

Configuration Manual

MSc Research Project
Data Analytics

Gautam Bhatia
Student ID: x22171118

School of Computing
National College of Ireland

Supervisor: Rejwanul Haque

National College of Ireland
Project Submission Sheet
School of Computing



Student Name:	Gautam Bhatia
Student ID:	x22171118
Programme:	Data Analytics
Year:	2018
Module:	MSc Research Project
Supervisor:	Rejwanul Haque
Submission Due Date:	20/12/2018
Project Title:	Configuration Manual
Word Count:	1117
Page Count:	11

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Gautam Bhatia
Date:	11th August 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission , to each project (including multiple copies).	<input type="checkbox"/>
You must ensure that you retain a HARD COPY of the project , both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Gautam Bhatia
x22171118

1 Introduction

This manual provides step-by-step information for configuring the food recognition and recipe retrieval assistant. The system is designed to help international students and users who don't know how to prepare the meal. By using image recognition models like CLIP and MobileNetV2, this system identifies food ingredients and potential allergens. This guide will walk you through the necessary steps to setup the environment and configure the system.

2 Environmental Setup

This section describes a list of all the tools and software that were used to complete the project successfully.

2.1 Hardware Requirements

The hardware specs used for this project were a 64-bit windows 11 operating system and 8GB of RAM. The processor used was an intel i5 (11th Gen). This figure shows the details of the hardware specifications used.

2.2 Software Requirements

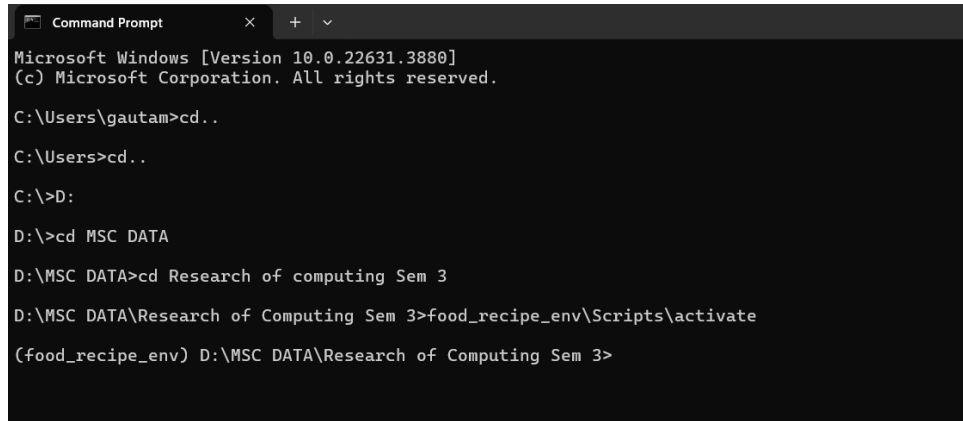
This section outlines the necessary software requirements needed to develop and run the food recognition and recipe assistant. Ensure that your system meets these software requirements. First is python programming language that was used for the development of this project. It is important to have python installed to run scripts and manage the app. For IDE author used visual studio because it is lightweight and better support with python programming language.

3 Virtual Environment

The 'food_recipe_env' virtual environment was created for this research. The following steps were taken to create and activate the environment in Visual Studio:

1. Opened the Command Prompt (CMD) by entering "cmd" in the Windows search bar, and then used the 'cd' command to change the directory to 'D:\MSC DATA\Research of Computing Sem 3'.

2. Created the virtual environment using the command 'python -m venv food_recipe_env'.
3. Activated the virtual environment in Visual Studio by running the command 'venv\Scripts\activate'.



```

Microsoft Windows [Version 10.0.22631.3880]
(c) Microsoft Corporation. All rights reserved.

C:\Users\gautam>cd..
C:\Users>cd..
C:\>D:
D:\>cd MSC DATA
D:\MSC DATA>cd Research of computing Sem 3
D:\MSC DATA\Research of Computing Sem 3>food_recipe_env\Scripts\activate
(food_recipe_env) D:\MSC DATA\Research of Computing Sem 3>

