TECHNOLOGICAL INSTITUTE OF THE PHILIPPINES
938 Aurora Boulevard, Cubao, Quezon City

COLLEGE OF ENGINEERING AND ARCHITECTURE
Computer Engineering Department

Banana Ripeness Detector Using Deep Learning and Streamlit

Submitted by:

Dean Lenard D. Perez

Instructor:

Engr. Noel Jason Lusung

November 2025

Objective(s):

"This project aims to apply all the learnings for the Final Period. The objective of this project is to train a deep learning model to detect the ripeness of bananas and deploy is as a web application using Streamlit."

Intended Learning Outcomes (ILOs)

Demonstrates how to train and save a deep learning model
Demonstrates how to deploy the model in the cloud

Dataset Description

The dataset is composed of three folders (train, valid, test) and each folder contains images for Ripe, Unripe, Overripe and Rotten Bananas. A total of 1200 images for the train folder, 400 images for the valid folder and 200 images for the test folder.

Total Images: 1600

```
PS C:\Users\Dean\OneDrive\Desktop\BananaRipeness> tree
Folder PATH listing for volume Acer
Volume serial number is 0ADD-BF19
C:.
├───test
│   ├───overripe
│   ├───ripe
│   ├───rotten
│   └───unripe
├───train
│   ├───overripe
│   ├───ripe
│   ├───rotten
│   └───unripe
└───valid
    ├───overripe
    ├───ripe
    ├───rotten
    └───unripe
```

Categories:

Ripe

Unripe

Overripe

Rotten

Model Training

1. **Model Architecture**

   Model Type: Convolutional Neural Network (CNN)

   Layers:

Conv2D + Pooling	- Extracts image features
Flatten	- Flattens the result
Dense	- Learns decision boundaries
Dropout	- Prevents overfitting
SoftMax	- Outputs probabilities for each class

   Input size: (128,128,3)

   Output: 4 Classes (Ripe, Unripe, Overripe, Rotten)

2. **Code Snippets** (Training Model)
    a. Build the CNN model

```
# Build CNN model using Rectified Linear Unit
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(128,128,3)),
    MaxPooling2D(2,2),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D(2,2),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(NUM_CLASSES, activation='softmax')
])
```

    b. Compile and Train

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train model
history = model.fit( train_generator, validation_data=valid_generator, epochs=20)
```

    c. Data Preparation / Processing

```
IMAGE_SIZE = (128, 128)
BATCH_SIZE = 32
NUM_CLASSES = 4

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    zoom_range=0.2,
    horizontal_flip=True)

valid_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    r"C:\Users\Dean\OneDrive\Desktop\BananaRipenessApp\pythonProject\BananaRipeness\train",
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical')

valid_generator = valid_datagen.flow_from_directory(
    r"C:\Users\Dean\OneDrive\Desktop\BananaRipenessApp\pythonProject\BananaRipeness\valid",
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical')
```

d. Accuracy Vs Loss

```python
import pickle
import matplotlib.pyplot as plt

# Load the saved training history
with open('training_history.pkl', 'rb') as f:
    history = pickle.load(f)

# Plot Accuracy
plt.plot( *args: history['accuracy'], label='Training Accuracy')
plt.plot( *args: history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot Loss
plt.plot( *args: history['loss'], label='Training Loss')
plt.plot( *args: history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

