**Author : Surya Teja Neerukattu  
  
Objective:** Developing a Landmark Recognition web app that helps users identify buildings and landmarks by uploading or capturing an image. The system will use image retrieval techniques to match the query image with images in the Oxford5k dataset and provide information about the identified landmark.  
  
**Practical use:**  
  
*User Need:* Tourists and locals often encounter buildings or landmarks they cannot identify.

Solution: Your system provides an easy way to recognize and learn about landmarks by simply taking a photo.

*Our project plan:*  
  
we have divided this project development into 5 stages:  
  
1) Data Preparation & Model Training

2) API Development for Landmark Recognition

3) Frontend Development with React

4) Backend Integration & Deployment

5) Testing, accuracy improvement measures, Deployment & Hosting.

**Development process:  
  
  
1)** Data Preparation & Model Training :

1.a) setup :  
  
 for model training and backend, programming language used : python 3.10.12

Frame work used for backend : flask or spring (need to decide)  
  
 libraries and tools installled:  
  
 pip install flask tensorflow keras scikit-learn opencv-python pillow numpy pandas faiss-cpu flask-cors sqlite3

for front end :

language : JavaScript  
 Frame work : React, bootstarp (css)

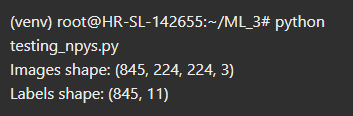
setup installation command :  
  
 npx create-react-app landmark-recognition-app

cd landmark-recognition-app

npm install axios bootstrap

**1.2) Data Preparation :**

From downloaded oxford5k\_data, we will have images and groundtruth.json which will have info about the image labels.

Now, we did preprocessing of data using preprocess\_data.py   
  
This script will generate two files, images.npy and labels.npy, which contain our preprocessed images and corresponding labels  
  
and also to test these , images.npy and labels.npy created properly not we have created a testing\_npys.py script :  
  
result :  
  
 

Form this we can see,  
Images shape: (845, 224, 224, 3): we have 845 images, each of size 224x224 with 3 channels (RGB). This is exactly what we want for model input.

Labels shape: (845, 11): we have 845 labels, each represented as a one-hot encoded vector of length 11 (which likely represents the number of unique categories or classes in our dataset).

**1.3) Model Selection and Training**

We will train a Convolutional Neural Network (CNN) model using the preprocessed dataset (images.npy and labels.npy) to classify landmarks.

Here , we used pre-trained ResNet50 model from tensorflow.keras.applications to build a classifier for recognizing the landmarks.

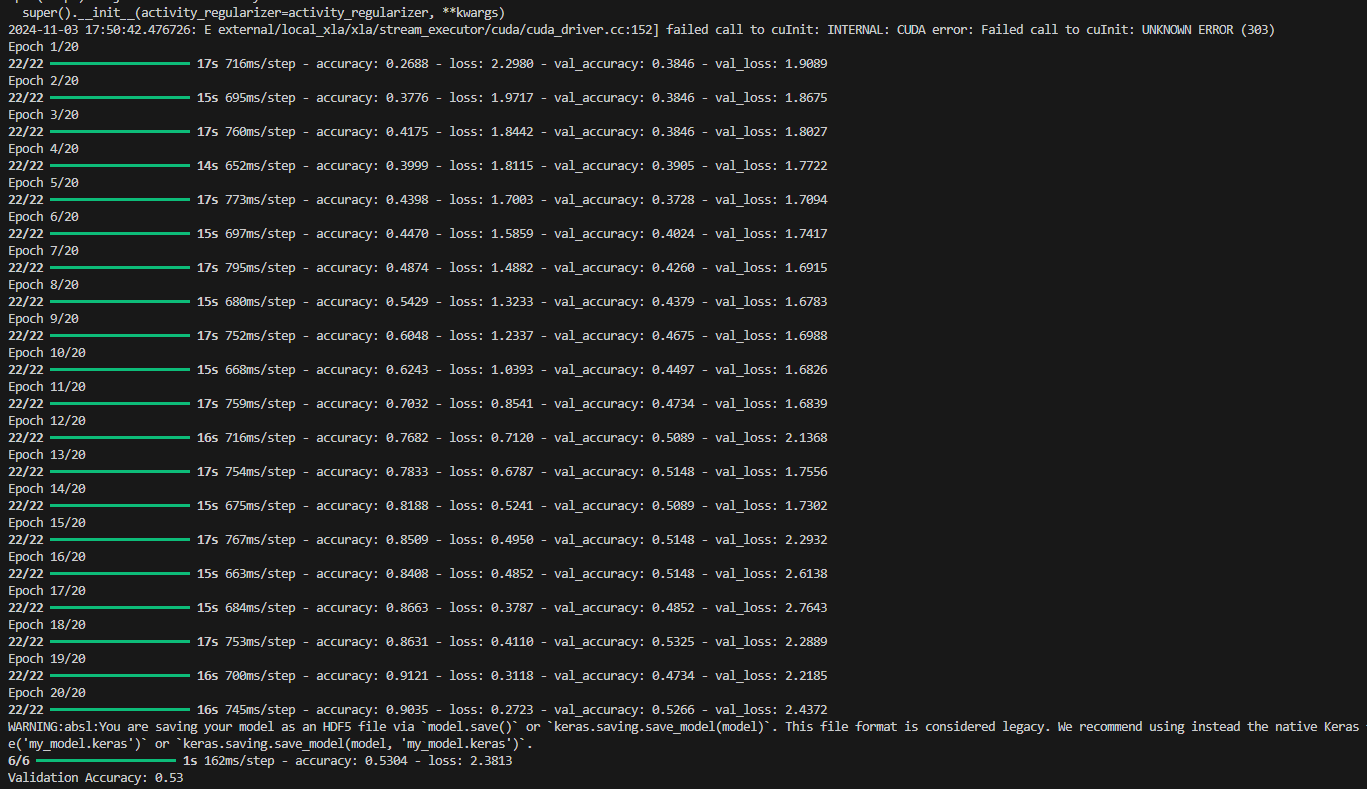
Tools used:  
  