```

Figure 1: Virtual Environment Setup

4 Importing Libraries

There are some libraries that need to be installed from 'pip' command. To install the libraries we used this command 'pip (library name)' at the terminal of visual studio. Here are the all libraries author used to build this project.

```

import pandas as pd
import os
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator # type: ignore
from tensorflow.keras.models import Sequential # type: ignore
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout # type: ignore
from tensorflow.keras.applications import MobileNetV2 # type: ignore
from tensorflow.keras.optimizers import Adam # type: ignore
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras.preprocessing import image # type: ignore
from tensorflow.keras.models import load_model # type: ignore

```

Figure 2: Importing libraries for MobileNetV2

```

import pandas as pd
import torch
from transformers import CLIPProcessor, CLIPModel
from PIL import Image
from sklearn.metrics import confusion_matrix, classification_report, top_k_accuracy_score
from sklearn.model_selection import train_test_split
from tqdm import tqdm
import matplotlib.pyplot as plt
import seaborn as sns

```

Figure 3: Importing libraries for CLIP

5 Datasets

The data selected with the Indian food images and Food-101 datasets from Kaggle. The data of Indian food images dataset contains 4000 images with 80 different categories and Food-101 dataset contains of 1,01,000 images with 101 food classes. Then I combined the both dataset and extract the image paths and labels in the csv format and shuffle the dataset and display the combined dataset. see this figure that load the dataset and combined it.

```
> # Indian Food Images Dataset
indian_food_path = 'dataset\Indian Food Images\Indian Food Images'
indian_food_labels = os.listdir(indian_food_path)
indian_food = pd.DataFrame(columns=['img_path', 'label'])

for label in indian_food_labels:
    img_dir_path = os.path.join(indian_food_path, label)
    for img in os.listdir(img_dir_path):
        img_path = os.path.join(img_dir_path, img)
        indian_food.loc[indian_food.shape[0]] = [img_path, label]

# Food101 Dataset using metadata
food101_images_path = 'D:\VMSC DATA\Research of Computing Sem 3\food_recipe_env\dataset\food-101\food-101\images'
food101_meta_path = 'dataset\food-101\food-101\meta'

food101 = pd.DataFrame(columns=['img_path', 'label'])

# Read the train and test metadata files
with open(os.path.join(food101_meta_path, 'train.txt'), 'r') as file:
    train_files = file.readlines()

with open(os.path.join(food101_meta_path, 'test.txt'), 'r') as file:
    test_files = file.readlines()

# Combine train and test files into a single list
all_files = train_files + test_files

# Extract image paths and labels
for file in all_files:
    file = file.strip()
    label = file.split('/')[0]
    img_path = os.path.join(food101_images_path, f'{file}.jpg')
    if os.path.isfile(img_path):
        food101.loc[food101.shape[0]] = [img_path, label]

# Combine both datasets
food_combined = pd.concat([indian_food, food101], ignore_index=True)

# Shuffle the dataset
food_combined = food_combined.sample(frac=1).reset_index(drop=True)

# Display the combined dataset
print("Number of images", food_combined.shape[0])
print("Number of labels", food_combined['label'].nunique())
```

[4] ... Number of images 105000
Number of labels 181

Figure 4: Loading Datasets

6 Splitting Dataset

Now in this section author split the dataset into training, validation and testing ratio this step is important to train model on your specific dataset. The dataset was divide into three parts 80% for training, 10% for validation and 10% for testing.

7 Data Augmentation

train_generator, val_generator, and test_generator are created to feed the model with the augmented and rescaled images during training and evaluation.

```

[3] food_combined = pd.read_csv('Food_101&Indian_Food.csv')

[4]
train_ratio = 0.80
val_ratio = 0.10
test_ratio = 0.10

train_data, test_data = train_test_split(food_combined, test_size=val_ratio + test_ratio, random_state=42)
val_data, test_data = train_test_split(test_data, test_size=test_ratio / (val_ratio + test_ratio), random_state=42)

print("Train data shape:", train_data.shape)
print("Validation data shape:", val_data.shape)
print("Test data shape:", test_data.shape)

[5]
... Train data shape: (84000, 3)
Validation data shape: (10500, 3)
Test data shape: (10500, 3)

```

Figure 5: Splitting the Datasets

```

[6]
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_dataframe(
    train_data,
    x_col='img_path',
    y_col='label',
    target_size=(224, 224),
    batch_size=32,
    class_mode='sparse'
)

val_generator = val_datagen.flow_from_dataframe(
    val_data,
    x_col='img_path',
    y_col='label',
    target_size=(224, 224),
    batch_size=32,
    class_mode='sparse'
)

test_generator = test_datagen.flow_from_dataframe(
    test_data,
    x_col='img_path',
    y_col='label',
    target_size=(224, 224),
    batch_size=32,
    class_mode='sparse',
    shuffle=False
)

... Found 84000 validated image filenames belonging to 181 classes.
Found 10500 validated image filenames belonging to 181 classes.
Found 10500 validated image filenames belonging to 181 classes.