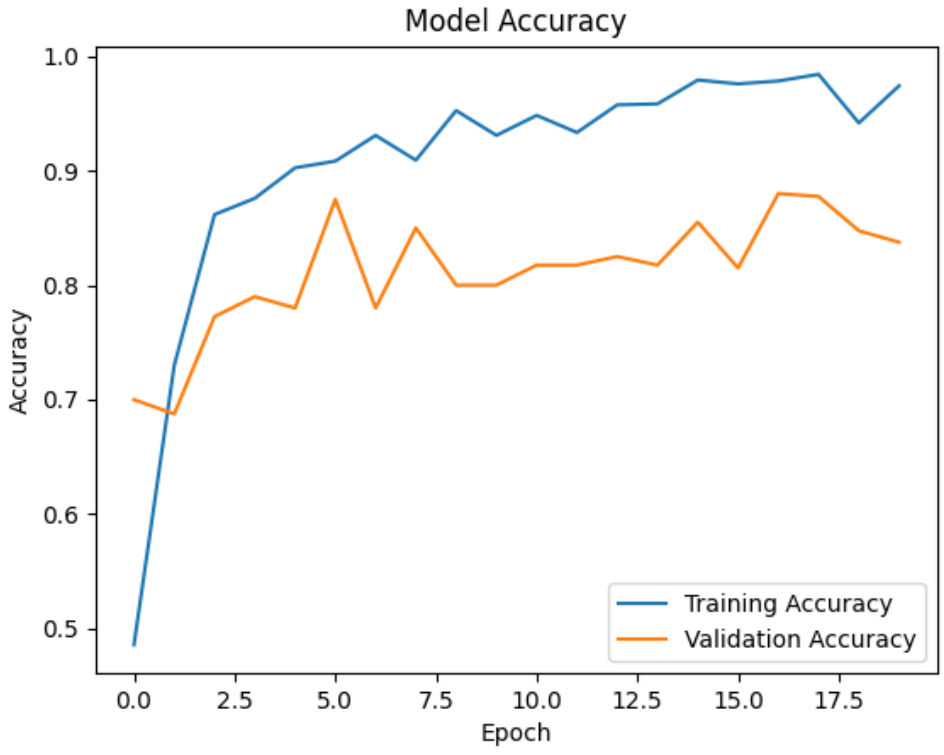
3. Training Results / Metrics

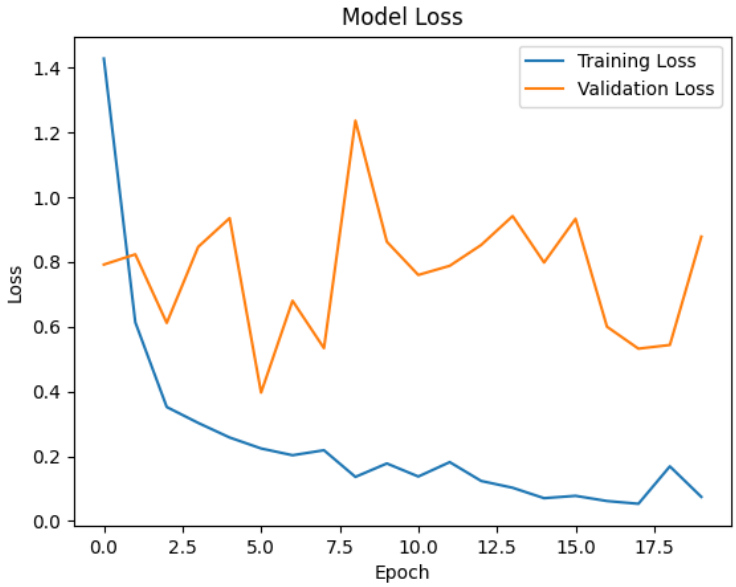
a. MODEL TRAINING

```
Epoch 1/20
38/38 ━━━━━━━━━━━━━━━━━━━━ 35s 906ms/step - accuracy: 0.4858 - loss: 1.4280 - val_accuracy: 0.7000 - val_loss: 0.7921
Epoch 2/20
38/38 ━━━━━━━━━━━━━━━━━━━━ 12s 319ms/step - accuracy: 0.7300 - loss: 0.6138 - val_accuracy: 0.6875 - val_loss: 0.8235
Epoch 3/20
38/38 ━━━━━━━━━━━━━━━━━━━━ 13s 348ms/step - accuracy: 0.8617 - loss: 0.3523 - val_accuracy: 0.7725 - val_loss: 0.6118
Epoch 4/20
38/38 ━━━━━━━━━━━━━━━━━━━━ 12s 308ms/step - accuracy: 0.8758 - loss: 0.3032 - val_accuracy: 0.7900 - val_loss: 0.8466
Epoch 5/20
38/38 ━━━━━━━━━━━━━━━━━━━━ 12s 304ms/step - accuracy: 0.9025 - loss: 0.2581 - val_accuracy: 0.7800 - val_loss: 0.9354
Epoch 6/20
38/38 ━━━━━━━━━━━━━━━━━━━━ 11s 301ms/step - accuracy: 0.9083 - loss: 0.2243 - val_accuracy: 0.8750 - val_loss: 0.3967
Epoch 7/20
38/38 ━━━━━━━━━━━━━━━━━━━━ 11s 291ms/step - accuracy: 0.9308 - loss: 0.2034 - val_accuracy: 0.7800 - val_loss: 0.6803
Epoch 8/20
38/38 ━━━━━━━━━━━━━━━━━━━━ 11s 301ms/step - accuracy: 0.9092 - loss: 0.2188 - val_accuracy: 0.8500 - val_loss: 0.5339
Epoch 9/20
38/38 ━━━━━━━━━━━━━━━━━━━━ 11s 293ms/step - accuracy: 0.9525 - loss: 0.1365 - val_accuracy: 0.8000 - val_loss: 1.2364
Epoch 10/20
38/38 ━━━━━━━━━━━━━━━━━━━━ 11s 292ms/step - accuracy: 0.9308 - loss: 0.1780 - val_accuracy: 0.8000 - val_loss: 0.8625
