Image-based Lost and Found

```
import streamlit as st
import os
import numpy as np
import json
import uuid
from PIL import Image
from tensorflow.keras.applications.mobilenet_v2 import MobileNetV2, preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from sklearn.metrics.pairwise import cosine_similarity

# Load model
model = MobileNetV2(weights="imagenet", include_top=False, pooling='avg')

# Extract image features
def extract_features(image):
    image = image.resize((224, 224))
    image = img_to_array(image)
    image = preprocess_input(image)
    image = np.expand_dims(image, axis=0)
    features = model.predict(image)
    return features.flatten()

# Save image + metadata
def save_image(img, folder, filename, meta):
    os.makedirs(folder, exist_ok=True)
    path = os.path.join(folder, filename)
    img.save(path)

    meta["image"] = path
    with open("items.json", "a") as f:
        f.write(json.dumps(meta) + "\n")

    return path

# Load embeddings
def load_embeddings(folder):
    embeddings = []
    files = []
    for fname in os.listdir(folder):
        path = os.path.join(folder, fname)
        if path.endswith((".jpg", ".png", ".jpeg")):
            try:
                img = Image.open(path)
                feat = extract_features(img)
```

```
            embeddings.append(feat)
            files.append(path)
        except:
            continue
    return np.array(embeddings), files


# Streamlit UI
st.set_page_config(page_title="FindItBack", layout="centered")
st.title("🎒 FindItBack – Lost & Found App (Campus Edition)")

# Search Section
st.subheader("🔎 Search Posted Items")
search_term = st.text_input("Search by name, place, or item ID")
if search_term:
    if os.path.exists("items.json"):
        st.info(f"Results for '{search_term}':")
        with open("items.json") as f:
            data = [json.loads(line) for line in f if search_term.lower() in line.lower()]
            if data:
                for item in data:
                    if os.path.exists(item["image"]):
                        st.image(item["image"], width=200)
                    else:
                        st.warning(f"Image not found: {item['image']}")
                    st.markdown(f"""
                    **Item Name:** {item.get("name", "N/A")}
                    **Place:** {item.get("place", "N/A")}
                    **Date:** {item.get("date", "N/A")}
                    **Item ID:** `{item.get("id", "N/A")}`
                    **Uploader:** {item.get("uploader_name", "N/A")}
                    **Contact:** `{item.get("uploader_contact", "N/A")}`
                    """)
            else:
                st.warning("No matching items found.")
    else:
        st.warning("No items posted yet.")

st.markdown("---")

# Upload Section
option = st.radio("Upload a:", ["Lost Item", "Found Item"])
uploaded_file = st.file_uploader("Upload Image", type=["jpg", "jpeg", "png"])

if uploaded_file:
    try:
        img = Image.open(uploaded_file)
        st.image(img, caption="Uploaded Image", use_column_width=True)
    except:
        st.error("Invalid image file!")
        st.stop()
```

```
name = st.text_input("Item Name")
place = st.text_input("Place where item was lost/found")
date = st.date_input("Date Found/Lost")
uploader_name = st.text_input("Your Name")
uploader_contact = st.text_input("Your Contact Number")

if st.button("Submit"):
    folder = "lost" if option == "Lost Item" else "found"
    item_id = str(uuid.uuid4())[:8]
    filename = f"{item_id}.jpg"

    meta = {
        "id": item_id,
        "type": option,
        "name": name,
        "place": place,
        "date": str(date),
        "uploader_name": uploader_name,
        "uploader_contact": uploader_contact
    }

    save_path = save_image(img, folder, filename, meta)

    st.success(f"{option} uploaded successfully!")

    # Matching
    other_folder = "found" if option == "Lost Item" else "lost"
    if len(os.listdir(other_folder)) == 0:
        st.info(f"No items in '{other_folder}' folder to match with.")
    else:
        st.subheader("🔍 Matching Results")
        query_feat = extract_features(img)
        db_feats, db_files = load_embeddings(other_folder)

        if len(db_feats) > 0:
            sims = cosine_similarity([query_feat], db_feats)[0]
            top_3 = sims.argsort()[-3:][::-1]
            for idx in top_3:
                matched_path = db_files[idx]
                matched_score = sims[idx]
                st.image(matched_path, caption=f"Match Score: {matched_score:.2f}", width=300)

                # Show match metadata
                if os.path.exists("items.json"):
                    with open("items.json") as f:
                        for line in f:
                            item = json.loads(line)
                            if item.get("image") == matched_path:
                                st.markdown(f"""
                                **Item Name:** {item.get("name", "N/A")}
```

```
                        **Place:** {item.get("place", "N/A")}
                        **Date:** {item.get("date", "N/A")}
                        **Uploaded By:** {item.get("uploader_name", "N/A")}
                        **Contact:** `{item.get("uploader_contact", "N/A")}`
                        """)
                        break
            else:
                st.warning("No valid images to match with.")

st.markdown("---")

# Lost Items Gallery
st.subheader("🗂️ Lost Items Gallery")
lost_files = os.listdir("lost") if os.path.exists("lost") else []
for f in lost_files:
    if f.endswith((".jpg", ".jpeg", ".png")):
        st.image(f"lost/{f}", width=150)

# Found Items Gallery
st.subheader("🗂️ Found Items Gallery")
found_files = os.listdir("found") if os.path.exists("found") else []
for f in found_files:
    if f.endswith((".jpg", ".jpeg", ".png")):
        st.image(f"found/{f}", width=150)


explain this code blockwise why we have written
```

