Food-Image-Classification

Project Overview

The goal of this project is to build a model that can accurately classify images of food into predefined categories. With the rise of health and fitness apps, such a model can be integrated into applications to automatically detect and log consumed food items based on user-uploaded images.This project uses deep learning to identify 34 different types of food. It involves gathering food images, ensuring a balanced dataset, training and testing different AI models, and making the final model available online through a simple web app.

1. Data Collection

- The dataset used for this project consists of images of various food items categorized into different classes. Each image is labeled with its corresponding food category.
- The dataset, consisting of images categorized into 34 food classes, was acquired from Kaggle.
- To download the dataset, please visit the following link:
- click here

2.Data Balancing

- We made sure our food image dataset was fair by having the same number of images for each of the 34 food types.
- We used Python programs to do this, giving each food category 200 images.
- Then, we split the dataset into three parts:
    - Training Set: 150 images per class
    - Validation Set: 30 images per class
    - Testing Set: 20 images per class
- After balancing and splitting, we put the whole dataset on Google Drive so it's easy to use.

3.Environment Initialization and Library Loading

- We use Google Colaboratory for this project , because its free GPU access makes training deep learning models much faster than using a CPU.
- We started by putting our dataset on Google Drive, then created a new Colab notebook to write our code.

- First, we connected Colab to our Drive so we could use the data.
- Then, we imported all the Python libraries we needed, each one playing a specific role in the project.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import sys
import cv2
import sklearn
import tensorflow
import json
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential,Model
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten
from tensorflow.keras.activations import relu, softmax
from sklearn.metrics import confusion_matrix
from tensorflow.keras.models import load_model
import warnings
warnings.filterwarnings('ignore')
```

4.Image Data Preprocessing and Enhancement

- The dataset contains 34 different classes of food images.
- Each class of image is divided into training,testing and validation images wherein 50% of images from each class are considered as training, 20% of images from each class are considered as testing and the remaining 30% samples as validation samples.
- We used ImageDataGenerator to efficiently manage and enhance our image data. This tool lets us create variations of our training images (like rotations, flips, zooms, and color tweaks) on the fly, which helps our model learn more general features and avoid overfitting.
- Think of it as showing the model lots of slightly different versions of the same food, so it becomes better at recognizing it.
- This also helps with memory management because the images are loaded and processed as needed during training.
- We also rescale the images, which means we adjust the pixel values to be between 0 and 1.

```
train_datagen = ImageDataGenerator(
rescale=1.0/255,  # Normalize pixel values
rotation_range=20,  # Randomly rotate images
```

```
width_shift_range=0.2,  # Shift images horizontally
height_shift_range=0.2,  # Shift images vertically
shear_range=0.2,  # Apply shearing transformations
zoom_range=0.2,  # Randomly zoom in images
horizontal_flip=True,  # Flip images horizontally
fill_mode='nearest'  # Fill in missing pixels)
```

- We only rescale the validation and test images; we don't augment them, so we can accurately measure how well the model performs on real-world data.

```
valid_datagen = ImageDataGenerator(rescale=1.0/255)
test_datagen = ImageDataGenerator(rescale=1.0/255)
```

- Now we will set up three image generators for training, validation, and testing data usingthe flow_from_dataframe method provided by TensorFlow's ImageDataGenerator class.