```

```
Epoch 11/20
38/38 ━━━━━━━━━━━━━━━━━━━━ 11s 300ms/step - accuracy: 0.9483 - loss: 0.1377 - val_accuracy: 0.8175 - val_loss: 0.7599
Epoch 12/20
38/38 ━━━━━━━━━━━━━━━━━━━━ 11s 290ms/step - accuracy: 0.9333 - loss: 0.1821 - val_accuracy: 0.8175 - val_loss: 0.7882
Epoch 13/20
38/38 ━━━━━━━━━━━━━━━━━━━━ 12s 303ms/step - accuracy: 0.9575 - loss: 0.1239 - val_accuracy: 0.8250 - val_loss: 0.8526
Epoch 14/20
38/38 ━━━━━━━━━━━━━━━━━━━━ 13s 350ms/step - accuracy: 0.9583 - loss: 0.1030 - val_accuracy: 0.8175 - val_loss: 0.9418
Epoch 15/20
38/38 ━━━━━━━━━━━━━━━━━━━━ 13s 339ms/step - accuracy: 0.9792 - loss: 0.0708 - val_accuracy: 0.8550 - val_loss: 0.7983
Epoch 16/20
38/38 ━━━━━━━━━━━━━━━━━━━━ 12s 316ms/step - accuracy: 0.9758 - loss: 0.0780 - val_accuracy: 0.8150 - val_loss: 0.9333
Epoch 17/20
38/38 ━━━━━━━━━━━━━━━━━━━━ 12s 302ms/step - accuracy: 0.9783 - loss: 0.0620 - val_accuracy: 0.8800 - val_loss: 0.6002
Epoch 18/20
38/38 ━━━━━━━━━━━━━━━━━━━━ 12s 312ms/step - accuracy: 0.9842 - loss: 0.0535 - val_accuracy: 0.8775 - val_loss: 0.5324
Epoch 19/20
38/38 ━━━━━━━━━━━━━━━━━━━━ 11s 299ms/step - accuracy: 0.9417 - loss: 0.1690 - val_accuracy: 0.8475 - val_loss: 0.5438
Epoch 20/20
38/38 ━━━━━━━━━━━━━━━━━━━━ 11s 298ms/step - accuracy: 0.9742 - loss: 0.0749 - val_accuracy: 0.8375 - val_loss: 0.8782
```

