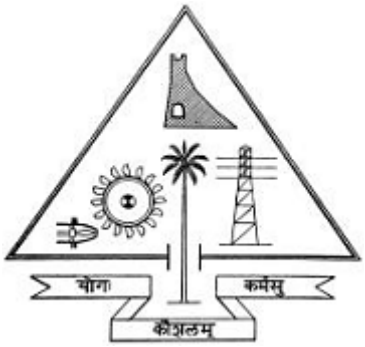
FOOD SCANNER APPLICATION

Mini project is submitted in partial fulfillment of the requirements for the award of the degree of ***Master of Computer Applications*** *of the* ***APJ Abdul Kalam Technological University***

submitted by

ANUSREE JNANAKRISHNAN (TCR20MCA-2011)

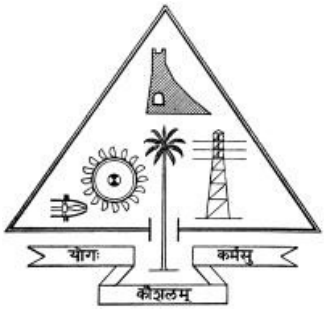


DEPARTMENT OF COMPUTER APPLICATIONS

GOVERNMENT ENGINEERING COLLEGE

THRISSUR - 680009

DEPARTMENT OF COMPUTER APPLICATIONS
GOVERNMENT ENGINEERING COLLEGE, THRISSUR
THRISSUR, KERALA STATE, PIN 680009



CERTIFICATE

This is to certify that the mini project titled
"Food Scanner Application"
is a bonafide work done by
ANUSREE JNANAKRISHNAN (TCR20MCA 2011)
under my supervision and guidance, and is submitted in February 2022 in partial fulfillment of the requirements for the award of the Degree of Master of Computer Applications from APJ Abdul Kalam Technological University(KTU).

Asst.Prof.Hussain Ahmed	Dr. Sminesh C N	Dr. Sminesh C N
Project Guide	**Project Coordinator**	**Head of Department**

Place : THRISSUR
Date :

DECLARATION

I hereby declare that the mini project named, **Food Scanner Application**, is my own work and that, to the best of my knowledge and belief, it contains no material previously published another person nor material which has been accepted for the award of any other degree or course of the university or any other institute of higher learning, except where due acknowledgement and reference has been made in the text.

Signature

ANUSREE JNANAKRISHNAN

Place : THRISSUR

Date :

ACKNOWLEDGEMENT

I would like to thank Computer Application Department of GEC Thrissur, for giving me this opportunity to pursue this project and successfully complete it.

I'm highly indebted to our Head of Department **Dr. Sminesh C N** for his guidance and constant supervision as well as for providing necessary information regarding the project and also for his support in completing the project.

I would like to express our deep gratitude to our project guide **Asst.Prof. Hussain Ahmed** for his timely suggestions and encouragement and guiding through the processes involved in the presentation of the project and the preparation of the Report.

I express my heart-felt gratitude to project coordinator, **Dr. Sminesh C N**, for his committed guidance, valuable suggestions, constructive criticisms and precious time that he invested throughout the work.

I sincerely thank all other faculties of MCA department for guiding through the processes involved in the project .

ABSTRACT

Automatic image-based food recognition is a particularly challenging task. Food, which is an important part of everyday life in all cultures, constitutes a particular challenge in the domain of image classification, due to its visually intricate complexity, as well as the semantic complexity stemming from the variation in the mixing of various ingredients practiced by regional communities. Traditional image analysis approaches have achieved low classification accuracy in the past, whereas deep learning approaches enabled the identification of food types and their ingredients. The contents of food dishes are typically deformable objects, usually including complex semantics, which makes the task of defining their structure very difficult. Deep learning methods have already shown very promising results in such challenges. The development of a new deep learning model (architecture) applied to a new domain (problem - task) is a complex and time-consuming process that requires programming and mathematical skills beyond the average, along an understanding of datadriven (empirical) model learning. On the other hand, transfer learning with fine-tuning should be adopted when an immediate solution is required.

CONTENTS

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

The most recent food recognition systems are developed based on deep convolutional neural network architectures.Deep learning is one of the only methods by which we can circumvent the challenges of feature extraction. This is because deep learning models are capable of learning to focus on the right features by themselves, requiring little guidance from the programmer.

1.1 Background

Food image Recognition is one of the Promising application of object recognition. A convolutional neural network can be used which is suited for image recognition.

The effectiveness of Deep Convolutional Neural Network (DCNN) for recognizing the food which will solve the image classification problem. Machine Learning which involves using Complicated Models is called Deep Neural Network. It is a subset of Machine Learning in which Multilayered Neural Network learn from vast amount of data.

Transfer learning is carried out be feeding a Pre-trained model to a new network Transfer Learning is done to the network to be created . Transfer Learning is that to use knowledge, that a model has Learned from a task where a lot of labeled training data is available. A Pre-trained model which is trained with a large number of dataset in the network. This Transfer learning make it easier to create a network.

1.2 Motivation

Previous image-to-recipe programs were a bit more simple in their approach. "It was like having a photo of the food and then searching in a huge cookbook of pictures to match it up." Inverse cooking algorithm instead of retrieving a recipe directly from an image, proposes a pipeline with an intermediate step where the set of ingredients is first obtained. This allows the generation of the instructions not only taking into account the image, but also the ingredients.

1.3 Objective

A CNN is developed using already Pre-trained CNN Model , which is been trained on a dataset and a pretrained model's is Loaded into the network . Food Identification is a kind of Visual recognition which is relatively harder. The Goal is to Recognize the food using a neural network. The neural network is trained with a large number of dataset. The network learns to predict the image and give which class it belongs to. The network is trained with different sets of labels which could predict different class of food. The layers are created depending upon the requirements.The last few layers of the Pre-Trained models network is removed and few more layer is added to it and the dataset is trained accordingly. After Transfer Learning the datasets are trained and food identification is carried out.

1.4 Contribution

The web application that uses the picture of the food, to recognize multiple food items in the same meal. The system then uses image processing and computational intelligence for food item recognition.

CHAPTER 2
EXISTING SYSTEM

1. EXISTING SYSTEM :

   Traditionally, the image-to-recipe problem has been formulated as a retrieval task, where a recipe is retrieved from a fixed dataset based on the image similarity score in an embedding space.

   The performance of such systems highly depends on the dataset size and diversity, as well as on the quality of the learned embedding. Not surprisingly, these systems fail when a matching recipe for the image query does not exist in the static dataset.

2. PROPOSED SYSTEM :

   An alternative to overcome the dataset constraints of retrieval systems is to formulate the image-to-recipe problem as a conditional generation one. Therefore, in this paper, we present a system that generates a cooking recipe containing a title, ingredients and cooking instructions directly from an image.

3. THE INVERSE COOKING SYSTEM : Generating a recipe (title, ingredients and instructions) from an image is a challenging task, which requires a simultaneous understanding of the ingredients composing the dish as well as the transformations they went through, e.g. slicing, blending or mixing with other ingredients. Instead of obtaining the recipe from an image directly, we argue that a recipe generation pipeline would benefit from an intermediate step predicting the ingredients list. The sequence of instructions would then be generated

conditioned on both the image and its corresponding list of ingredients, where the interplay between image and ingredients could provide additional insights on how the latter were processed to produce the resulting dish.

