**Food Images Classification Project Documentation**

**Project Title:**

Multilabel Classification using Custom and Transfer Learning Models on Food Images Dataset.

**Objective:**

* To develop an end-to-end deep learning solution using the Food Images dataset. The task includes:
  + 1. Creating a multi label classification model using:
       - * A custom deep learning model.
         * Transfer learning models (VGG16 and ResNet50).
    2. Design a frontend interface to upload images and predict the output class and display the predicted class as well as the nutrition info about the predicted class using Flask and HTML.

**Dataset Details:**

Dataset name is Food Images Classification dataset. It consists of 10,000 images, out of all 6,500 images belongs to training and remaining 3,500 images belongs to training and validation. There are 34 different classes which consists of different food images. The images are color and vary in size.

**CUSTOM CNN MODEL**

**Data Preprocessing:**

* I have created new folder in the desktop for this project.
* I worked out this project on pycharm community edition.
* At first, create the virtual environment for the project directory and then install the required libraries using pip install command.
* Download the dataset from Kaggle.
* Import the required libraries at the beginning of the program.
* We need to preprocess data using Image Data Generator class from TensorFlow.
* Using Image Data Generator, we should rescale the image to normalize the pixel values, set shear ang zoom range, add rotation range, width shift range, height shift range and also horizontal flip for training images.
* Apply only rescale to normalize the values for testing and validation images.
* Then we need to load the using flow from directory, which is a method of Image data generator class.
* We should pass the training, testing and validation images path to the method and also select the target size, batch size, class mode and color mode separately to load the data for training, testing and validation.
* Now, extract the class labels into a list.

**Custom CNN Architecture:**

* We initialize the model by using Sequential method.
* Then add conv layers to the model and give the input shape, kernel size (3,3) and relu activation function.
* After we add Batch Normalization.
* Then add the max pooling layer of pool size (2,2).
* Finally add the Dropout.
* Repeat the same process as many times you want based on the data.
* Then we need to Global Average Pooling 2D to flatten the data, it works similar to flatten but it reduces the trainable parameters, which helps the model to train and predict quicker than ever.
* After that, add the fully connected dense layers to the model with relu activation.
* Later add the Batch Normalization and Dropout after every dense layer.
* And finally add the dense layer to the model for output layer. Here our activation will be softmax.
* Softmax predicts the probabilities of the classes, where it gives the highest prob value for the predicted class.
* Using argmax we can retrieve the index of predicted class.

**Model Training:**

* After creating the architecture, we should compile the model using adam optimizer and loss will be categorical crossentropy.
* After compiling, then we need to fit the model with the training data and also validation data with batch size 10. Set epochs to 30 for model training.
* After training save the model.
* If u have NVidia GPU in your system, then aim for higher epochs for better results.
* The more the model trains, the better the outcomes.

**Model Evaluation:**

* By passing test data to evaluate method, the outcomes will be test loss and test accuracy.
* For finding more metrics like precision, f1 score and confusion matrix, we need to find the y actual and y predicted values.
* Then we can find the values of necessary metrics like precision, recall, f1 score, accuracy and loss.
* After that we can find the confusion matrix which is 34x34 in size.
* I stored all the metrics like TP, TN. FP, FN for each class in the json file.
* And also calculated the overall values by adding them.
* By using them we can calculate accuracy, f1 score, precision and also recall.
* If you find your model has high accuracy, low precision and recall then you can consider that your model is not performing well.
* Accuracy alone is’nt enough for good outcomes.
* Stored all this information in the json file.

**VGG16 and ResNet50 PRETRAINED MODEL**

**Data Preprocessing:**

* I have created new folder in the desktop for this project.
* I worked out this project on pycharm community edition.
* At first, create the virtual environment for the project directory and then install the required libraries using pip install command.
* Download the dataset from Kaggle.
* Import the required libraries at the beginning of the program.
* We need to preprocess data using Image Data Generator class from TensorFlow.
* Using Image Data Generator, we should rescale the image to normalize the pixel values, set shear ang zoom range, add rotation range, width shift range, height shift range and also horizontal flip for training images.
* Apply only rescale to normalize the values for testing and validation images.
* Then we need to load the using flow from directory, which is a method of Image data generator class.
* We should pass the training, testing and validation images path to the method and also select the target size, batch size, class mode and color mode separately to load the data for training, testing and validation.
* Now, extract the class labels into a list.