b. Model Accuracy and Loss Plot



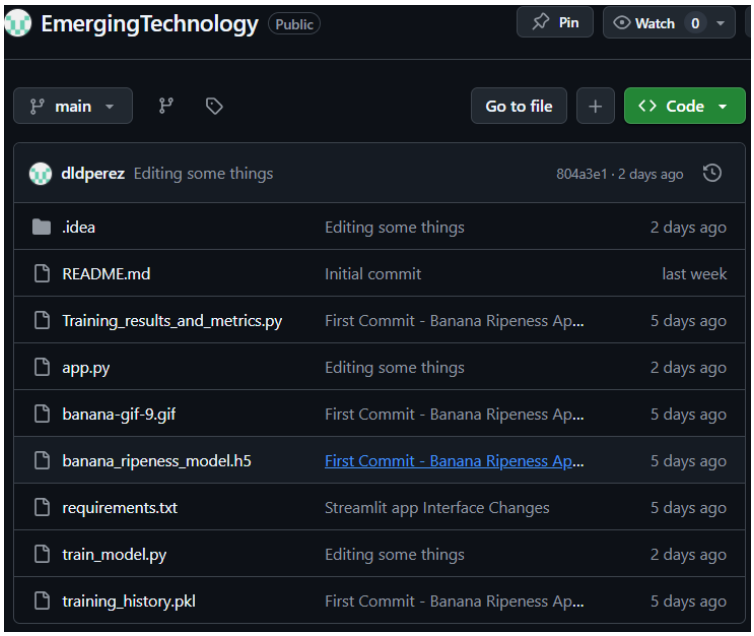


4. Saving the Model

```
56    # Save trained model
57    model.save('banana_ripeness_model.h5')
58    print("Model saved as banana_ripeness_model.h5")
```

```
Model saved as banana_ripeness_model.h5
```

5. Streamlit Deployment
    a. Folder Structure/ GITHUB



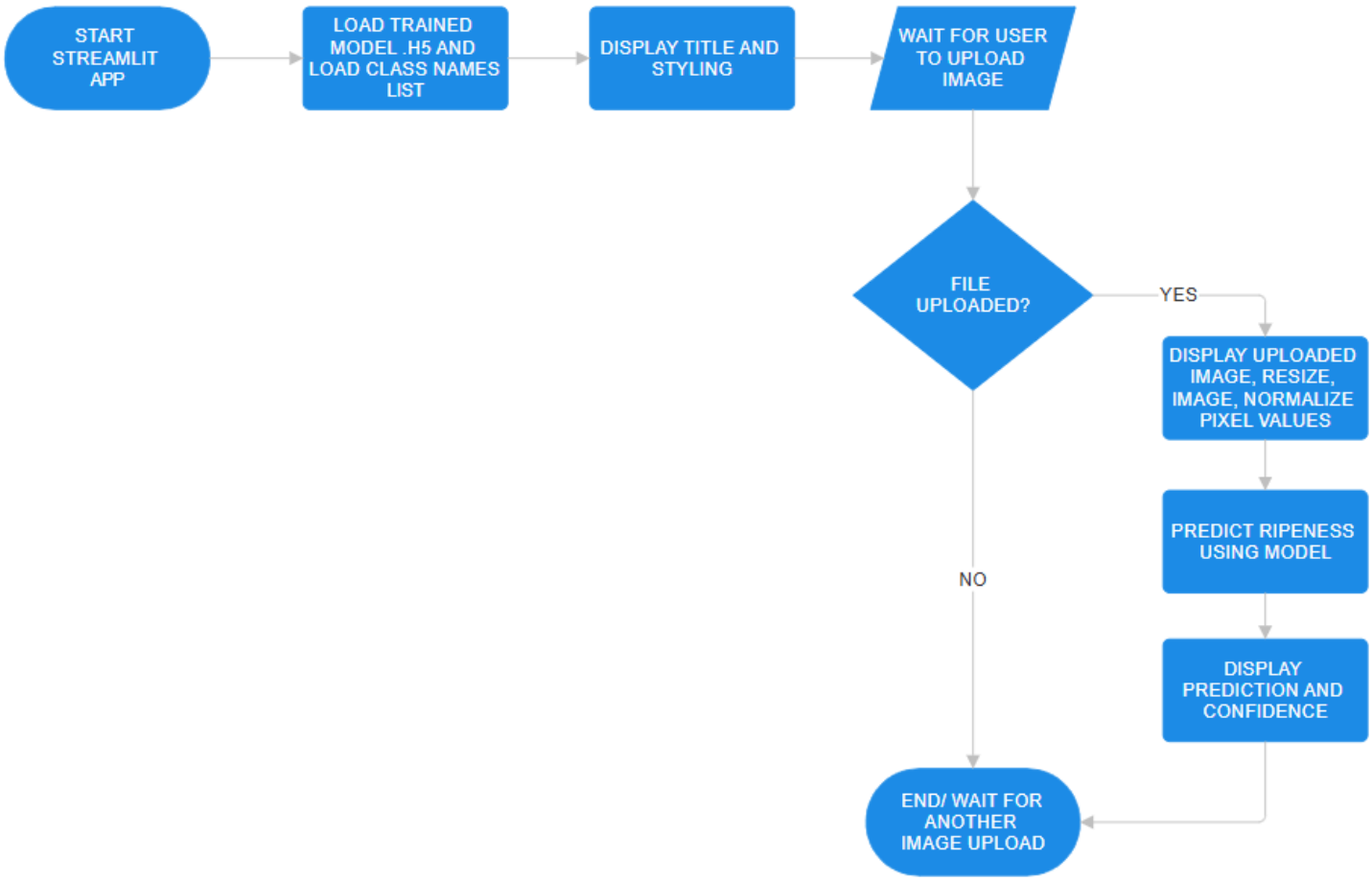
b. App Features
   - A user can upload a banana image
   - The system will predict the ripeness
   - Shows the confidence of the system