```

Figure 6: Data Augmentation

8 Building Model for Training

A base model using MobileNetV2 *MobileNet* (n.d.) (pre-trained on ImageNet) is created with the top layers removed (include_top=False), meaning the classification layer is excluded. A custom model is built on top of the base model, including a Global Average Pooling layer and fully connected (Dense) layers. The model is compiled using the Adam optimizer and sparse categorical cross-entropy loss, and then trained using the training and validation data. After training for 10 epochs, the model's performance is evaluated on the test set.

```
base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
base_model.trainable = False
epochs = 10
model = Sequential()
model.add(base_model)
model.add(GlobalAveragePooling2D())
model.add(Dense(512, activation='relu'))
model.add(Dense(food_combined['label'].nunique(), activation='softmax'))
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

Figure 7: MobileNetV2 Architecture

```
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.n // 64,
    epochs=epochs,
    validation_data=val_generator,
    validation_steps=val_generator.n // 64
)
```

Epoch 1/10
WARNING:tensorflow:From c:\Users\seautan\anaconda3\lib\site-packages\keras\src\tf_utils.py:492: The name tf.nn.conv2d is deprecated. Please use tf.nn.conv2d instead.
WARNING:tensorflow:From c:\Users\seautan\anaconda3\lib\site-packages\keras\src\engine\base_layer_utils.py:386: The name tf.nn.conv2d is deprecated. Please use tf.nn.conv2d instead.

Epoch	Step	Loss	Accuracy	Val Loss	Val Accuracy
1/10	2215s	2.7524	0.3574	2.1789	0.4632
2/10	2096s	2.2262	0.4489	1.9614	0.4981
3/10	3192s	2.1068	0.4718	1.9553	0.5059
4/10	4508s	2.0231	0.4897	1.9421	0.5154
5/10	3821s	1.9630	0.5017	1.9598	0.5086
6/10	2518s	1.9464	0.5051	1.9280	0.5116
7/10	2281s	1.8919	0.5159	1.8738	0.5354
8/10	2190s	1.8616	0.5253	1.9168	0.5246
9/10	2288s	1.8428	0.5290	1.9010	0.5177
10/10	2995s	1.8131	0.5355	1.9094	0.5320

Figure 8: Model Training

```
test_loss, test_acc = model.evaluate(test_generator)
print(f"Test accuracy: {test_acc}")
```

329/329 [=====] - 504s 2s/step - loss: 1.9418 - accuracy: 0.5230
Test accuracy: 0.5230476260185242

Figure 9: Model Testing

9 Fine-Tuning with Additional Training

Additionally, author tried to change some parameters or add some parameters to increase the accuracy of the model. The base model is further fine-tuned with a larger model that includes additional layers and dropout for regularization and fit the model to see the performance.

```
base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
base_model.trainable = False
epochs = 20
optimizer = Adam(learning_rate=0.0001)
model_1 = Sequential()
model_1.add(base_model)
model_1.add(GlobalAveragePooling2D())
model_1.add(Dense(1024, activation='relu'))
model_1.add(Dropout(0.33))
model_1.add(Dense(food_combined['label'].nunique(), activation='softmax'))
model_1.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model_1.summary()
```

WARNING:tensorflow:From C:\Users\gaum\anaconda3\lib\site-packages\keras\src\backend.py:1398: The name tf.executing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

WARNING:tensorflow:From C:\Users\gaum\anaconda3\lib\site-packages\keras\src\layers\normalization\batch_normalization.py:979: The name tf.nn.fused_batch_norm is deprecated. Please use tf.compat.v1.nn.fused_batch_norm instead.

Model: "sequential"

Layer (type)	Output Shape	Param #
mobilenetv2_1.00_224 (Functional)	(None, 7, 7, 1280)	2257984
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	0
dense (Dense)	(None, 1024)	1311744
dropout (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 181)	185525

=====
Total params: 3755253 (14.33 MB)
Trainable params: 1497269 (5.71 MB)
Non-trainable params: 2257984 (8.92 MB)
Total layers: 6
Trainable layers: 4

Figure 10: Fine-Tuning

