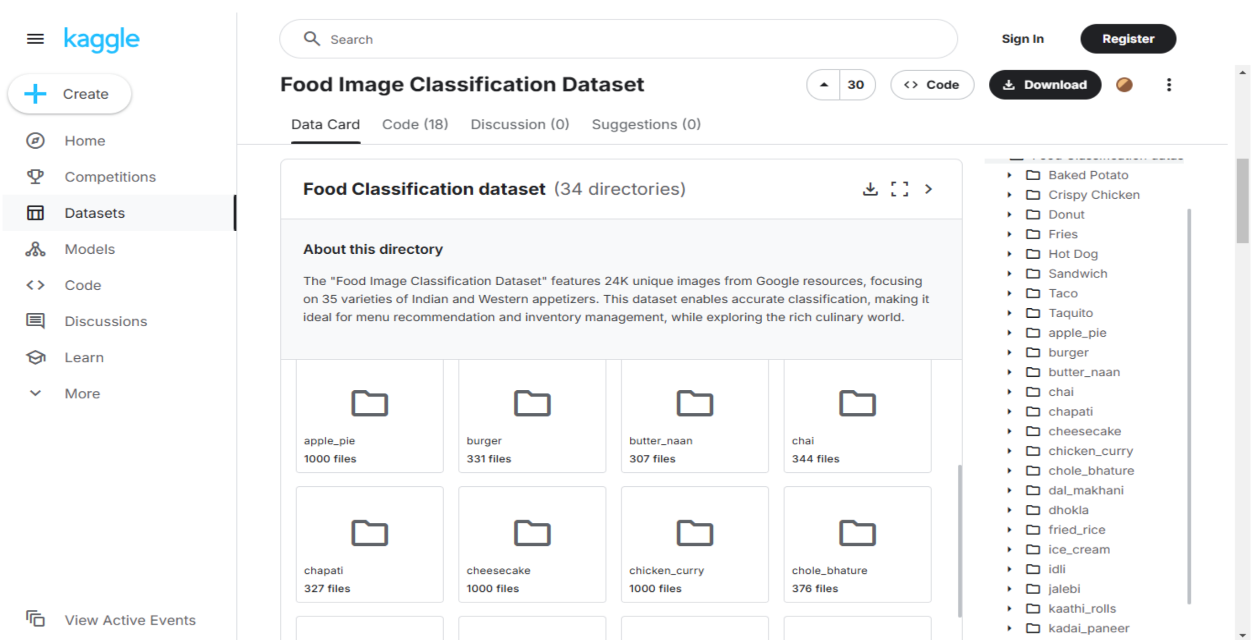
FOOD CLASSIFICATION PROJECT

Objective: The main goal of this project is to classify food items by uploading their images. This deep learning-based food classification model leverages multiple architectures to categorize 34 different food types. The project encompasses data collection, dataset balancing, model training, validation, and deployment using Flask.

1.DATA COLLECTION:

- For this food classification project, we require a high-quality dataset to train our deep learning model effectively.
- The dataset is sourced from Kaggle, containing images of 34 different food categories.
- It includes a diverse range of food items to improve model accuracy and generalization.
- You can download the dataset from the following link:
- Food classification dataset



2.DATA BALANCING:

- To improve model performance and ensure fair representation, data balancing techniques are applied, making the dataset evenly distributed across all 34 food categories.
- We use Python scripts to balance the dataset, ensuring each class contains exactly **200 images**.
- The dataset is then split into three subsets:
  Training Set: 150 images per class
  Validation Set: 30 images per class
  Testing Set: 20 images per class
- Finally, the balanced dataset is uploaded to Google Drive for easy access and further processing.