c. Codes Snippets (Banana Ripeness Application)

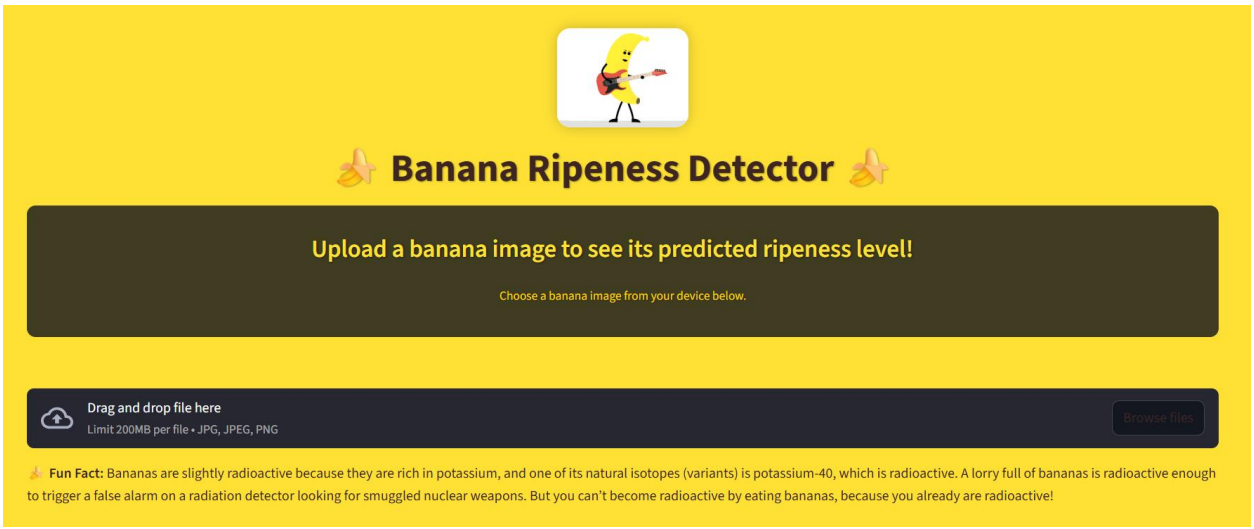
```python
import streamlit as st
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import numpy as np
from PIL import Image

# Load model
model = load_model('banana_ripeness_model.h5')
class_names = ['Overripe', 'Ripe', 'Rotten', 'Unripe']

#Page Title
st.set_page_config(page_title="Banana Ripeness Detector", layout="wide")

# CSS styling
st.markdown("""
    <style>
    .stApp {
        background-color: #FFE135 !important;
        color: #3e2723;
    }
    h1, h2, h3, p {
        color: #3e2723;
        text-shadow: 1px 1px 2px rgba(0,0,0,0.3);
    }
    h1 {
        text-align: center;
        color: #fdd835;
    }
    .instructions {
        text-align: center;
        background-color: rgba(30,30,30,0.85);
        padding: 20px;
        border-radius: 10px;
        margin-bottom: 30px;
    }
    .instructions h3 {
        color: #FFE135;
        font-size: 28px;
        text-shadow: 1px 1px 3px rgba(0,0,0,0.5);
        margin-bottom: 10px;
```

```
    }
    .instructions p {
        color: #FFD700;
        font-size: 14px;
        margin-top: 0;
    }
    .title-gif {
        display: block;
        margin-left: auto;
        margin-right: auto;
        width: 150px;
        border-radius: 15px;
        box-shadow: 0px 0px 15px rgba(0,0,0,0.2);
    }
    </style>
""", unsafe_allow_html=True)
# GIFs Banana Playing Guitar
st.markdown(
    '<img class="title-gif" src="https://raw.githubusercontent.com/dldperez/EmergingTechnology/main/banana-gif-9.gif">',
    unsafe_allow_html=True
)

# Title
st.title("🍌 Banana Ripeness Detector 🍌")

# Instructions
st.markdown("""
    <div class="instructions">
        <h3>Upload a banana image to see its predicted ripeness level!</h3>
        <p>Choose a banana image from your device below.</p>
    </div>
