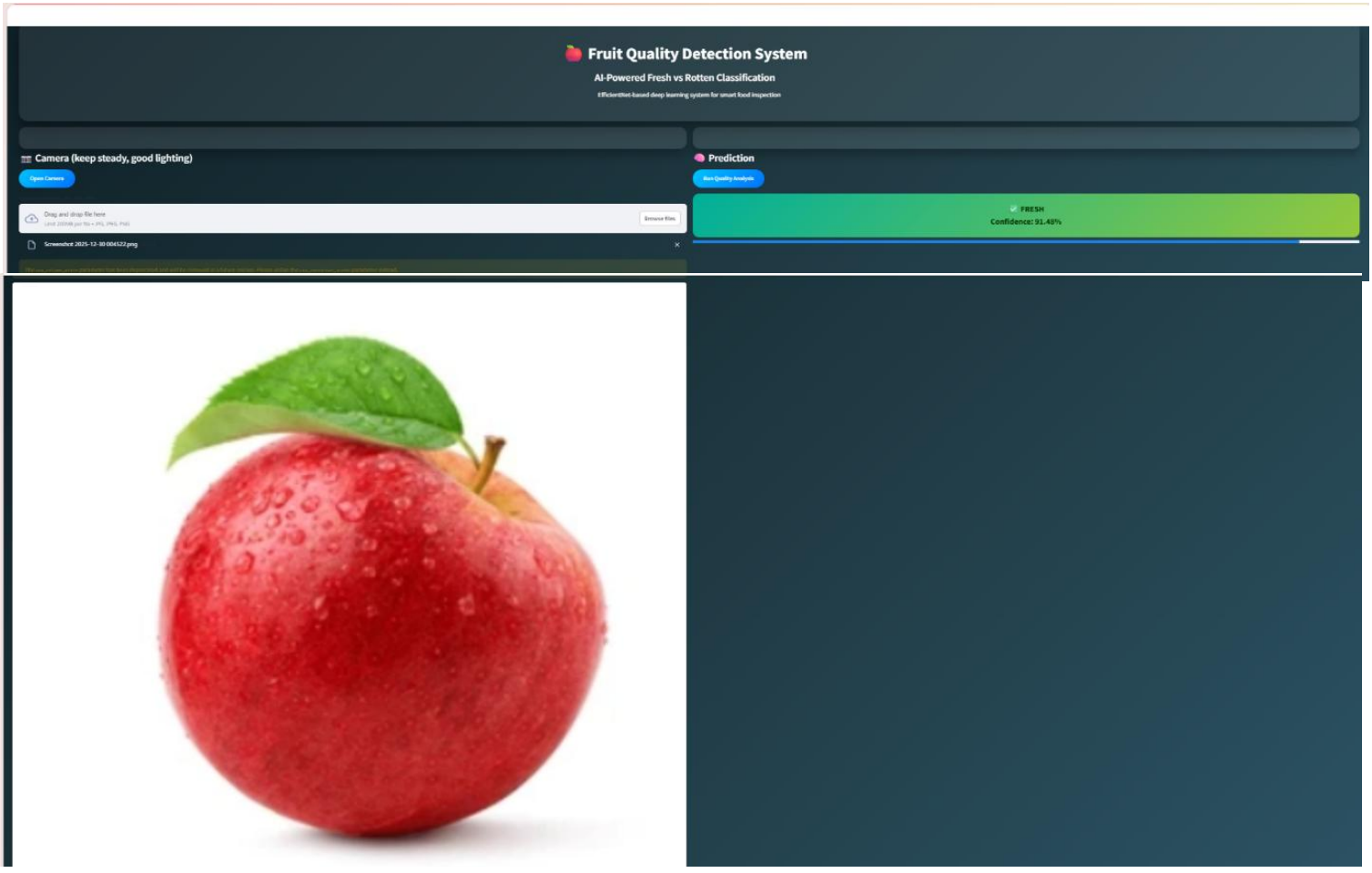
UI DISPLAY AND REAL-TIME PREDICTIONS:



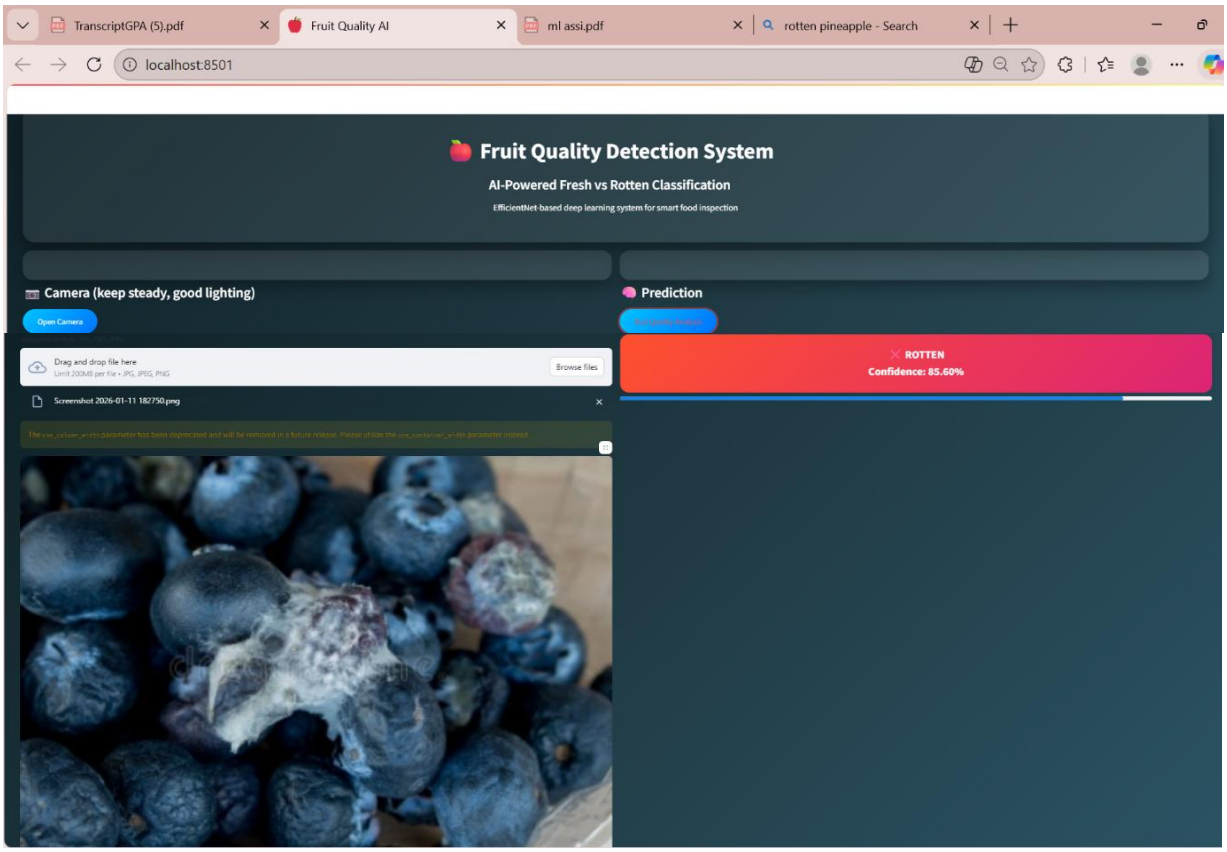

TranscriptGPA (5).pdf
Fruit Quality AI
ml assi.pdf
localhost:8501
Chat
Training Performance
Accuracy
Accuracy
Epoch
Train Acc
Val Acc
Fine-tuning Start
Loss
Loss
Epoch
Train Loss
Val Loss
Fine-tuning Start


TranscriptGPA (5).pdf
Fruit Quality AI
ml assi.pdf
localhost:8501
Confusion Matrix & Classification Report
Fresh vs Rotten Confusion Matrix
True
fresh
rotten
1159
5
20
1514
fresh
rotten
Classification Report (Fresh vs Rotten):
precision recall f1-score support
fresh 0.98 1.00 0.99 1164
rotten 1.00 0.99 0.99 1534
accuracy 0.99 2698
macro avg 0.99 0.99 0.99 2698
weighted avg 0.99 0.99 0.99 2698

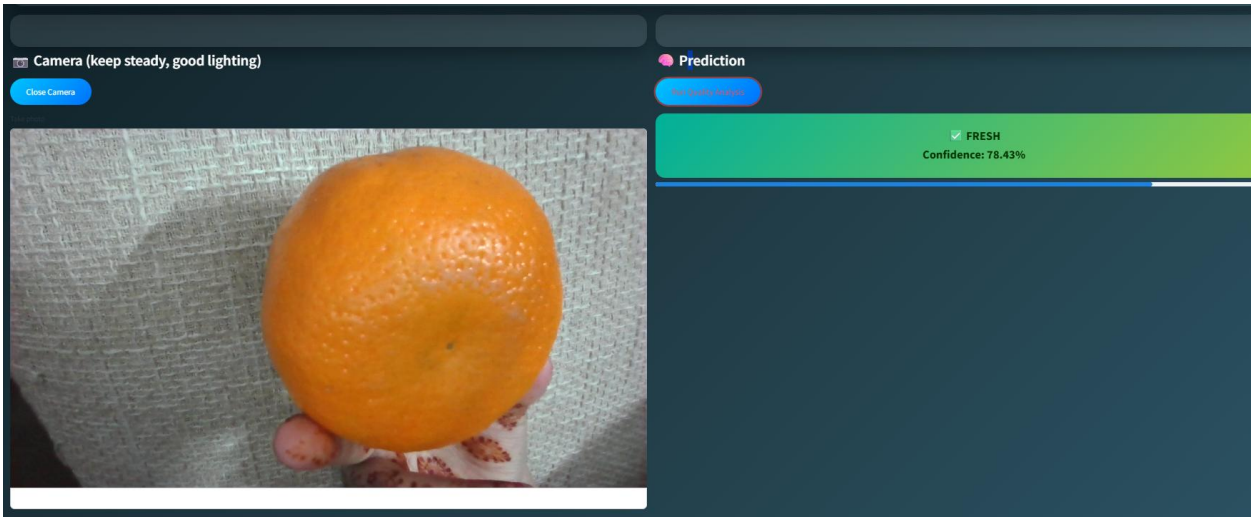




- **ABOVE ARE THE PREDICTIONS DONE FOR FRESH FRUITS IN WHICH APPLE IS UNSEEN IMAGE AND ORANGE IS REAL TIME IMAGE TAKEN AT HOME**



- **ABOVE IS THE PREDICTION ON UNSEEN DATA, ROTTEN BLUE BERRIES AND ALSO ITS NOT APART OF THE FRUIT CATEGORIES OR TRAINING DATASET**
- **STILL IT CLASSIFIED IT CORRECTLY THAT IT IS 85.60% ROTTEN.**



ABOVE IS REAL TIME PREDICTION DONE THROUGH CAMERA FOR FRESH ORANGE, JUST DUE TO NOT A WHITE BACKGROUND THE SCORE IS 78.43%, IF IT WAS WHITE IT WOULD BE MORE HIGHER