This Streamlit application allows users to upload images of lost or found items, search for previously posted items, and match similar items based on image content. Here's a breakdown of the code, explained block by block:

1. Imports

```
python

`import streamlit as st
import os
import numpy as np
import json
import uuid
from PIL import Image
from tensorflow.keras.applications.mobilenet_v2 import MobileNetV2, preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from sklearn.metrics.pairwise import cosine_similarity
`
```

- **streamlit**: This is used to build the web application.
- **os**: To interact with the file system (e.g., saving/loading files).
- **numpy**: For numerical operations, especially array manipulation.
- **json**: To read and write JSON files for item metadata.

- **uuid**: For generating unique identifiers for each item.
- **PIL (Pillow)**: For image processing tasks (opening and resizing images).
- **MobileNetV2**: A pre-trained deep learning model for feature extraction from images.
- **cosine_similarity**: To calculate similarity between image features, allowing us to match similar images.

2. Loading the Pre-trained Model

```
python

`model = MobileNetV2(weights="imagenet", include_top=False, pooling='avg')
`
```

- **MobileNetV2** is loaded here with pre-trained weights from ImageNet. The **`include_top=False`** argument excludes the final classification layer, so the model outputs feature vectors, which are useful for similarity matching.
- **Pooling='avg'**: Uses global average pooling to reduce the feature map dimensions to a single vector.

3. Extract Features from Image

```
python

`def extract_features(image):
    image = image.resize((224, 224))
    image = img_to_array(image)
    image = preprocess_input(image)
    image = np.expand_dims(image, axis=0)
    features = model.predict(image)
    return features.flatten()
`
```

- This function processes the image:
    1. **Resize**: Resizes the image to 224x224, the input size expected by MobileNetV2.
    2. **img_to_array**: Converts the image to a NumPy array.
    3. **preprocess_input**: Preprocesses the image (scaling pixel values to match the model's training data).
    4. **np.expand_dims**: Adds an extra dimension to simulate batch input (required by Keras models).
    5. **model.predict**: Extracts the features (a vector) from the image.
    6. **flatten**: Converts the feature vector to a 1D array.

4. Save Image and Metadata

```
python

`def save_image(img, folder, filename, meta):
    os.makedirs(folder, exist_ok=True)
    path = os.path.join(folder, filename)
```

```
    img.save(path)

    meta["image"] = path
    with open("items.json", "a") as f:
        f.write(json.dumps(meta) + "\n")

    return path
`
```

- **save_image**: This function saves the uploaded image and its metadata.
    1. It ensures the target folder exists (`os.makedirs`).
    2. Saves the image to the folder with the generated filename.
    3. Adds the image path to the metadata and appends it to the `items.json` file as a new entry.
    4. Returns the image path.

5. Load Embeddings for All Images in Folder

```
python

`def load_embeddings(folder):
    embeddings = []
    files = []
    for fname in os.listdir(folder):
        path = os.path.join(folder, fname)
        if path.endswith((".jpg", ".png", ".jpeg")):
            try:
                img = Image.open(path)
                feat = extract_features(img)
                embeddings.append(feat)
                files.append(path)
            except:
                continue
    return np.array(embeddings), files
`
```

- This function loads all images in a specified folder, extracts their features using `extract_features()`, and returns the feature vectors (embeddings) and the paths of the images.

6. Streamlit UI Setup

```
python

`st.set_page_config(page_title="FindItBack", layout="centered")
st.title("🎒 FindItBack – Lost & Found App (Campus Edition)")
`
```

- **st.set_page_config**: Configures the Streamlit page with a title and layout.
- **st.title**: Sets the title of the app.