CHAPTER 3
ENVIRONMENTAL STUDY

3.1 System Configuration

System configuration describe the hardware and software requirement of the system for development

3.1.1 Hardware Requirements

- Memory : 4 GB of RAM
- Processor : Intel Core i3 or equivalent CPU
- Speed : 2.4 GHz
- Proper Internet Connection

3.1.2 Software Requirements

- Operating system : Windows
- Front End : Python
- IDE Used : Jupyter notebook(open-source IDE)
- Libraries : torch,torchvision,Flask,pillow,matplotlib,tensorflow ,numpy

3.2 Software Specification

3.2.1 Python

Python is an interpreted, high-level and general-purpose programming language. Python's design philosophy emphasizes code readability with its

notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

3.2.2 Torch

Torch is an open-source machine learning library, a scientific computing framework, and a script language based on the Lua programming language.It provides a wide range of algorithms for deep learning.The core package of Torch is torch. It provides a flexible N-dimensional array or Tensor, which supports basic routines for indexing, slicing, transposing, type-casting, resizing, sharing storage and cloning. This object is used by most other packages and thus forms the core object of the library. The Tensor also supports mathematical operations like max, min, sum, statistical distributions like uniform, normal and multinomial, and BLAS operations like dot product, matrix-vector multiplication, matrix-matrix multiplication, matrix-vector product and matrix product.

3.2.3 Pytorch

PyTorch is a small part of a computer software which is based on Torch library. It is a Deep Learning framework introduced by Facebook. PyTorch is a Machine Learning Library for Python programming language which is used for applications such as Natural Language Processing.

3.2.4 Flask

Flask is a web application framework written in Python. Armin Ronacher, who leads an international group of Python enthusiasts named Pocco, develops it. Flask is based on Werkzeug WSGI toolkit and Jinja2 template engine. Both are Pocco projects.

3.2.5 Pillow

Python Pillow module is built on top of PIL (Python Image Library). It is the essential modules for image processing in Python. But it is not supported by Python 3. But, we can use this module with the Python 3.x versions as PIL. It supports the variability of images such as jpeg, png, bmp, gif, ppm, and tiff.

3.2.6 TensorFlow

TensorFlow is an open-source framework developed by Google researchers to use machine learning, deep learning, and other analytical and statistical work elements. Like similar platforms, it was designed to simplify the process of designing and implementing advanced analytic programs for users such as data scientists. And typical programmers and forecasters. TensorFlow handles data sets that are placed as graphical interfaces, and the edges connecting the nodes to the graph can represent vectors or multidimensional matrices, forming what is known as tensors.

3.2.7 matplotlib

Matplotlib is a low level graph plotting library in python that serves as a visualization utility. Matplotlib was created by John D. Hunter. Matplotlib is open source and we can use it freely. Matplotlib is mostly written in python, a few segments are written in C, Objective-C and Javascript for Platform compatibility.

Numpy

NumPy is a library for Python. It adds support for large matrices and multidimensional arrays, along with a large collection of high-level mathematical functions to operate on these arrays. NumPy in Python gives functionality comparable to MATLAB since they are both interpreted. They allow the user to write fast programs as long as most operations work on arrays or matrices instead of scalars.

Trained model divides the given video into specified number of frames.A frames contains attributes such as width, height and RGB color code. Dataset trained with Corrosion detection stored as array values. And Validation set and training set is also taken as array for training with lstm . We used numpy to fit these with the model

CHAPTER 4
SYSTEM DESIGN

4.1 Workflow

Inverse Cooking recipe generation model implemented by multiple encoders and decoders.

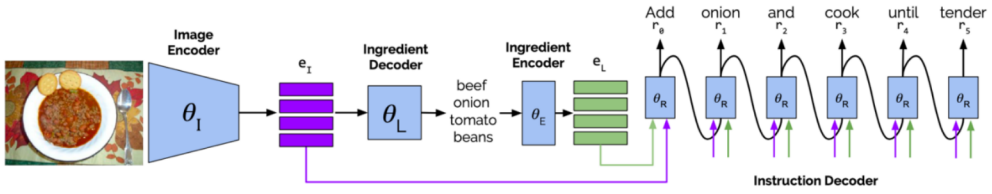


Fig. 4.1: Workflow Diagram

1. Image Encoder - We extract image features eI with the image encoder, parametrized by I
2. Ingredient Decoder - Ingredients are predicted by L
3. Ingredient Encoder - encoded into ingredient embeddings eL with e.
4. Cooking instruction decoder - parametrized by R generates a recipe title and a sequence of cooking steps by attending to image embeddings eI , ingredient embeddings eL, and previously predicted words (r0, ..., rt1).

The implementation of our project will have 4 stages:

1. Implementation of Encoders

2. Implementation of Decoders

3. Transfer learning and fine-tuning

4. Web Application Development using Python Flask

The Workflow diagram of the system is given in Figure 3.1

4.2 Methodology

4.2.1 Implementation of Encoders

Autoencoders are a type of neural network which generates an "n-layer" coding of the given input and attempts to reconstruct the input using the code generated. This Neural Network architecture is divided into the encoder structure, the decoder structure, and the latent space, also known as the "bottleneck". To learn the data representations of the input, the network is trained using Unsupervised data. These compressed, data representations go through a decoding process wherein which the input is reconstructed. An autoencoder is a regression task that models an identity function.

The encoder contains self-attention layers. In a self-attention layer all of the keys, values and queries come from the same place, in this case, the output of the previous layer in the encoder. Each position in the encoder can attend to all positions in the previous layer of the encoder.

Our recipe generation system takes a food image as an input and outputs a sequence of cooking instructions, which are generated by means of an instruction decoder that takes as input two embeddings. The first one represents visual features extracted from an image, while the second one encodes the ingredients extracted from the image.

4.2.2 Implementation of Decoders

Given an input image with associated ingredients, we aim to produce a sequence of instructions R = (r1, ..., rT) (where rt denotes a word in the sequence) by means of an instruction transformer.The title is predicted as

the first instruction. This transformer is conditioned jointly on two inputs: the image representation eI and the ingredient embedding eL.

We extract the image representation with a ResNet-50 encoder and obtain the ingredient embedding eL by means of a decoder architecture to predict ingredients, followed by a single embedding layer mapping each ingredient into a fixed-size vector. The instruction decoder is composed of transformer blocks, each of them containing two attention layers followed by a linear layer. The first attention layer applies self-attention over previously generated outputs, whereas the second one attends to the model conditioning in order to refine the self-attention output. The transformer model is composed of multiple transformer blocks followed by a linear layer and a softmax nonlinearity that provides a distribution over recipe words for each time step t.

4.2.3 Transfer learning and fine-tuning

The alternative path in deep learning-based food recognition approaches, relies on using popular pre-trained deep learning models, implementing the transfer learning technique. Transfer learning in deep learning theory is a popular machine learning technique, in which a model developed and trained for a specific task is reused for another similar task under some parametrization (fine tuning).

An intuitive definition of transfer learning was provided by Torrey and Shavlink in (2009): "Transfer learning is the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned". Fine tuning is the process in which a pre-trained model is used without its top layers, which are replaced by new layers, more appropriate to the new task (dataset).

4.3 Data Flow Diagram

A data flow diagram is a graphical representation of the "flow" of data through an information system, modeling its process aspects. A DFD is

often used as a preliminary step to create an overview of the system, which can later be elaborated. DFDs can also be used for the visualization of data processing (Structured Design). A DFD shows what kind of information will be input to and output from the system, where the data will come from and go to, and where the data will be stored. It does not show information about the timing of process or information about whether processes will operate in sequence or in parallel.

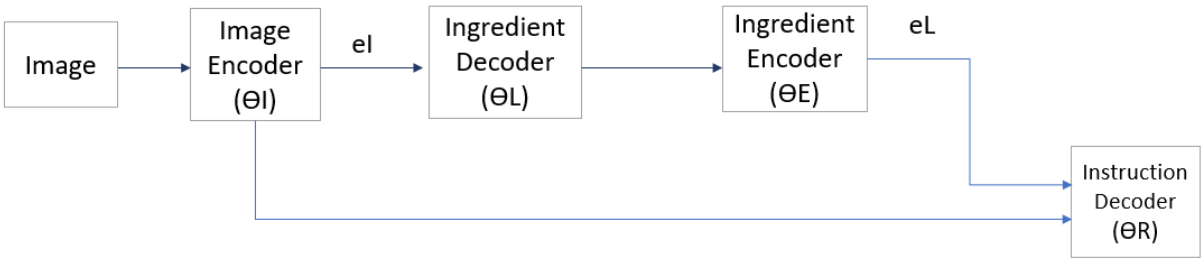


Fig. 4.2: Inverse Cooking System Architecture.