3. Development Environment & Library Imports:

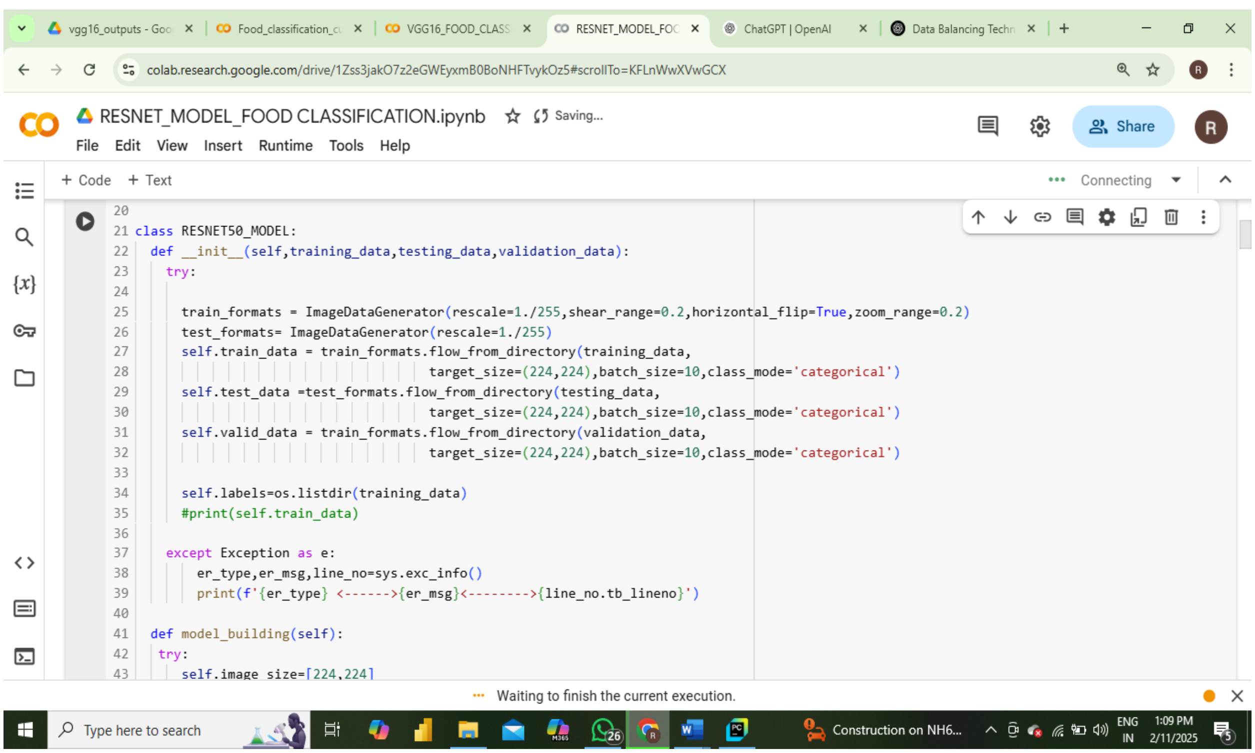
- We use **Google Colaboratory** for this project because it provides free access to **GPUs**, making deep learning model training more efficient compared to CPUs.
- After uploading the dataset to **Google Drive**, a new **Colab notebook** is created to begin the coding process.
- The first step is to **mount Google Drive** to access the dataset:
- Once the drive is mounted, we navigate to the dataset directory and verify the files before proceeding.
- Next, we **import the necessary libraries**, which are used for various tasks such as numerical computations, data preprocessing, visualization, and deep learning model development.

```
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
import os
import sys
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Conv2D,MaxPool2D,Flatten
from tensorflow.keras.activations import relu,sigmoid
from sklearn.metrics
importaccuracy_score,confusion_matrix,classification_report
```

```python
import warnings
warnings.filterwarnings('ignore')
```

4. Image Data Preprocessing & Augmentation:

- **ImageDataGenerator** is used to efficiently load and augment image data, creating variations such as **rotations, flips, zoom, and color adjustments** to enhance model generalization.
- Data augmentation **increases the effective dataset size**, helping the model learn more robust features and reducing overfitting.
- It also enables **real-time image loading**, reducing memory usage and improving training efficiency.
- **Rescaling images** standardizes their size and normalizes pixel values, ensuring consistency and improving model performance. This helps **reduce computational load** and allows the model to **converge faster** during training.
- These techniques **ensure effective image preprocessing**, making the dataset more diverse and improving the **generalization ability** of the deep learning model.



5.MODEL DEVELPOPMENT:

We experiment with **three different deep learning models** for food classification:

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.h5')  # Save in HDF5 format
```

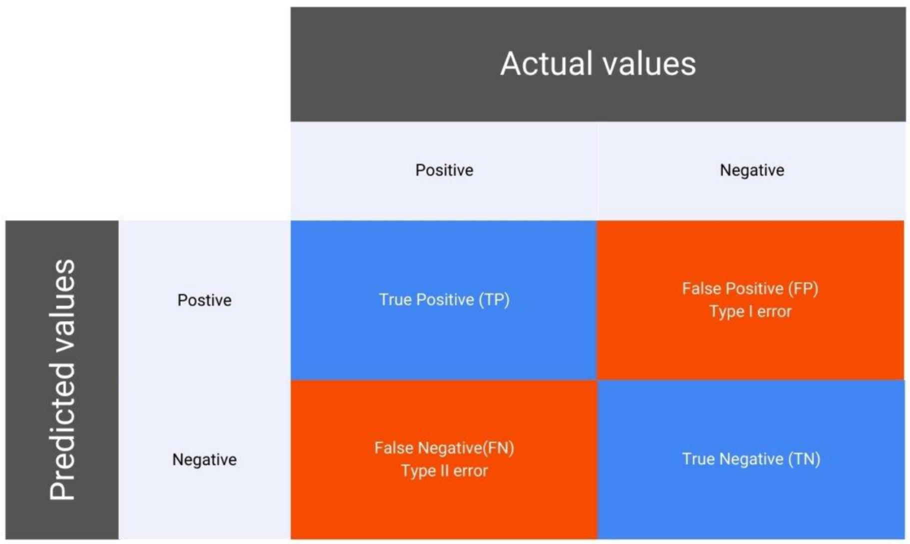
6.MODEL EVALUATION:

After training the model, we evaluate its performance using various metrics to assess accuracy and reliability.

Steps for Model Evaluation:

- **Load the Trained Model**
    - The trained model is loaded to make predictions on the test dataset.
- **Make Predictions**
    - The model classifies test images, generating predicted labels for each input.
- **Compute the Confusion Matrix**
    - A confusion matrix is created using predicted values and actual labels to analyze model performance.
- **Calculate Performance Metrics**
    - Extract key values from the confusion matrix:
        - **True Positives (TP)** – Correctly predicted positive samples
        - **True Negatives (TN)** – Correctly predicted negative samples
        - **False Positives (FP)** – Incorrectly predicted positive samples
        - **False Negatives (FN)** – Incorrectly predicted negative samples
    - Using these values, we calculate:
        - **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.

- **Store Evaluation Results**
  - All calculated values are saved in a structured format, such as a dictionary, for future reference.
  - The evaluation results are also stored in a JSON file for further analysis and model comparison.



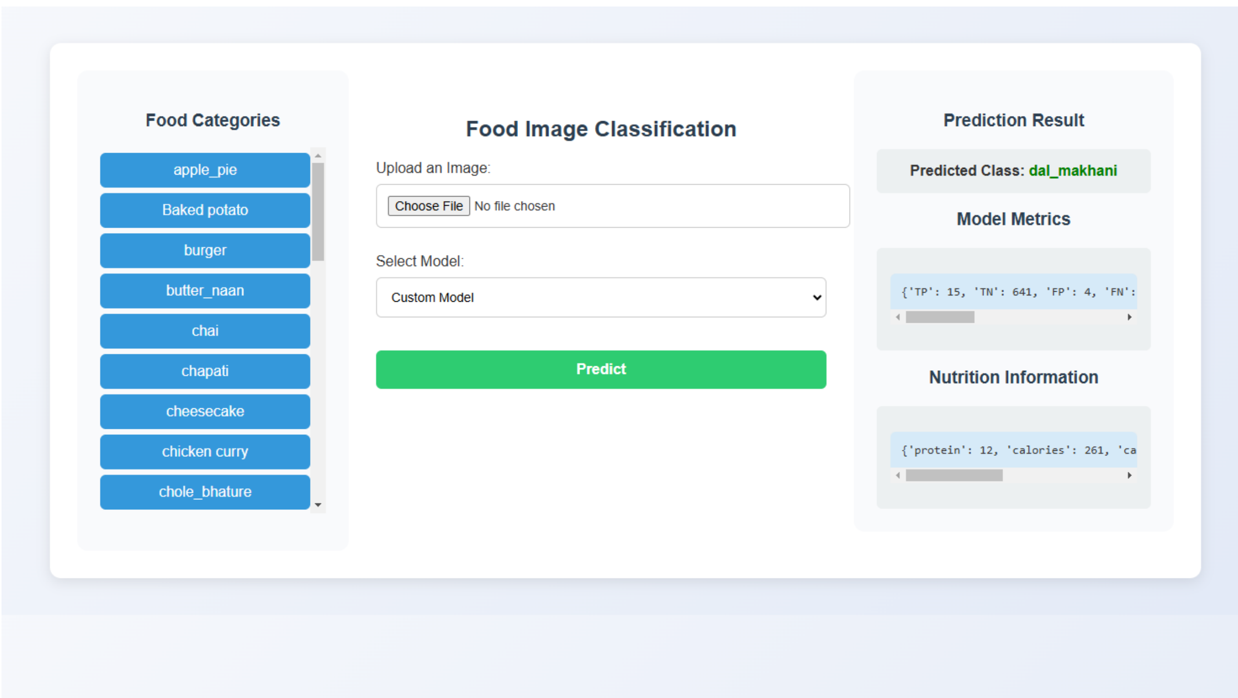
7.DEPLOYMENT:

Once the model is trained and evaluated, the next step is to deploy it so users can classify food images through a web interface.

Deployment Process:

- **Use Flask for Backend Development:**
  - Flask, a lightweight Python web framework, is used to serve the deep learning model.
  - The trained model is loaded in Flask to process user-uploaded images and return predictions.
- **Build the Frontend with HTML & CSS:**
  - A simple **HTML/CSS** interface is designed to allow users to upload food images.
  - The interface sends the uploaded image to the backend for classification.

- **Integrate Flask with HTML:**
    - Flask handles image uploads, processes them, and returns classification results.
    - The prediction results are displayed on the web page in a user-friendly manner.
- **Run the Web Application Locally:**
    - The Flask app is developed and tested in **PyCharm** (or any preferred IDE).
    - The web app runs locally, allowing users to upload images and get real-time predictions.
- **Optional: Deploy to a Cloud Platform**
    - The application can be deployed on cloud services like **Heroku, AWS, or Google Cloud** to make it accessible online.



Repository link : [food _classication_project_repository]

*** completed***

Thank you....

Submitted by

V.J.V.RAGHAVA SAI