7. Search Section

```
python

`st.subheader("🔎 Search Posted Items")
search_term = st.text_input("Search by name, place, or item ID")
if search_term:
```

```
    if os.path.exists("items.json"):
        st.info(f"Results for '{search_term}':")
        with open("items.json") as f:
            data = [json.loads(line) for line in f if search_term.lower() in line.lower()]
            if data:
                for item in data:
                    if os.path.exists(item["image"]):
                        st.image(item["image"], width=200)
                    else:
                        st.warning(f"Image not found: {item['image']}")
                    st.markdown(f"""
                    **Item Name:** {item.get("name", "N/A")}
                    **Place:** {item.get("place", "N/A")}
                    **Date:** {item.get("date", "N/A")}
                    **Item ID:** `{item.get("id", "N/A")}`
                    **Uploader:** {item.get("uploader_name", "N/A")}
                    **Contact:** `{item.get("uploader_contact", "N/A")}`
                    """)
            else:
                st.warning("No matching items found.")
    else:
        st.warning("No items posted yet.")
`
```

- **Search Input**: A search bar allows users to search for items by name, place, or ID.
- The code checks if the search term matches any item in `items.json`. If matches are found, their metadata (including image) is displayed.

8. Upload Section

```
python

`option = st.radio("Upload a:", ["Lost Item", "Found Item"])
uploaded_file = st.file_uploader("Upload Image", type=["jpg", "jpeg", "png"])

if uploaded_file:
    try:
        img = Image.open(uploaded_file)
        st.image(img, caption="Uploaded Image", use_column_width=True)
    except:
        st.error("Invalid image file!")
        st.stop()

    name = st.text_input("Item Name")
    place = st.text_input("Place where item was lost/found")
    date = st.date_input("Date Found/Lost")
    uploader_name = st.text_input("Your Name")
    uploader_contact = st.text_input("Your Contact Number")

    if st.button("Submit"):
        folder = "lost" if option == "Lost Item" else "found"
        item_id = str(uuid.uuid4())[:8]
        filename = f"{item_id}.jpg"

        meta = {
            "id": item_id,
            "type": option,
            "name": name,
            "place": place,
            "date": str(date),
            "uploader_name": uploader_name,
            "uploader_contact": uploader_contact
        }

        save_path = save_image(img, folder, filename, meta)

        st.success(f"{option} uploaded successfully!")
`
```

- **Upload UI**: Users can choose whether they are uploading a "Lost Item" or a "Found Item".
- The uploaded image is displayed, and the user is asked to fill out metadata (e.g., item name, place, date, uploader's contact).
- On submission, the image and metadata are saved, and a unique ID is generated for the item.

9. Matching Items

```
python


`# Matching
other_folder = "found" if option == "Lost Item" else "lost"
if len(os.listdir(other_folder)) == 0:
    st.info(f"No items in '{other_folder}' folder to match with.")
else:
    st.subheader("🔍 Matching Results")
    query_feat = extract_features(img)
    db_feats, db_files = load_embeddings(other_folder)

    if len(db_feats) > 0:
        sims = cosine_similarity([query_feat], db_feats)[0]
        top_3 = sims.argsort()[-3:][::-1]
        for idx in top_3:
            matched_path = db_files[idx]
            matched_score = sims[idx]
            st.image(matched_path, caption=f"Match Score: {matched_score:.2f}", width=300)
            # Show match metadata
            if os.path.exists("items.json"):
                with open("items.json") as f:
                    for line in f:
                        item = json.loads(line)
                        if item.get("image") == matched_path:
                            st.markdown(f"""
                            **Item Name:** {item.get("name", "N/A")}
                            **Place:** {item.get("place", "N/A")}
                            **Date:** {item.get("date", "N/A")}
                            **Uploaded By:** {item.get("uploader_name", "N/A")}
                            **Contact:** `{item.get("uploader_contact", "N/A")}`
                            """)
                            break
    else:
        st.warning("No valid images to match with.")
`
```

- After an image is uploaded, it is matched with images in the opposite folder (found or lost). Features are extracted from the query image and compared with the embeddings of other items using cosine similarity.
- The top 3 most similar images are displayed with match scores.

10. Item Galleries

```
python


`st.subheader("📂 Lost Items Gallery")
lost_files = os.listdir("lost") if os.path.exists("lost") else []
for f in lost_files:
    if f.endswith((".jpg", ".jpeg", ".png")):
        st.image(f"lost/{f}", width=150)

# Found Items Gallery
st.subheader("📂 Found Items Gallery")
found_files = os.listdir("found") if os.path.exists("found") else []
for f in found_files:
    if f.endswith((".jpg", ".jpeg", ".png")):
```

```
        st.image(f"found/{f}", width=150)
`
```

- Displays galleries of lost and found items by loading images from the `lost` and `found` folders and showing them in the app.

Summary

This Streamlit app serves as a "Lost & Found" platform, allowing users to upload images of lost or found items, search for existing items, and match similar images. It utilizes a pre-trained MobileNetV2 model for image feature extraction and cosine similarity to match lost and found items.