4.4 Web Application Development using Python Flask

Flask is a small and lightweight Python web framework that provides useful tools and features that make creating web applications in Python easier. It gives developers flexibility and is a more accessible framework for new developers since you can build a web application quickly using only a single Python file. Flask is also extensible and doesn't force a particular directory structure or require complicated boilerplate code before getting started.

The Bootstrap toolkit is used to style the application so it is more visually appealing. Bootstrap will help you incorporate responsive web pages in your web application so that it also works well on mobile browsers without writing your own HTML, CSS, and JavaScript code to achieve these goals.

Flask uses the Jinja template engine to dynamically build HTML pages using familiar Python concepts such as variables, loops, lists, and so on. You'll use these templates as part of this project.

4.5 Implementation

The images resized to 256 pixels in their shortest side and take random crops of 224×224 for training and we select central 224×224 pixels for evaluation. For the instruction decoder, we use a transformer with 16 blocks and 8 multi-head attentions, each one with dimensionality 64. For the ingredient decoder, we use a transformer with 4 blocks and 2 multi-head attentions, each one with dimensionality of 256. To obtain image embeddings we use the last convolutional layer of ResNet-50 model. Both image and ingredients embedings are of dimension 512. We keep a maximum of 20 ingredients per recipe and truncate instructions to a maximum of 150 words. The models are trained with Adam optimizer until early-stopping criteria is met (using patience of 50 and monitoring validation loss). All models are implemented with PyTorch.

4.6 ResNet-50

Deep residual networks like the popular ResNet-50 model is a convolutional neural network (CNN) that is 50 layers deep. A Residual Neural Network (ResNet) is an Artificial Neural Network (ANN) of a kind that stacks residual blocks on top of each other to form a network. You can load a pretrained version of the network trained on more than a million images from the ImageNet database. The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images. The network has an image input size of 224-by-224.

CHAPTER 5
RESULTS AND SCREENSHOTS

5.1 Source code screenshots

Encoder.py

encoder.py - Notepad

File Edit Format View Help

```
From torchvision.models import resnet18, resnet50, resnet101, resnet152, vgg16, vgg19, inception_v3
import torch
import torch.nn as nn
import random
import numpy as np


class EncoderCNN(nn.Module):
    def __init__(self, embed_size, dropout=0.5, image_model='resnet101', pretrained=True):
        """Load the pretrained ResNet-152 and replace top fc layer."""
        super(EncoderCNN, self).__init__()
        resnet = globals()[image_model](pretrained=pretrained)
        modules = list(resnet.children())[:-2]  # delete the last fc layer.
        self.resnet = nn.Sequential(*modules)

        self.linear = nn.Sequential(nn.Conv2d(resnet.fc.in_features, embed_size, kernel_size=1, padding=0),
                                    nn.Dropout2d(dropout))

    def forward(self, images, keep_cnn_gradients=False):
        """Extract feature vectors from input images."""

        if keep_cnn_gradients:
            raw_conv_feats = self.resnet(images)
        else:
            with torch.no_grad():
                raw_conv_feats = self.resnet(images)
        features = self.linear(raw_conv_feats)
        features = features.view(features.size(0), features.size(1), -1)

        return features


class EncoderLabels(nn.Module):
    def __init__(self, embed_size, num_classes, dropout=0.5, embed_weights=None, scale_grad=False):

        super(EncoderLabels, self).__init__()
        embeddinglayer = nn.Embedding(num_classes, embed_size, padding_idx=num_classes-1, scale_grad_by_freq=scale_grad)
        if embed_weights is not None:
            embeddinglayer.weight.data.copy_(embed_weights)
        self.pad_value = num_classes - 1
```

Fig. 5.1: Source code sample screenshots.1

```
        self.linear = embeddinglayer
        self.dropout = dropout
        self.embed_size = embed_size

    def forward(self, x, onehot_flag=False):

        if onehot_flag:
            embeddings = torch.matmul(x, self.linear.weight)
        else:
            embeddings = self.linear(x)

        embeddings = nn.functional.dropout(embeddings, p=self.dropout, training=self.training)
        embeddings = embeddings.permute(0, 2, 1).contiguous()

        return embeddings
```

Fig. 5.2: Source code sample screenshots.2

Decoder.py

transformer_decoder.py - Notepad

File Edit Format View Help

```
import math
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.nn.modules.utils import _single
import Foodimg2Ing.modules.utils as utils
from Foodimg2Ing.modules.multihead_attention import MultiheadAttention
import numpy as np
device = torch.device('cpu')
import copy


def make_positions(tensor, padding_idx, left_pad):
    """Replace non-padding symbols with their position numbers.
    Position numbers begin at padding_idx+1.
    Padding symbols are ignored, but it is necessary to specify whether padding
    is added on the left side (left_pad=True) or right side (left_pad=False).
    """

    # creates tensor from scratch - to avoid multigpu issues
    max_pos = padding_idx + 1 + tensor.size(1)
    #if not hasattr(make_positions, 'range_buf'):
    range_buf = tensor.new()
    #make_positions.range_buf = make_positions.range_buf.type_as(tensor)
    if range_buf.numel() < max_pos:
        torch.arange(padding_idx + 1, max_pos, out=range_buf)
    mask = tensor.ne(padding_idx)
    positions = range_buf[:tensor.size(1)].expand_as(tensor)
    if left_pad:
        positions = positions - mask.size(1) + mask.long().sum(dim=1).unsqueeze(1)

    out = tensor.clone()
    out = out.masked_scatter_(mask,positions[mask])
    return out
```

Fig. 5.3: Source code sample screenshots.3

```
class LearnedPositionalEmbedding(nn.Embedding):
    """This module learns positional embeddings up to a fixed maximum size.
    Padding symbols are ignored, but it is necessary to specify whether padding
    is added on the left side (left_pad=True) or right side (left_pad=False).
    """

    def __init__(self, num_embeddings, embedding_dim, padding_idx, left_pad):
        super().__init__(num_embeddings, embedding_dim, padding_idx)
        self.left_pad = left_pad
        nn.init.normal_(self.weight, mean=0, std=embedding_dim ** -0.5)

    def forward(self, input, incremental_state=None):
        """Input is expected to be of size [bsz x seqlen]."""
        if incremental_state is not None:
            # positions is the same for every token when decoding a single step

            positions = input.data.new(1, 1).fill_(self.padding_idx + input.size(1))
        else:

            positions = make_positions(input.data, self.padding_idx, self.left_pad)
        return super().forward(positions)

    def max_positions(self):
        """Maximum number of supported positions."""
        return self.num_embeddings - self.padding_idx - 1

class SinusoidalPositionalEmbedding(nn.Module):
    """This module produces sinusoidal positional embeddings of any length.
    Padding symbols are ignored, but it is necessary to specify whether padding
    is added on the left side (left_pad=True) or right side (left_pad=False).
    """

    def __init__(self, embedding_dim, padding_idx, left_pad, init_size=1024):
        super().__init__()
        self.embedding_dim = embedding_dim
        self.padding_idx = padding_idx
        self.left_pad = left_pad
        self.weights = SinusoidalPositionalEmbedding.get_embedding(
            init_size,
            embedding_dim,
```