```
train_images = train_datagen.flow_from_dataframe(train_path,target_size=(224,
224),class_mode='categorical',batch_size=32)
val_images = valid_datagen.flow_from_dataframe(validation_path,target_size=(224,
224),class_mode='categorical',batch_size=32)
test_images = test_datagen.flow_from_dataframe(test_path,target_size=(224,
224),class_mode='categorical',batch_size=32)
```

5.Implementing a Model

We tried three different deep learning models to classify food images:

1. Custom Model

- A fully custom CNN architecture is built from scratch using convolutional layers, max-pooling layers, and fully connected dense layers.
- The model includes 34 dense output neurons (one for each food category) with a softmax activation function.
- Key components include:
    - Kernel layers for feature extraction
    - Max pooling layers for reducing spatial dimensions
    - Hidden layers for learning complex patterns

2. VGG16 Model

- The VGG16 model is a pretrained CNN used for transfer learning.
- We import the pretrained VGG16 layers, freeze the initial layers, and fine-tune the final layers to adapt to our dataset.
- This approach benefits from the pre-learned hierarchical features, speeding up convergence.

3. ResNet Model

- Similar to VGG16, we use ResNet, another pretrained deep learning model that is optimized for residual learning.
- The ResNet layers are imported and fine-tuned for food classification.
- ResNet helps in solving vanishing gradient issues, making deep models more effective.
- Model Training & Saving
    - Each model is trained for 10 epochs due to computational limitations.
    - Higher epochs (e.g., 2000 epochs) can improve accuracy but require months of training time on standard hardware.
- After training, models are saved in one of the following formats:
    - HDF5 (.h5)
    - Pickle (.pkl)
    - Keras model format
- Saving the Trained Model

model.save("food_classification_model.keras") # Save in keras format

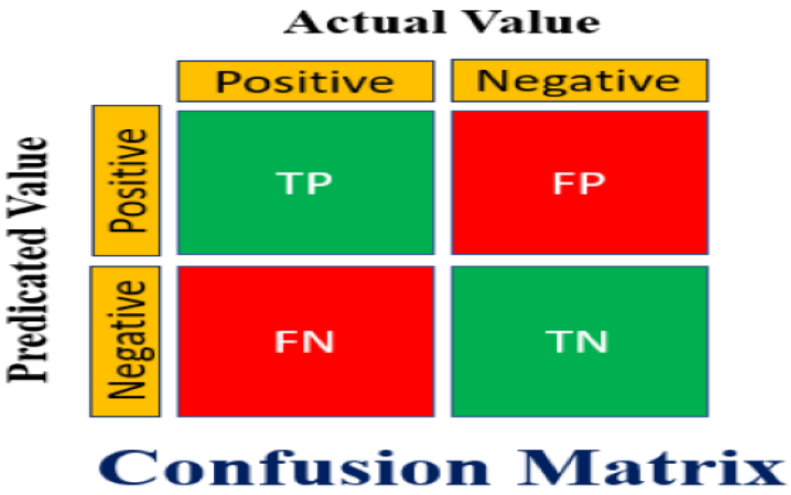
6.Model Evaluation and Validation

After training, we checked how well our model performed. Here's what we did:

- **Loaded the model:** We loaded the saved, trained model.
- **Made predictions:** We used the model to classify the test images and got its predictions.
- **Made a confusion matrix:** - We created a table that shows where the model made correct and incorrect classifications. - This helps us understand what the model is good at and where it struggles.
- **Calculated metrics:** From the confusion matrix, we calculated several important measures:

- **True Positives:** How many times the model correctly said "yes" when the answer was actually "yes."
- **True Negatives:** How many times the model correctly said "no" when the answer was actually "no."
- **False Positives:** How many times the model incorrectly said "yes" when the answer was actually "no."
- **False Negatives:** How many times the model incorrectly said "no" when the answer was actually "yes."

- Using these, we calculated:
  - **Precision** – The percentage of correctly predicted positive samples.
  - **Recall** – The percentage of actual positives correctly identified.
  - **F1-score** – A balance between precision and recall.
  - **Overall Accuracy** – The proportion of correct predictions across all test samples.

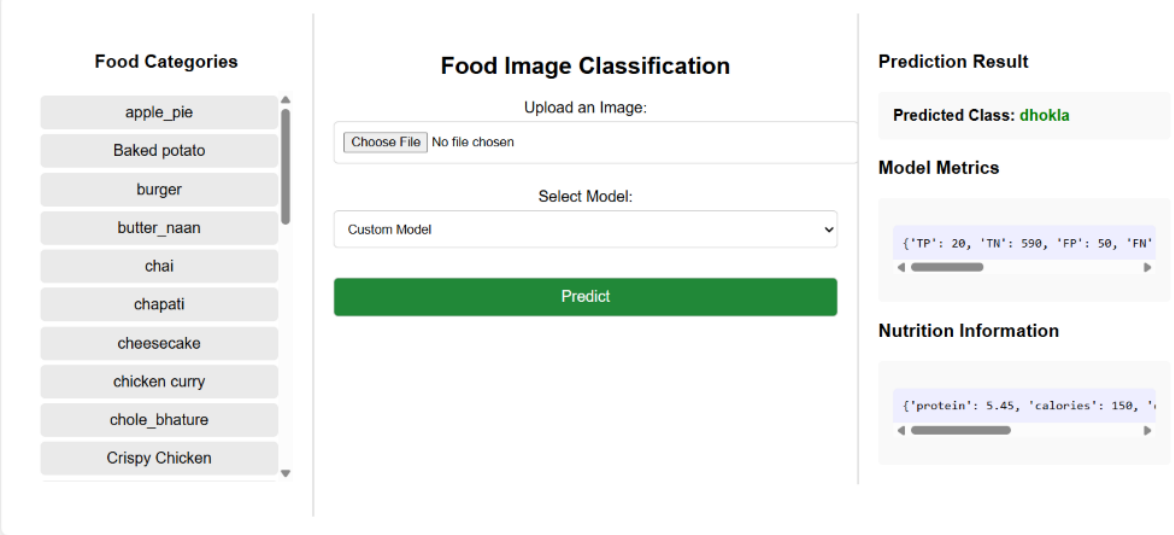


- Saved the results:
  - We saved all these numbers in a structured way (like a dictionary) and also as a JSON file, so we could easily look at them later and compare different models.

7.Application Deployment

- After training and evaluation, the model is deployed for web-based food image classification.
- Our trained model was deployed as a web application using Flask for the backend and HTML/CSS for the frontend.

- Flask serves the model, processing uploaded images and returning predictions, which are then displayed on the user-friendly web interface.



- We set up a local development environment using PyCharm (or any IDE) to build and test our Flask-based web application.
- This allowed us to simulate the real-world user experience, with users able to upload images and receive predictions in real time.

Thank you……

Submitted by

M.Lavanya