```
history = model_1.fit(
    train_generator,
    steps_per_epoch=train_generator.n // 64,
    epochs=epochs,
    validation_data=val_generator,
    validation_steps=val_generator.n // 64
)
```

Epoch 1/20
WARNING:tensorflow:From C:\Users\gaum\anaconda3\lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\gaum\anaconda3\lib\site-packages\keras\src\engine\base_layer_utils.py:384: The name tf.executing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

1148/1148 [=====] - 3454s 3s/step - loss: 3.4975 - accuracy: 0.2365 - val_loss: 2.5014 - val_accuracy: 0.4198
Epoch 2/20
1148/1148 [=====] - 2262s 2s/step - loss: 2.5988 - accuracy: 0.3843 - val_loss: 2.1690 - val_accuracy: 0.4751
Epoch 3/20
1148/1148 [=====] - 2781s 2s/step - loss: 2.3747 - accuracy: 0.4249 - val_loss: 2.0590 - val_accuracy: 0.4937
Epoch 4/20
1148/1148 [=====] - 2975s 3s/step - loss: 2.2296 - accuracy: 0.4496 - val_loss: 1.9814 - val_accuracy: 0.5033
Epoch 5/20
1148/1148 [=====] - 2738s 2s/step - loss: 2.1523 - accuracy: 0.4651 - val_loss: 1.9059 - val_accuracy: 0.5184
Epoch 6/20
1148/1148 [=====] - 2458s 2s/step - loss: 2.0845 - accuracy: 0.4813 - val_loss: 1.8829 - val_accuracy: 0.5256
Epoch 7/20
1148/1148 [=====] - 2783s 2s/step - loss: 2.0256 - accuracy: 0.4932 - val_loss: 1.8400 - val_accuracy: 0.5354
Epoch 8/20
1148/1148 [=====] - 3043s 3s/step - loss: 1.9681 - accuracy: 0.5045 - val_loss: 1.8352 - val_accuracy: 0.5399
Epoch 9/20
1148/1148 [=====] - 2129s 2s/step - loss: 1.9439 - accuracy: 0.5074 - val_loss: 1.7894 - val_accuracy: 0.5477
Epoch 10/20
1148/1148 [=====] - 1833s 2s/step - loss: 1.8936 - accuracy: 0.5204 - val_loss: 1.7865 - val_accuracy: 0.5476
Epoch 11/20
...
Epoch 19/20
1148/1148 [=====] - 1989s 2s/step - loss: 1.6934 - accuracy: 0.5635 - val_loss: 1.7239 - val_accuracy: 0.5640
Epoch 20/20
1148/1148 [=====] - 2358s 2s/step - loss: 1.6787 - accuracy: 0.5639 - val_loss: 1.6882 - val_accuracy: 0.5673
Output is truncated. View as a [scrollable element](#) or open in a [text editor](#). Adjust cell output [settings](#).

Figure 11: Model training

10 Saving Model

The trained model is saved as (DishPrediction.h5),(food_recognition_model_epochs_20.h5) Training and validation accuracy/loss over the epochs are plotted to visualize the model's performance.

11 Prediction Function

After loading the model of dish prediction.H5 the code includes a prediction function predict_dish that preprocesses an image, makes a prediction using the trained model, and returns the predicted class. get_class_name maps the predicted class index to the actual class name.

```
model = load_model('Models\DishPrediction.h5')

def preprocess_image(img_path):
    img = image.load_img(img_path, target_size=(224, 224))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = img_array / 255.0
    return img_array

def predict_dish(img_path, model):
    img_array = preprocess_image(img_path)
    predictions = model.predict(img_array)
    predicted_class = np.argmax(predictions, axis=1)
    return predicted_class[0]

def get_class_name(class_index, label_map):
    return label_map[class_index]

food_combined = 'Food_101&Indian_Food.csv'
food_combined = pd.read_csv(food_combined)
labels = sorted(food_combined['label'].unique())
label_map = {index: label for index, label in enumerate(labels)}

img_path = 'dataset\prediction images\escargots.jpeg'

predicted_class_index = predict_dish(img_path, model)
predicted_class_name = get_class_name(predicted_class_index, label_map)