Fig. 5.4: Source code sample screenshots.4

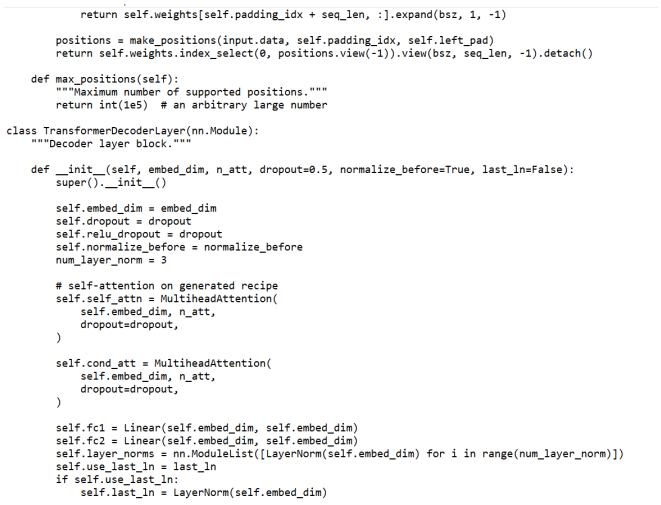
```
            init_size,
            embedding_dim,
            padding_idx,
        )
        self.register_buffer('_float_tensor', torch.FloatTensor())

    @staticmethod
    def get_embedding(num_embeddings, embedding_dim, padding_idx=None):
        """Build sinusoidal embeddings.
        This matches the implementation in tensor2tensor, but differs slightly
        from the description in Section 3.5 of "Attention Is All You Need".
        """
        half_dim = embedding_dim // 2
        emb = math.log(10000) / (half_dim - 1)
        emb = torch.exp(torch.arange(half_dim, dtype=torch.float) * -emb)
        emb = torch.arange(num_embeddings, dtype=torch.float).unsqueeze(1) * emb.unsqueeze(0)
        emb = torch.cat([torch.sin(emb), torch.cos(emb)], dim=1).view(num_embeddings, -1)
        if embedding_dim % 2 == 1:
            # zero pad
            emb = torch.cat([emb, torch.zeros(num_embeddings, 1)], dim=1)
        if padding_idx is not None:
            emb[padding_idx, :] = 0
        return emb

    def forward(self, input, incremental_state=None):
        """Input is expected to be of size [bsz x seqlen]."""
        # recompute/expand embeddings if needed
        bsz, seq_len = input.size()
        max_pos = self.padding_idx + 1 + seq_len
        if self.weights is None or max_pos > self.weights.size(0):
            self.weights = SinusoidalPositionalEmbedding.get_embedding(
                max_pos,
                self.embedding_dim,
                self.padding_idx,
            )
        self.weights = self.weights.type_as(self._float_tensor)

        if incremental_state is not None:
            # positions is the same for every token when decoding a single step
            return self.weights[self.padding_idx + seq_len, :].expand(bsz, 1, -1)
```

Fig. 5.5: Source code sample screenshots.5



```
            return self.weights[self.padding_idx + seq_len, :].expand(bsz, 1, -1)

        positions = make_positions(input.data, self.padding_idx, self.left_pad)
        return self.weights.index_select(0, positions.view(-1)).view(bsz, seq_len, -1).detach()

    def max_positions(self):
        """Maximum number of supported positions."""
        return int(1e5)  # an arbitrary large number

class TransformerDecoderLayer(nn.Module):
    """Decoder layer block."""

    def __init__(self, embed_dim, n_att, dropout=0.5, normalize_before=True, last_ln=False):
        super().__init__()

        self.embed_dim = embed_dim
        self.dropout = dropout
        self.relu_dropout = dropout
        self.normalize_before = normalize_before
        num_layer_norm = 3

        # self-attention on generated recipe
        self.self_attn = MultiheadAttention(
            self.embed_dim, n_att,
            dropout=dropout,
        )

        self.cond_att = MultiheadAttention(
            self.embed_dim, n_att,
            dropout=dropout,
        )

        self.fc1 = Linear(self.embed_dim, self.embed_dim)
        self.fc2 = Linear(self.embed_dim, self.embed_dim)
        self.layer_norms = nn.ModuleList([LayerNorm(self.embed_dim) for i in range(num_layer_norm)])
        self.use_last_ln = last_ln
        if self.use_last_ln:
            self.last_ln = LayerNorm(self.embed_dim)
```

Fig. 5.6: Source code sample screenshots.6

```
def forward(self, x, ingr_features, ingr_mask, incremental_state, img_features):

    # self attention
    residual = x
    x = self.maybe_layer_norm(0, x, before=True)
    x, _ = self.self_attn(
        query=x,
        key=x,
        value=x,
        mask_future_timesteps=True,
        incremental_state=incremental_state,
        need_weights=False,
    )
    x = F.dropout(x, p=self.dropout, training=self.training)
    x = residual + x
    x = self.maybe_layer_norm(0, x, after=True)

    residual = x
    x = self.maybe_layer_norm(1, x, before=True)

    # attention
    if ingr_features is None:

        x, _ = self.cond_att(query=x,
                             key=img_features,
                             value=img_features,
                             key_padding_mask=None,
                             incremental_state=incremental_state,
                             static_kv=True,
                             )
    elif img_features is None:
        x, _ = self.cond_att(query=x,
                             key=ingr_features,
                             value=ingr_features,
                             key_padding_mask=ingr_mask,
                             incremental_state=incremental_state,
                             static_kv=True,
                             )
```

Fig. 5.7: Source code sample screenshots.7

```
    else:
        # attention on concatenation of encoder_out and encoder_aux, query self attn (x)
        kv = torch.cat((img_features, ingr_features), 0)
        mask = torch.cat((torch.zeros(img_features.shape[1], img_features.shape[0], dtype=torch.uint8).to(device),
                          ingr_mask), 1)
        x, _ = self.cond_att(query=x,
                             key=kv,
                             value=kv,
                             key_padding_mask=mask,
                             incremental_state=incremental_state,
                             static_kv=True,
        )
    x = F.dropout(x, p=self.dropout, training=self.training)
    x = residual + x
    x = self.maybe_layer_norm(1, x, after=True)

    residual = x
    x = self.maybe_layer_norm(-1, x, before=True)
    x = F.relu(self.fc1(x))
    x = F.dropout(x, p=self.relu_dropout, training=self.training)
    x = self.fc2(x)
    x = F.dropout(x, p=self.dropout, training=self.training)
    x = residual + x
    x = self.maybe_layer_norm(-1, x, after=True)

    if self.use_last_ln:
        x = self.last_ln(x)

    return x

def maybe_layer_norm(self, i, x, before=False, after=False):
    assert before ^ after
    if after ^ self.normalize_before:
        return self.layer_norms[i](x)
    else:
        return x
```

Fig. 5.8: Source code sample screenshots.8

```
        # T x B x C -> B x T x C
        x = x.transpose(0, 1)

        x = self.linear(x)
        _, predicted = x.max(dim=-1)

        return x, predicted

    def sample(self, ingr_features, ingr_mask, greedy=True, temperature=1.0, beam=-1,
               img_features=None, first_token_value=0,
               replacement=True, last_token_value=0):

        incremental_state = {}

        # create dummy previous word
        if ingr_features is not None:
            fs = ingr_features.size(0)
        else:
            fs = img_features.size(0)

        if beam != -1:
            if fs == 1:
                return self.sample_beam(ingr_features, ingr_mask, beam, img_features, first_token_value,
                                        replacement, last_token_value)
            else:
                print ("Beam Search can only be used with batch size of 1. Running greedy or temperature sampling...")

        first_word = torch.ones(fs)*first_token_value

        first_word = first_word.to(device).long()
        sampled_ids = [first_word]
        logits = []

        for i in range(self.seq_length):
            # forward
            outputs, _ = self.forward(ingr_features, ingr_mask, torch.stack(sampled_ids, 1),
                                      img_features, incremental_state)
            outputs = outputs.squeeze(1)
            if not replacement:
```