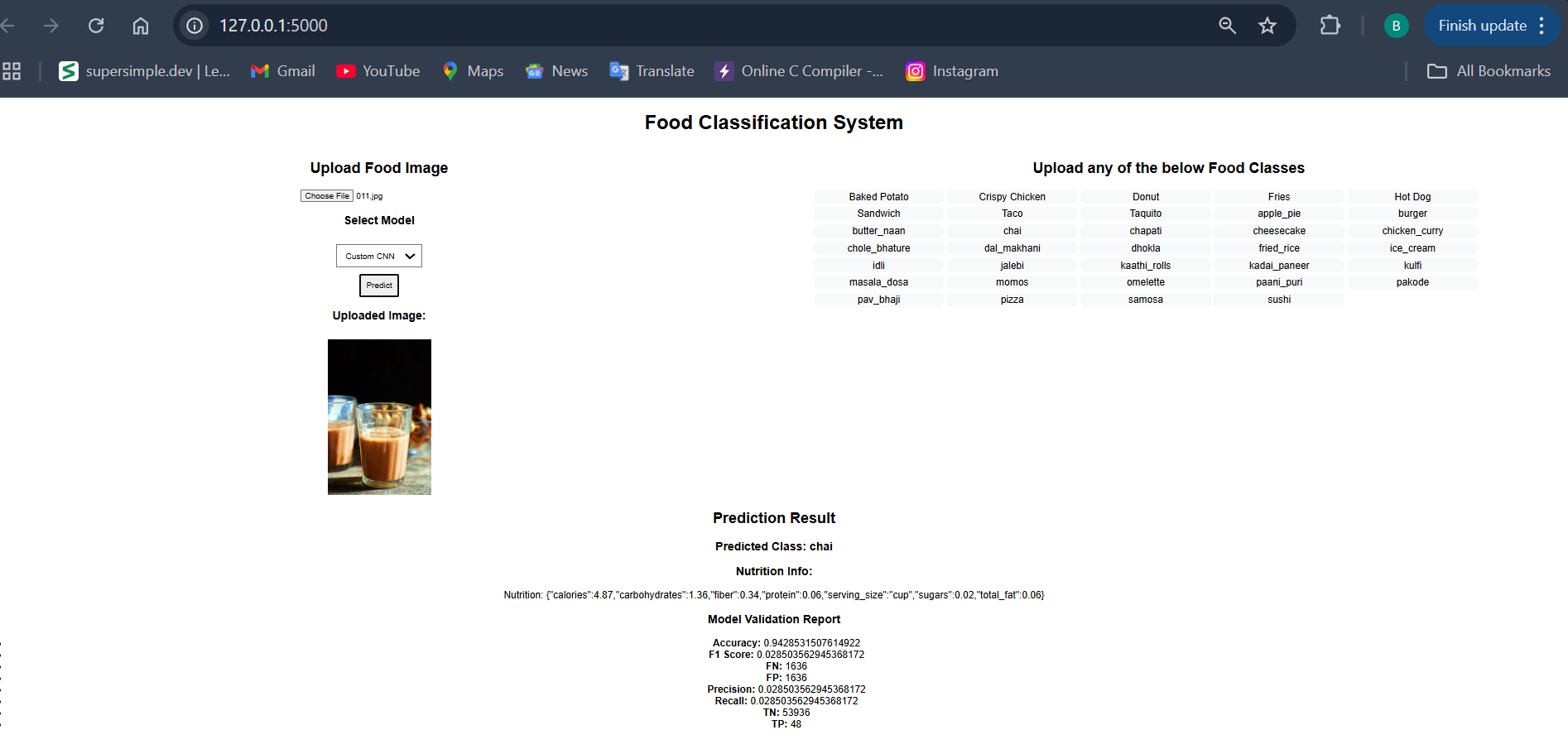
**Build and Train VGG16 and ResNet50:**

* Build the base model using imported functions and include top = false, input shape = (224, 224, 3), weights = ‘imagenet’.
* By using imagenet weights, the model is predicting better.
* Then we need to add the custom layers with the help of sequential function.
* First base model, then flatten the data using Global Average Pooling 2D and then add dense layers with relu activation function, Batch Normalization and dropout.
* Finally add the output dense layer with softmax activation.
* Then compile the model.
* Then train the model using fit method by giving epochs and batch size.
* Finally save the model.

**Model Evaluation:**

* By passing test data to evaluate method, the outcomes will be test loss and test accuracy.
* For finding more metrics like precision, f1 score and confusion matrix, we need to find the y actual and y predicted values.
* Then we can find the values of necessary metrics like precision, recall, f1 score, accuracy and loss.
* After that we can find the confusion matrix which is 34x34 in size.
* I stored all the metrics like TP, TN. FP, FN for each class in the json file.
* And also calculated the overall values by adding them.
* By using them we can calculate accuracy, f1 score, precision and also recall.
* If you find your model has high accuracy, low precision and recall then you can consider that your model is not performing well.
* Accuracy alone is’nt enough for good outcomes.
* Stored all this information in the json file.

**Model Deployment:**

* For deployment, I have created two files app.py and index.html.
* I have created the index html file inside the templates folder.
* In that app file I have used Flask and other necessary libraries to do the backend work for the deployment.
* Index html file is for the frontend code for the user to upload the image and choose the model and predict.
* I have saved the weights of custom CNN, VGG16 and ResNet50 models while training them.
* I did used them in app python file to make predictions and also, I have preprocessed image based on the model.
* After creating all this code, I run this on local server and got the predictions.
* Predictions are not accurate enough.
* The interface displays the uploaded image along with the predicted class name and its nutritional values.

**Challenges Faced:**

* It is very difficult to train transfer learning models like VGG16 and ResNet50 and also custom CNN in laptops with CPU.
* It is taking so much time without the GPU help.
* So, to make my work easy and efficient, I have trained them in google colab with GPU.
* Even with google colab GPU it is literally consuming hours of time and I did put lot of effort to complete this task on time.

**Conclusion:**

Finally, I can say that all three models are poor at predicting, but there is always room for improvement. By using fine tuning techniques, we can definitely improve and boost the performance the model.