""", unsafe_allow_html=True)
# File uploader
uploaded_file = st.file_uploader("", type=["jpg", "jpeg", "png"])

if uploaded_file is not None:
    img = Image.open(uploaded_file)
    st.image(img, caption="Uploaded Image", use_container_width=True)

    # Preprocess image
    img = img.resize((128, 128))
    img_array = image.img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)

    # Predict
    predictions = model.predict(img_array)
    class_idx = np.argmax(predictions)
    confidence = predictions[0][class_idx]
    prediction = class_names[class_idx]

    # Prediction results and confidence
    st.subheader(f"Prediction: **{prediction}** 🍌")
    st.write(f"Confidence: {confidence*100:.2f}%")
#Banana Fun Fact
st.markdown("""
    <div class="funfact">
        <b>🍌 Fun Fact:</b>
        Bananas are slightly radioactive because they are rich in potassium,
        and one of its natural isotopes (variants) is potassium-40, which is radioactive.
        A lorry full of bananas is radioactive enough to trigger a false alarm on a radiation detector
        looking for smuggled nuclear weapons. But you can't become radioactive by eating bananas,
        because you already are radioactive!
    </div>
""", unsafe_allow_html=True)
```

d. App Flowchart



e. App Interface Screenshot



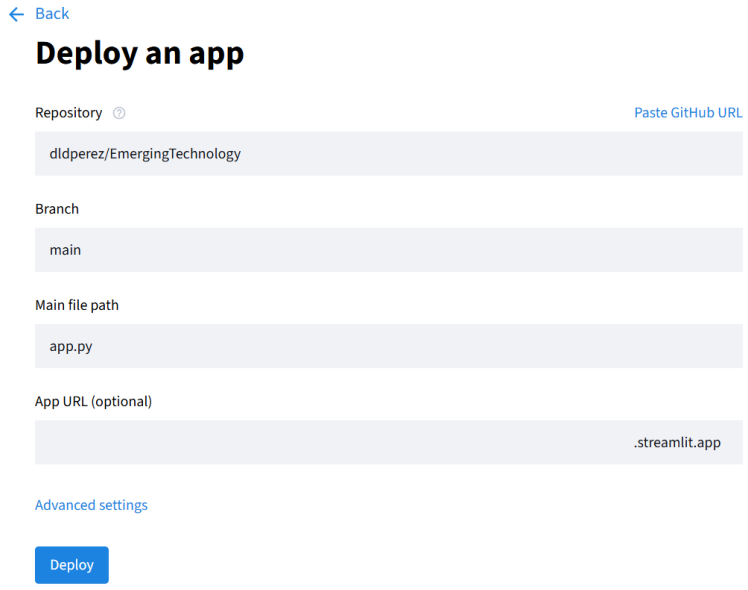
6. Deployment URL
   - GitHub File Upload

```
C:\Users\Dean\OneDrive\Desktop\BananaRipenessApp\pythonProject>git commit -m "First Commit - Banana Ripeness Application"
[master (root-commit) ab16ece] First Commit - Banana Ripeness Application
 6 files changed, 142 insertions(+)
 create mode 100644 Training_results_and_metrics.py
 create mode 100644 app.py
 create mode 100644 banana-gif-9.gif
 create mode 100644 banana_ripeness_model.h5
 create mode 100644 train_model.py
 create mode 100644 training_history.pkl
```