Fig. 5.9: Source code sample screenshots.9

```
                if img_features is not None:
                    img_features = img_features.permute(0, 2, 1)
                    img_features = img_features.transpose(0, 1)
                    if self.normalize_inputs:
                        self.layer_norms_in[1](img_features)

                if ingr_mask is not None:
                    ingr_mask = (1-ingr_mask.squeeze(1)).byte()

                # embed positions
                if self.embed_positions is not None:
                    positions = self.embed_positions(captions, incremental_state=incremental_state)
                if incremental_state is not None:
                    if self.embed_positions is not None:
                        positions = positions[:, -1:]
                    captions = captions[:, -1:]

                # embed tokens and positions
                x = self.embed_scale * self.embed_tokens(captions)

                if self.embed_positions is not None:
                    x += positions

                if self.normalize_inputs:
                    x = self.layer_norms_in[2](x)

                x = F.dropout(x, p=self.dropout, training=self.training)

                # B x T x C -> T x B x C
                x = x.transpose(0, 1)

                for p, layer in enumerate(self.layers):
                    x  = layer(
                        x,
                        ingr_features,
                        ingr_mask,
                        incremental_state,
                        img_features
                    )
```

Fig. 5.10: Source code sample screenshots.10

```
        # T x B x C -> B x T x C
        x = x.transpose(0, 1)

        x = self.linear(x)
        _, predicted = x.max(dim=-1)

        return x, predicted

    def sample(self, ingr_features, ingr_mask, greedy=True, temperature=1.0, beam=-1,
               img_features=None, first_token_value=0,
               replacement=True, last_token_value=0):

        incremental_state = {}

        # create dummy previous word
        if ingr_features is not None:
            fs = ingr_features.size(0)
        else:
            fs = img_features.size(0)

        if beam != -1:
            if fs == 1:
                return self.sample_beam(ingr_features, ingr_mask, beam, img_features, first_token_value,
                                        replacement, last_token_value)
            else:
                print ("Beam Search can only be used with batch size of 1. Running greedy or temperature sampling...")

        first_word = torch.ones(fs)*first_token_value

        first_word = first_word.to(device).long()
        sampled_ids = [first_word]
        logits = []

        for i in range(self.seq_length):
            # forward
            outputs, _ = self.forward(ingr_features, ingr_mask, torch.stack(sampled_ids, 1),
                                      img_features, incremental_state)
            outputs = outputs.squeeze(1)
            if not replacement:
```

Fig. 5.11: Source code sample screenshots.11

```
                outputs = outputs.squeeze(1)
                if not replacement:
                    # predicted mask
                    if i == 0:
                        predicted_mask = torch.zeros(outputs.shape).float().to(device)
                    else:
                        # ensure no repetitions in sampling if replacement==False
                        batch_ind = [j for j in range(fs) if sampled_ids[i][j] != 0]
                        sampled_ids_new = sampled_ids[i][batch_ind]
                        predicted_mask[batch_ind, sampled_ids_new] = float('-inf')

                    # mask previously selected ids
                    outputs += predicted_mask

                logits.append(outputs)
                if greedy:
                    outputs_prob = torch.nn.functional.softmax(outputs, dim=-1)
                    _, predicted = outputs_prob.max(1)
                    predicted = predicted.detach()
                else:
                    k = 10
                    outputs_prob = torch.div(outputs.squeeze(1), temperature)
                    outputs_prob = torch.nn.functional.softmax(outputs_prob, dim=-1).data

                    # top k random sampling
                    prob_prev_topk, indices = torch.topk(outputs_prob, k=k, dim=1)
                    predicted = torch.multinomial(prob_prev_topk, 1).view(-1)
                    predicted = torch.index_select(indices, dim=1, index=predicted)[:, 0].detach()

                sampled_ids.append(predicted)

            sampled_ids = torch.stack(sampled_ids[1:], 1)
            logits = torch.stack(logits, 1)

            return sampled_ids, logits

        def sample_beam(self, ingr_features, ingr_mask, beam=3, img_features=None, first_token_value=0,
                        replacement=True, last_token_value=0):
            k = beam
            alpha = 0.0
```

Fig. 5.12: Source code sample screenshots.12

```
    def sample_beam(self, ingr_features, ingr_mask, beam=3, img_features=None, first_token_value=0,
                    replacement=True, last_token_value=0):
        k = beam
        alpha = 0.0
        # create dummy previous word
        if ingr_features is not None:
            fs = ingr_features.size(0)
        else:
            fs = img_features.size(0)
        first_word = torch.ones(fs)*first_token_value

        first_word = first_word.to(device).long()

        sequences = [[[first_word], 0, {}, False, 1]]
        finished = []

        for i in range(self.seq_length):
            # forward
            all_candidates = []
            for rem in range(len(sequences)):
                incremental = sequences[rem][2]
                outputs, _ = self.forward(ingr_features, ingr_mask, torch.stack(sequences[rem][0], 1),
                                          img_features, incremental)
                outputs = outputs.squeeze(1)
                if not replacement:
                    # predicted mask
                    if i == 0:
                        predicted_mask = torch.zeros(outputs.shape).float().to(device)
                    else:
                        # ensure no repetitions in sampling if replacement==False
                        batch_ind = [j for j in range(fs) if sequences[rem][0][i][j] != 0]
                        sampled_ids_new = sequences[rem][0][i][batch_ind]
                        predicted_mask[batch_ind, sampled_ids_new] = float('-inf')

                    # mask previously selected ids
                    outputs += predicted_mask

                outputs_prob = torch.nn.functional.log_softmax(outputs, dim=-1)
                probs, indices = torch.topk(outputs_prob, beam)
```

Fig. 5.13: Source code sample screenshots.13

```
                probs, indices = torch.topk(outputs_prob, beam)
                # tokens is [batch x beam ] and every element is a list
                # score is [ batch x beam ] and every element is a scalar
                # incremental is [batch x beam ] and every element is a dict


                for bid in range(beam):
                    tokens = sequences[rem][0] + [indices[:, bid]]
                    score = sequences[rem][1] + probs[:, bid].squeeze().item()
                    if indices[:,bid].item() == last_token_value:
                        finished.append([tokens, score, None, True, sequences[rem][-1] + 1])
                    else:
                        all_candidates.append([tokens, score, incremental, False, sequences[rem][-1] + 1])

            # if all the top-k scoring beams have finished, we can return them
            ordered_all = sorted(all_candidates + finished, key=lambda tup: tup[1]/(np.power(tup[-1],alpha)),
                                 reverse=True)[:k]
            if all(el[-1] == True for el in ordered_all):
                all_candidates = []

            # order all candidates by score
            ordered = sorted(all_candidates, key=lambda tup: tup[1]/(np.power(tup[-1],alpha)), reverse=True)
            # select k best
            sequences = ordered[:k]
            finished = sorted(finished,  key=lambda tup: tup[1]/(np.power(tup[-1],alpha)), reverse=True)[:k]

        if len(finished) != 0:
            sampled_ids = torch.stack(finished[0][0][1:], 1)
            logits = finished[0][1]
        else:
            sampled_ids = torch.stack(sequences[0][0][1:], 1)
            logits = sequences[0][1]
        return sampled_ids, logits

    def max_positions(self):
        """Maximum output length supported by the decoder."""
        return self.embed_positions.max_positions()

    def upgrade_state_dict(self, state_dict):
        if isinstance(self.embed_positions, SinusoidalPositionalEmbedding):
```