TensorFlow: For building and training the model.

NumPy: To handle the loaded data.

Matplotlib: for visualizing training history if needed.  
  
we did this with train\_model.py script.

while running this script:  
  
1) we are used categorical cross-entropy and trained for 20 epochs

2) Saved the trained model as landmark\_recognition\_model.h5 for later use

Outputs:  
  


We have got 53% of validation accuracy for model.

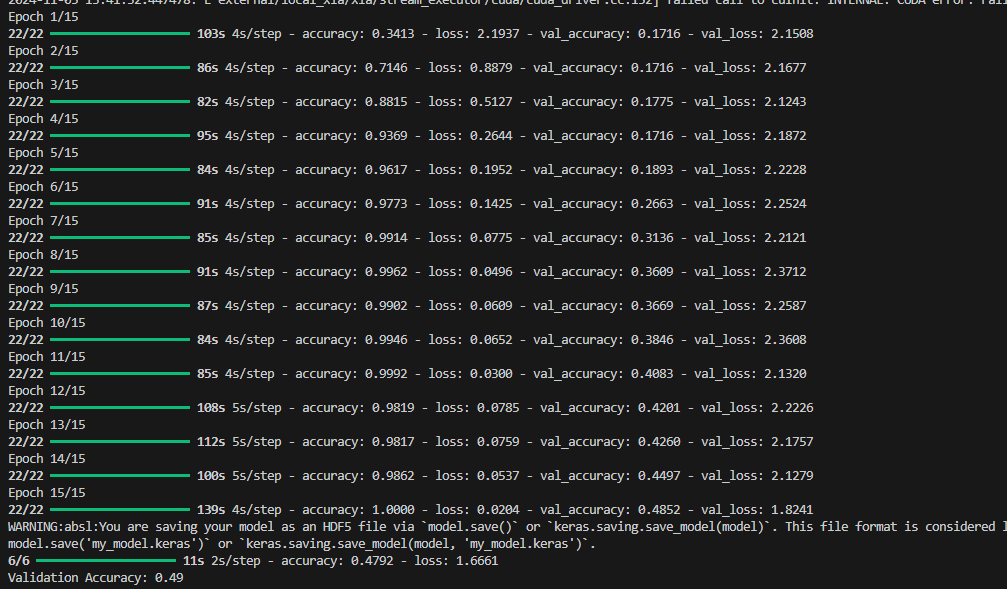
The current accuracy of **53%** for our model suggests that it could benefit from further optimization or experimentation with other models. So we have tried with Instead of a simple CNN, use **transfer learning** with a pre-trained model from Keras applications. These pre-trained models often achieve better accuracy because they are already trained on a large dataset.   
  
For example, we have chose ResNet50, a pre-trained model trained on ImageNet, and fine-tune it for your specific task and takes advantage of the features already learned by the model on a massive dataset. Achieves faster convergence and better accuracy with fewer data.  
  