   - GITHUB Connection

```
PS C:\WINDOWS\system32> Set-Service -Name ssh-agent -StartupType Manual
PS C:\WINDOWS\system32> Start-Service ssh-agent
PS C:\WINDOWS\system32> Get-Service ssh-agent

Status   Name               DisplayName
------   ----               -----------
Running  ssh-agent          OpenSSH Authentication Agent


PS C:\WINDOWS\system32> ssh-keygen -t ed25519 -C "qdldperez@tip.edu.ph"
Generating public/private ed25519 key pair.
Enter file in which to save the key (C:\Users\Dean/.ssh/id_ed25519):
C:\Users\Dean/.ssh/id_ed25519 already exists.
Overwrite (y/n)?
PS C:\WINDOWS\system32> ssh-add C:\Users\Dean\.ssh\id_ed25519
>>
Identity added: C:\Users\Dean\.ssh\id_ed25519 (qdldperez@tip.edu.ph)
PS C:\WINDOWS\system32> type C:\Users\Dean\.ssh\id_ed25519.pub
>>
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIDZHC9EsrKJsNNsSFtKFV8FSWcWwH9TpBg8cYM9Mzj24 qdldperez@tip.edu.ph
PS C:\WINDOWS\system32> git remote set-url origin git@github.com:dldperez/EmergingTechnology.git
fatal: not a git repository (or any of the parent directories): .git
PS C:\WINDOWS\system32> cd "C:\Users\Dean\OneDrive\Desktop\BananaRipenessApp\pythonProject"
>>
PS C:\Users\Dean\OneDrive\Desktop\BananaRipenessApp\pythonProject> git remote set-url origin git@github.com:dldperez/EmergingTechnology.git
PS C:\Users\Dean\OneDrive\Desktop\BananaRipenessApp\pythonProject> ssh -T git@github.com
Hi dldperez! You've successfully authenticated, but GitHub does not provide shell access.
PS C:\Users\Dean\OneDrive\Desktop\BananaRipenessApp\pythonProject>
```

   - Deploying the app on Streamlit



   - Streamlit App Link: https://dldperez-emergingtechnology-app-oceynf.streamlit.app/

7. **Reflection / Learnings**

During the model training, I trained it using a few epochs hoping it will be enough to make predictions but it led to few errors. I realized that using a few epochs can lead to underfitting where the model fails to learn the important patterns. Monitoring the training and loss also helps us determine if our model is functioning properly.

8. **Conclusion**

This project helped me apply everything that I learned in this subject. I was able to train and save a deep learning model that detects the ripeness of bananas and deploy it as a web application using Streamlit. In this project, I learned how to connect the model training for the deep learning by uploading all the project files to GitHub Repository, this includes the model, background, Python Script and requirements. Then we connected the GitHub repository to Streamlit Cloud, setting the main file path and deploying the app successfully. Seeing that the application is deployed and even got it to working successfully is very rewarding.

9. **References**

- GitHub link:
  https://github.com/dldperez/EmergingTechnology.git
- Dataset Link:
  https://universe.roboflow.com/roboflow-universe-projects/banana-ripeness-classification
- Banana Fun Fact:
  Villazon, L. (2024, July 1). *How many bananas would I need to eat to become radioactive? BBC Science Focus magazine.* *https://www.sciencefocus.com/science/how-many-bananas-would-i-need-to-eat-to-become-radioactive*