Fig. 5.14: Source code sample screenshots.14

```
    def upgrade_state_dict(self, state_dict):
        if isinstance(self.embed_positions, SinusoidalPositionalEmbedding):
            if 'decoder.embed_positions.weights' in state_dict:
                del state_dict['decoder.embed_positions.weights']
            if 'decoder.embed_positions._float_tensor' not in state_dict:
                state_dict['decoder.embed_positions._float_tensor'] = torch.FloatTensor()
        return state_dict


def Embedding(num_embeddings, embedding_dim, padding_idx, ):
    m = nn.Embedding(num_embeddings, embedding_dim, padding_idx=padding_idx)
    nn.init.normal_(m.weight, mean=0, std=embedding_dim ** -0.5)
    return m


def LayerNorm(embedding_dim):
    m = nn.LayerNorm(embedding_dim)
    return m


def Linear(in_features, out_features, bias=True):
    m = nn.Linear(in_features, out_features, bias)
    nn.init.xavier_uniform_(m.weight)
    nn.init.constant_(m.bias, 0.)
    return m


def PositionalEmbedding(num_embeddings, embedding_dim, padding_idx, left_pad, learned=False):
    if learned:
        m = LearnedPositionalEmbedding(num_embeddings, embedding_dim, padding_idx, left_pad)
        nn.init.normal_(m.weight, mean=0, std=embedding_dim ** -0.5)
        nn.init.constant_(m.weight[padding_idx], 0)
    else:
        m = SinusoidalPositionalEmbedding(embedding_dim, padding_idx, left_pad, num_embeddings)
    return m
```

Fig. 5.15: Source code sample screenshots.15

routes.py

routes.py - Notepad

File Edit Format View Help

```
from flask import render_template ,url_for,flash,redirect,request
from Foodimg2Ing import app
from Foodimg2Ing.output import output
import os


@app.route('/',methods=['GET'])
def home():
    return render_template('home.html')

@app.route('/about',methods=['GET'])
def about():
    return render_template('about.html')

@app.route('/',methods=['POST','GET'])
def predict():
    imagefile=request.files['imagefile']
    image_path=os.path.join(app.root_path,'static\\images\\demo_imgs',imagefile.filename)
    imagefile.save(image_path)
    img="/images/demo_imgs/"+imagefile.filename
    title,ingredients,recipe = output(image_path)
    return render_template('predict.html',title=title,ingredients=ingredients,recipe=recipe,img=img)

@app.route('/<samplefoodname>')
def predictsample(samplefoodname):
    imagefile=os.path.join(app.root_path,'static\\images',str(samplefoodname)+".jpg")
    img="/images/"+str(samplefoodname)+".jpg"
    title,ingredients,recipe = output(imagefile)
    return render_template('predict.html',title=title,ingredients=ingredients,recipe=recipe,img=img)
```

Fig. 5.16: Source code sample screenshots.16

output.py

```
# import the necessary libraries

import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import numpy as np
import os
from Foodimg2Ing.args import get_parser
import pickle
from Foodimg2Ing.model import get_model
from torchvision import transforms
from Foodimg2Ing.utils.output_utils import prepare_output
from PIL import Image
import time
from tensorflow.keras.preprocessing import image
from Foodimg2Ing import app


def output(uploadedfile):

    # Keep all the codes and pre-trained weights in data directory
    data_dir=os.path.join(app.root_path,'data')


    # code will run in gpu if available and if the flag is set to True, else it will run on cpu
    use_gpu = False
    device = torch.device('cpu')
    map_loc = device



    # code below was used to save vocab files so that they can be loaded without Vocabulary class
    ingrs_vocab = pickle.load(open(os.path.join(data_dir, 'ingr_vocab.pkl'), 'rb'))
    vocab = pickle.load(open(os.path.join(data_dir, 'instr_vocab.pkl'), 'rb'))

    ingr_vocab_size = len(ingrs_vocab)
    instrs_vocab_size = len(vocab)
    output_dim = instrs_vocab_size
```

Fig. 5.17: Source code sample screenshots.17

```
    t = time.time()
    import sys; sys.argv=['']; del sys
    args = get_parser()
    args.maxseqlen = 15
    args.ingrs_only=False
    model=get_model(args, ingr_vocab_size, instrs_vocab_size)

    # Load the pre-trained model parameters
    model_path = os.path.join(data_dir, 'modelbest.ckpt')
    model.load_state_dict(torch.load(model_path, map_location=map_loc))
    model.to(device)
    model.eval()
    model.ingrs_only = False
    model.recipe_only = False


    transf_list_batch = []
    transf_list_batch.append(transforms.ToTensor())
    transf_list_batch.append(transforms.Normalize((0.485, 0.456, 0.406),
                                                  (0.229, 0.224, 0.225)))
    to_input_transf = transforms.Compose(transf_list_batch)

    greedy = [True, False]
    beam = [-1, -1]
    temperature = 1.0
    numgens = len(greedy)

    uploaded_file=uploadedfile

    img=image.load_img(uploaded_file)

    show_anyways = False #if True, it will show the recipe even if it's not valid
    transf_list = []
    transf_list.append(transforms.Resize(256))
    transf_list.append(transforms.CenterCrop(224))
    transform = transforms.Compose(transf_list)
```

Fig. 5.18: Source code sample screenshots.18

```
    image_transf = transform(img)
    image_tensor = to_input_transf(image_transf).unsqueeze(0).to(device)

    num_valid = 1
    title=[]
    ingredients=[]
    recipe=[]
    for i in range(numgens):
        with torch.no_grad():
            outputs = model.sample(image_tensor, greedy=greedy[i],
                                   temperature=temperature, beam=beam[i], true_ingrs=None)

        ingr_ids = outputs['ingr_ids'].cpu().numpy()
        recipe_ids = outputs['recipe_ids'].cpu().numpy()

        outs, valid = prepare_output(recipe_ids[0], ingr_ids[0], ingrs_vocab, vocab)

        if valid['is_valid'] or show_anyways:

            title.append(outs['title'])

            ingredients.append(outs['ingrs'])

            recipe.append(outs['recipe'])


        else:
            title.append("Not a valid recipe!")
            recipe.append("Reason: "+valid['reason'])

    return title,ingredients,recipe
```

Fig. 5.19: Source code sample screenshots.19

run.py

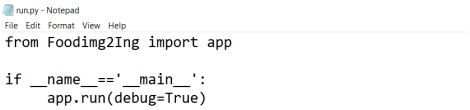


Fig. 5.20: Source code sample screenshots.20

Templates about.html

about.html - Notepad

File Edit Format View Help

```
{% extends "layout.html" %}
{% block content %}



        </div>
    </div>
</div>

<div id="section-about"onload="sampleFunction()">
    <div id="aboutcard" class="card col-md-8">
        <div class="card-header ">
          <b>ABOUT</b>
        </div>
        <div class="card-body">
          <p class="title">TITLE : Recipe Generation From Food Image</p>
          <p class="title">ABSTRACT :</p>
          <p class="aboutcontent">People enjoy food photography because they appreciate
            food. Behind each meal there is a story described in a com plex recipe and, unfortunately, by simply looking at a food
            image we do not have access to its preparation process.
            Therefore, in this paper we introduce an inverse cooking
            system that recreates cooking recipes given food images.
            Our system predicts ingredients as sets by means of a novel
            architecture, modeling their dependencies without impos ing any order, and then generates cooking instructions by
            attending to both image and its inferred ingredients simul taneously.
            </p>
        </div>
      </div>
</div>

{% endblock content %}
```

Fig. 5.21: Source code sample screenshots.21

header.html

header.html - Notepad

File Edit Format View Help

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0,shrink-to-fit=no">

    <!-- CSS -->
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css" integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous
    <link rel="stylesheet" href="{{ url_for('static',filename='css/style.css') }}">

    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.3/css/all.min.css"/>



    <title>COOKBOOK</title>
    <link rel="shortcut icon" href="{{ url_for('static', filename='images/cookbook.ico') }}">
</head>
<body>