print(f"The predicted dish is: {predicted_class_name}")
```

1/1 [=====] - 2s 2s/step
The predicted dish is: escargots

Figure 12: Model Prediction

12 CLIP Model

After importing the libraries and loading the clip model and processor to implement our project, the code checks the device configuration and loads the pre-trained model ('clip-vit-base-patch32') CLIP (n.d.). This model is responsible for processing both images and text.

13 Loading and splitting the dataset

Then load the dataset, which is food_combined, which contains image paths and their corresponding dish labels and dish_ingredients, which contains dish labels and their ingredients. The food_combined dataset is split into training and testing using 80–20so the train shape is 94,500 and the test shape is 10,500, after splitting the dataset next step is to create label maps and preprocess the data for applying the model.

```

# Device configuration
device = "cuda" if torch.cuda.is_available() else "cpu"

# Load the CLIP model and processor
clip_model = CLIPModel.from_pretrained("openai/clip-vit-base-patch32")
clip_processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32")
clip_model.to(device)

WARNING:tensorflow:From c:\Users\gautam\anaconda3\lib\site-packages\keras/src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated.

CLIPModel(
  (text_model): CLIPTextTransformer(
    (embeddings): CLIPTextEmbeddings(
      (token_embedding): Embedding(49408, 512)
      (position_embedding): Embedding(77, 512)
    )
    (encoder): CLIPEncoder(
      (layers): ModuleList(
        (0-11): 12 x CLIPEncoderLayer(
          (self_attn): CLIPAttention(
            (k_proj): Linear(in_features=512, out_features=512, bias=True)
            (v_proj): Linear(in_features=512, out_features=512, bias=True)
            (q_proj): Linear(in_features=512, out_features=512, bias=True)
            (out_proj): Linear(in_features=512, out_features=512, bias=True)
          )
          (layer_norm1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
          (mlp): CLIPMLP(
            (activation_fn): QuickGELUActivation()
            (fc1): Linear(in_features=512, out_features=2048, bias=True)
            (fc2): Linear(in_features=2048, out_features=512, bias=True)
          )
          (layer_norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        )
      )
    )
    ...
    (post_layernorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
  )
  (visual_projection): Linear(in_features=768, out_features=512, bias=False)
  (text_projection): Linear(in_features=512, out_features=512, bias=False)

```

Figure 13: Loaded CLIP model and processor

```

# Load the dataset
food_combined = pd.read_csv('Food_101&Indian_Food.csv')
dish_ingredients = pd.read_csv('dish_ingredients.csv')

# Define train, validation, and test ratios
train_ratio = 0.80
test_ratio = 0.10

train_data, test_data = train_test_split(food_combined, test_size=test_ratio, random_state=42)

print("Train data shape:", train_data.shape)
print("Test data shape:", test_data.shape)

Train data shape: (94500, 3)
Test data shape: (10500, 3)

```

Figure 14: Loading and splitting the dataset

14 Label and preprocess the data

The unique dish labels are mapped to indices ('label_map') for easy handling of the model prediction and then the 'preprocess_image_for_clip' function loads and preprocesses an image for the CLIP model. It makes sure that the image is in RGB format and handles the errors if the image cannot be opened.

```
# Load labels and map to indices
labels = sorted(dish_ingredients['label'].unique())
label_map = {label: index for index, label in enumerate(labels)}

# Inverse mapping to get label names from indices
index_to_label = {index: label for label, index in label_map.items()}
```

Figure 15: Label Map

```
def preprocess_image_for_clip(img_path):
    try:
        img = Image.open(img_path).convert("RGB")
        return img
    except Exception as e:
        print(f"Error opening image {img_path}: {e}")
        return None

def predict_dish_with_clip(img, model, processor, index_to_label):
    if img is None:
        return None, None

    # Prepare text input: list of labels (strings)
    text_labels = list(index_to_label.values())

    # Process text and image with processor
    inputs = processor(text=text_labels, images=img, return_tensors="pt", padding=True)

    # Make predictions
    with torch.no_grad():
        outputs = model(**inputs.to(device))
        logits_per_image = outputs.logits_per_image
        probs = logits_per_image.softmax(dim=-1)

    # Get predicted class index
    predicted_class_index = probs.argmax().item()

    return predicted_class_index, probs
```

Figure 16: Preprocess the images

15 Dish prediction with CLIP

This function takes an image, processes it with the CLIP model to predict the dish, and then prepares text inputs by listing all possible dish labels using the processor to handle both text labels and the image, then passes the processed inputs to the CLIP model to get prediction probabilities. This prediction function will predict our dish name, ingredients, and allergen information. The model was tested on the image of a pizza that I made at home. I took a snapshot of the pizza and then uploaded it to my model. I saw that the CLIP model accurately predicted the pizza ingredients and allergen information.