**Process of using this new model:**  
  
ResNet50, a pre-trained model, for transfer learning on the landmark recognition task. Here’s what we did in this new approach:

Pre-Trained Model: ResNet50, which is already trained on ImageNet, is used as the base model.

Transfer Learning: We freeze the layers of ResNet50 so it retains its pre-learned features. We only train the custom classifier layers on top.

Previous approach used a custom CNN from scratch, requiring more training and typically yielding lower accuracy without a large dataset.

This new approach leverages an advanced pre-trained model to improve accuracy and requires less training effort.

Results of ResNet50 training approach:  
  


So here we got validation accuracy of 49%

Result comparison:  
  
1)Custom CNN achieved a slightly higher accuracy of 53%, while ResNet50 with fine-tuning reached 49%.

2) Fine-Tuning showed promise and adapted well to the dataset but may benefit from more training, further layer unfreezing, or even additi1onal data augmentation.

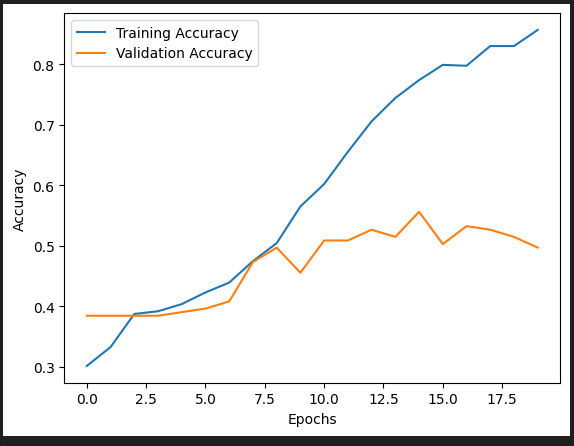
**Future Steps:**

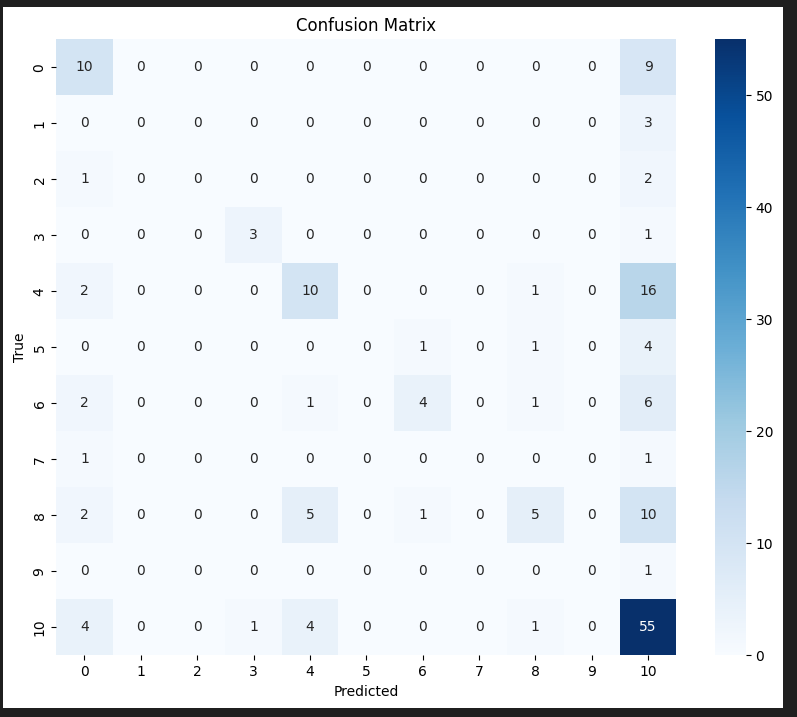
You can try unfreezing more layers to allow ResNet50 to better learn landmark-specific features.

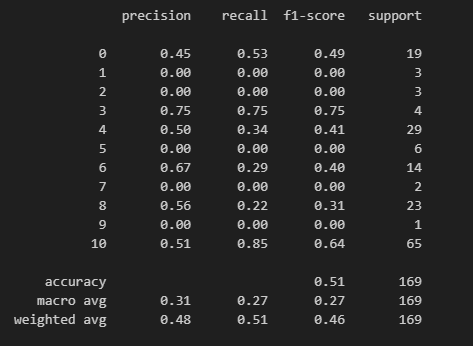
Consider adding more data augmentation to improve generalization.

Increase the number of epochs to help further fine-tune the unfreezed layers.

And finally developing full api and user interface to complete whole project.

Custom CNN result :  
  






Results for ResNet50:  
  