    <!-- Navigation Bar -->
    <div id="Section-1">
        <nav class="navbar navbar-expand-lg navbar-light bg-grey">
            <a class="navbar-brand mr-auto" href="{{url_for('home')}}" style="font-family: 'Times New Roman', Times, serif;">
                <img src="{{ url_for('static',filename='images/cookbook.ico') }}" width="30" height="30" class="d-inline-block align-top" alt="" style="margin-top: -5.5px;">
                COOKBOOK</a>
            <a href="{{url_for('home')}}" class="w-100 text-center text-dark">RECIPE GENERATION FROM FOOD IMAGE</a>
            <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNavDropdown" aria-controls="navbarNavDropdown" aria-expanded="false" aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse " id="navbarNavDropdown">
            <ul class="navbar-nav ml-auto">
                <li class="nav-item active">
                <a class="nav-link" href="{{url_for('home')}}">Home <span class="sr-only">(current)</span></a>
                </li>
                <li class="nav-item active">
                <a class="nav-link" href="{{url_for('about')}}">About</a>
                </li>
            </ul>
            </div>
        </nav>
    </div>
```

Fig. 5.22: Source code sample screenshots.22

home.html

```
    {% block content %}{% endblock %}



    <!-- jQuery first, then Popper.js, then Bootstrap JS -->
    <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN" crossorigin="anonymous"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js" integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q" crossorigin="anonymous"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js" integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl" crossorigin="anonymous"></script>

</body>
<footer>
  <script src="{{ url_for('static', filename='main.js') }}" type="text/javascript"></script>
</footer>
</html>
```

Fig. 5.23: Source code sample screenshots.23

```
home.html - Notepad
File  Edit  Format  View  Help
{% extends "layout.html" %}
{% block home %}


            <div id="info" class="col-md-8">
                <div class="card m-3">
                    <img src="{{ url_for('static',filename='images/COOKBOOK.jpg')}}" class="card-img-top" alt="..." >
                    <div class="card-body" >
                      <h5 class="card-title text-center" id=infotitle >Recipe Generation from Food Image</h5>
                      <br>
                      <p class="card-text" id="infobody">Upload Food Image<br>Our Model will predict its Food Name , Ingredients and its Recipe.</p>
                      <p class="card-text"><small class="text-muted"></small></p>
                    </div>
                  </div>
            </div>


            <div id="loading" style="display:none;" class="col-md-8">
                <div class="loader">
                    <div class="inner one"></div>
                    <div class="inner two"></div>
                    <div class="inner three"></div>
                    <h4>Loading</h4>
                    <h5>Kindly wait for few seconds</h5>
                </div>
            </div>
        </div>
    </div>
</div>

{% endblock home %}
```

Fig. 5.24: Source code sample screenshots.24

layout.html

layout.html - Notepad

File Edit Format View Help

```
{% extends "header.html" %}
{% block content %}

<div id="section-2">
  <div class="container pt-5">
      <div class="row">
          <div class="col-md-4">
              <div class="wrapper">
              <div class="image">
                  {% if img %}
                  <img id="foodimage" onload="$(this).data('loaded', 'loaded');" src="{{ url_for('static',filename='images/')}}" alt="">
                  {% else %}
                  <img id="foodimage" onload="$(this).data('loaded', 'loaded');" src="" alt="">

                  {% endif %}


              </div>
              <div class="content">
                  <div class="icon text-center">
                    <i class="fas fa-cloud-upload-alt"></i>
                  </div>
                  <div class="text">
                      No file chosen, yet!
                  </div>
              </div>
              <div id="cancel-btn">
                <i class="fas fa-times"></i>
              </div>
              <div class="file-name">
                  File name here
              </div>
              </div>
              <button onclick="defaultBtnActive()" id="custom-btn">Upload a Food Image</button>
              <form id='foodimgform'action="/" method="POST" enctype="multipart/form-data">
                <input id="default-btn" type="file" hidden name="imagefile">
              </form>
              <button id="buttonsample" data-toggle="modal" data-target="#myModal">Choose from Sample Image</button>
              <div class="modal" id="myModal" tabindex="-1">
                <div class="modal-dialog">
                  <div class="modal-content">
                    <div class="modal-header">
```

Fig. 5.25: Source code sample screenshots.25

```
            <div class="modal-dialog">
              <div class="modal-content">
                <div class="modal-header">
                  <h5 class="modal-title">Sample Food Images</h5>
                  <button id="close" type="button" class="close" data-dismiss="modal" aria-hidden="true">
                    <i class="fas fa-times"></i>
                </button>
                </div>
                <div class="modal-body">
                    <div class="container">
                        <div class="row">
                          <div class="col">
                            <div class="card" style="width: 18rem;">
                                <img src="{{ url_for('static',filename='images/burger.jpg') }}" class="card-img-top" alt="...">
                                <div class="card-body">
                                  <h5 class="card-title">img1</h5>
                                  <a href="burger" class="btn btn-primary" onclick="select('burger.jpg')">Select</a>
                                </div>
                            </div>
                          </div>
                          <div class="col">
                            <div class="card" style="width: 18rem;">
                                <img src="{{ url_for('static',filename='images/chocolate-cake.jpg') }}" class="card-img-top" alt="..." >
                                <div class="card-body">
                                  <h5 class="card-title">img2</h5>
                                  <a href="chocolate-cake" class="btn btn-primary" onclick="select('chocolate-cake.jpg')" >Select</a>
                                </div>
                            </div>
                          </div>
                          <div class="col">
                            <div class="card" style="width: 18rem;">
                                <img src="{{ url_for('static',filename='images/grilled-beef.jpg') }}" class="card-img-top" alt="...">
                                <div class="card-body">
                                  <h5 class="card-title">img3</h5>
                                  <a href="grilled-beef" class="btn btn-primary" onclick="select('grilled-beef.jpg')">Select</a>
                                </div>
                            </div>
                          </div>
                        </div>
                    </div>
                </div>
                <div class="modal-footer">
```

Fig. 5.26: Source code sample screenshots.26

predict.html

predict.html - Notepad

File Edit Format View Help

```
{% extends "layout.html" %}
{% block home %}


<div id="info"  class="col-md-8">
    <div class="card text-center pb-2">
        <div class="card-header">
          <ul class="nav nav-tabs card-header-tabs">
            <li class="nav-item">
              <a class="nav-link active " id="tabbtn1" href="#" onclick="javascript:myFunctab1()">Recipe 1</a>
            </li>
            <li class="nav-item">
              <a class="nav-link " id="tabbtn2" href="#" onclick="javascript:myFunctab2()">Recipe 2</a>
            </li>
          </ul>
        </div>
        <div id="tab1" >
          <div class="card-body" align="left" id="result">
            {% if title[0] != "Not a valid recipe!" %}
              <h5 class="card-title" align='center'>{{title[0]}}</h5>
              <br>
              <h6>Ingredients :</h6>
              <p class="card-text">{{', '.join(ingredients[0])}}</p>
              <br>
              <h6>Recipe :</h6>
              <ol class="pl-3">
              {% for i in recipe[0] %}
              <li class="card-text">{{i}}</li>
              {% endfor %}
              </ol>

            {% else %}
            <h5 class="card-title" align='center'>{{title[0]}}</h5>
            <br>
            <h6 class="card-text">{{recipe[0]}}</h6>
            {% endif %}
            <br>
            <p class="card-text"><small class="text-muted">Check both the Recipes</small></p>
          </div>
        </div>
```

Fig. 5.27: Source code sample screenshots.27

```
                <div  id="tab2" style="display:None">
                  <div class="card-body" align="left" id="result">
                    {% if title[1] != "Not a valid recipe!" %}
                      <h5 class="card-title" align='center'>{{title[1]}}</h5>
                      <br>
                      <h6>Ingredients :</h6>
                      <p class="card-text">{{', '.join(ingredients[1])}}</p>
                      <br>
                      <h6>Recipe :</h6>
                      <ol class="pl-3">
                      {% for i in recipe[1] %}
                      <li class="card-text">{{i}}</li>
                      {% endfor %}
                      </ol>

                     {% else %}
                     <h5 class="card-title" align='center'>{{title[1]}}</h5>
                     <br>
                     <h6 class="card-text">{{recipe[1]}}</h6>
                     {% endif %}
                    </div>
                  </div>