```

img_path = 'dataset\\prediction_images\\homemade_pizza.jpg'

# Make prediction
predicted_class_index_clip = predict_dish_with_clip(img_path, clip_model, clip_processor, index_to_label)

# Retrieve the predicted class name
predicted_class_name_clip = index_to_label[predicted_class_index_clip]

print(f"The predicted dish using CLIP is: {predicted_class_name_clip}")

# List of allergens
allergens = [
    "peanut", "milk", "soy", "wheat", "tree nut", "shellfish", "fish", "egg",
    "sesame", "gluten", "mustard", "celery", "sulfite", "lupin", "mollusk",
    "corn", "legumes", "meat", "poultry", "fruits", "vegetables", "dairy",
    "yeast", "garlic", "onion"
]

# Get ingredients for the predicted dish
predicted_ingredients = dish_ingredients[dish_ingredients['label'] == predicted_class_name_clip]['ingredients'].values[0]

# Check for allergens
predicted_allergens = [allergen for allergen in allergens if allergen in predicted_ingredients]

print(f"Ingredients: {predicted_ingredients}")
print(f"Potential allergens: {predicted_allergens}")

```

Unused or unrecognized kwargs: padding.
The predicted dish using CLIP is: pizza
Ingredients: 500 g bread flour(3 3/4 cups), 2 1/2 tsp Dry Yeast instant or active (10 grams), 3/4 tsp Table Salt(5 grams), 3/4 tsp Sugar, plus a pinch (about 3 grams), 1 1/2 cup water at room temperatur
Potential allergens: ['onion']

Figure 17: Dish prediction

16 Model Evaluation

This function evaluates the model on the test set by iterating through each test image, making predictions and comparing them with the true labels. It then collects predictions and true labels figured into the confusion matrix and shows the classification report for understanding the model performance, as well as calculating the top-5 accuracy, which checks if the correct label is among the top 5 predicted probabilities.

```

def evaluate_model(test_data, model, processor, index_to_label):
    predictions = []
    true_labels = []
    all_probs = []

    for _, row in tqdm(test_data.iterrows(), total=test_data.shape[0]):
        img_path = row['img_path']
        true_label = row['label']

        # Predict the label
        img = preprocess_image_for_clip(img_path)
        predicted_class_index, probs = predict_dish_with_clip(img, model, processor, index_to_label)

        if predicted_class_index is not None:
            predicted_label = index_to_label[predicted_class_index]

        # Collect predictions and true labels
        predictions.append(predicted_label)
        true_labels.append(true_label)
        all_probs.append(probs.cpu().numpy().flatten())

    # Compute confusion matrix
    cm = confusion_matrix(true_labels, predictions, labels=list(index_to_label.values()))

    # Classification report
    report = classification_report(true_labels, predictions, target_names=list(index_to_label.values()))

    # Top-5 accuracy
    all_probs = torch.tensor(all_probs)
    top5_accuracy = top_k_accuracy_score(true_labels, all_probs, k=5, labels=list(index_to_label.values()))

    return cm, report, top5_accuracy

```

Figure 18: Model Evaluation Function

```

def plot_confusion_matrix(cm, labels):
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.title('Confusion Matrix')
    plt.show()

# Evaluate the model
cm, report, top5_accuracy = evaluate_model(test_data, clip_model, clip_processor, index_to_label)
print(f"Top-5 accuracy on test set: {top5_accuracy:.2f}")
print("Classification Report:")
print(report)

# Plot the confusion matrix
plot_confusion_matrix(cm, list(index_to_label.values()))

```

Figure 19: Evaluation metrics

References

CLIP (n.d.). Hugging Face. Available at: https://huggingface.co/docs/transformers/en/model_doc/clip.

MobileNet (n.d.). Keras Documentation. Available at: <https://keras.io/api/applications/mobilenet/>.