                </div>
            </div>

            <div id="loading" style="display:none;" class="col-md-8">
              <div class="loader">
                   <div class="inner one"></div>
                   <div class="inner two"></div>
                   <div class="inner three"></div>
                   <h4>Loading</h4>
                   <h5>Kindly wait for few seconds</h5>
              </div>
            </div>

        </div>
    </div>
</div>

{% endblock home %}
```

Fig. 5.28: Source code sample screenshots.28

style.css

Fig. 5.29: Source code sample screenshots.29

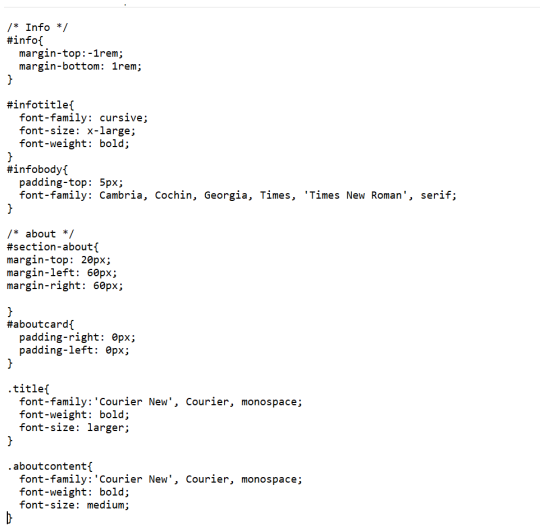
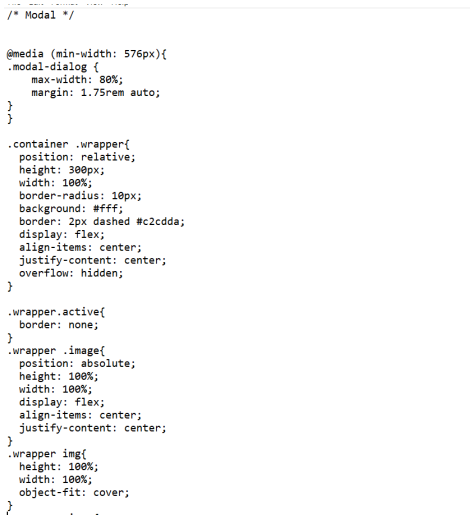


Fig. 5.30: Source code sample screenshots.30



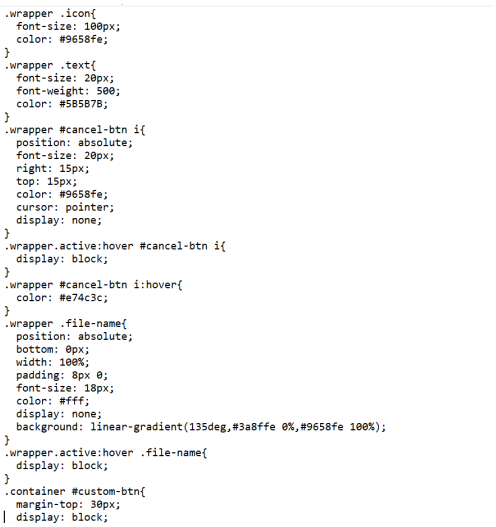
```
/* Modal */

@media (min-width: 576px){
.modal-dialog {
    max-width: 80%;
    margin: 1.75rem auto;
}
}

.container .wrapper{
  position: relative;
  height: 300px;
  width: 100%;
  border-radius: 10px;
  background: #fff;
  border: 2px dashed #c2cdda;
  display: flex;
  align-items: center;
  justify-content: center;
  overflow: hidden;
}

.wrapper.active{
  border: none;
}
.wrapper .image{
  position: absolute;
  height: 100%;
  width: 100%;
  display: flex;
  align-items: center;
  justify-content: center;
}
.wrapper img{
  height: 100%;
  width: 100%;
  object-fit: cover;
}
```

Fig. 5.31: Source code sample screenshots.31



```
.wrapper .icon{
  font-size: 100px;
  color: #9658fe;
}
.wrapper .text{
  font-size: 20px;
  font-weight: 500;
  color: #5B5B7B;
}
.wrapper #cancel-btn i{
  position: absolute;
  font-size: 20px;
  right: 15px;
  top: 15px;
  color: #9658fe;
  cursor: pointer;
  display: none;
}
.wrapper.active:hover #cancel-btn i{
  display: block;
}
.wrapper #cancel-btn i:hover{
  color: #e74c3c;
}
.wrapper .file-name{
  position: absolute;
  bottom: 0px;
  width: 100%;
  padding: 8px 0;
  font-size: 18px;
  color: #fff;
  display: none;
  background: linear-gradient(135deg,#3a8ffe 0%,#9658fe 100%);
}
.wrapper.active:hover .file-name{
  display: block;
}
.container #custom-btn{
  margin-top: 30px;
  display: block;
```

Fig. 5.32: Source code sample screenshots.32

```
.wrapper.active:hover .file-name{
  display: block;
}
.container #custom-btn{
  margin-top: 30px;
  display: block;
  width: 100%;
  height: 50px;
  border: none;
  outline: none;
  border-radius: 25px;
  color: #fff;
  font-size: 18px;
  font-weight: 500;
  letter-spacing: 1px;
  text-transform: uppercase;
  cursor: pointer;
  background: linear-gradient(135deg,#3a8ffe 0%,#9658fe 100%);
}

/* Animation Loading */
.loader h4{
  padding-left: 80px;
  padding-top: 20px;
}
.loader h5{
  width: 400px;
  padding-top: 37px;
  margin-left: -30px;
}

.loader {
  position: absolute;
  top: calc(35% - 32px);
  left: calc(50% - 32px);
  width: 64px;
  height: 64px;
  border-radius: 50%;
  perspective: 800px;
}
```

Fig. 5.33: Source code sample screenshots.33

```
/* Animation Loading */
.loader h4{
  padding-left: 80px;
  padding-top: 20px;
}
.loader h5{
  width: 400px;
  padding-top: 37px;
  margin-left: -30px;
}

.loader {
  position: absolute;
  top: calc(35% - 32px);
  left: calc(50% - 32px);
  width: 64px;
  height: 64px;
  border-radius: 50%;
  perspective: 800px;
}

.inner {
  position: absolute;
  box-sizing: border-box;
  width: 100%;
  height: 100%;
  border-radius: 50%;
}

.inner.one {
  left: 0%;
  top: 0%;
  animation: rotate-one 1s linear infinite;
  border-bottom: 3px solid black;
}

.inner.two {
  right: 0%;
  top: 0%;
  animation: rotate-two 1s linear infinite;
  border-right: 3px solid black;
```

Fig. 5.34: Source code sample screenshots.34

```
.inner.two {
  right: 0%;
  top: 0%;
  animation: rotate-two 1s linear infinite;
  border-right: 3px solid black;
}

.inner.three {
  right: 0%;
  bottom: 0%;
  animation: rotate-three 1s linear infinite;
  border-top: 3px solid black;
}

@keyframes rotate-one {
  0% {
    transform: rotateX(35deg) rotateY(-45deg) rotateZ(0deg);
  }
  100% {
    transform: rotateX(35deg) rotateY(-45deg) rotateZ(360deg);
  }
}

@keyframes rotate-two {
  0% {
    transform: rotateX(50deg) rotateY(10deg) rotateZ(0deg);
  }
  100% {
    transform: rotateX(50deg) rotateY(10deg) rotateZ(360deg);
  }
}

@keyframes rotate-three {
  0% {
    transform: rotateX(35deg) rotateY(55deg) rotateZ(0deg);
  }
  100% {
    transform: rotateX(35deg) rotateY(55deg) rotateZ(360deg);
  }
}
```

Fig. 5.35: Source code sample screenshots.35

5.2 Results

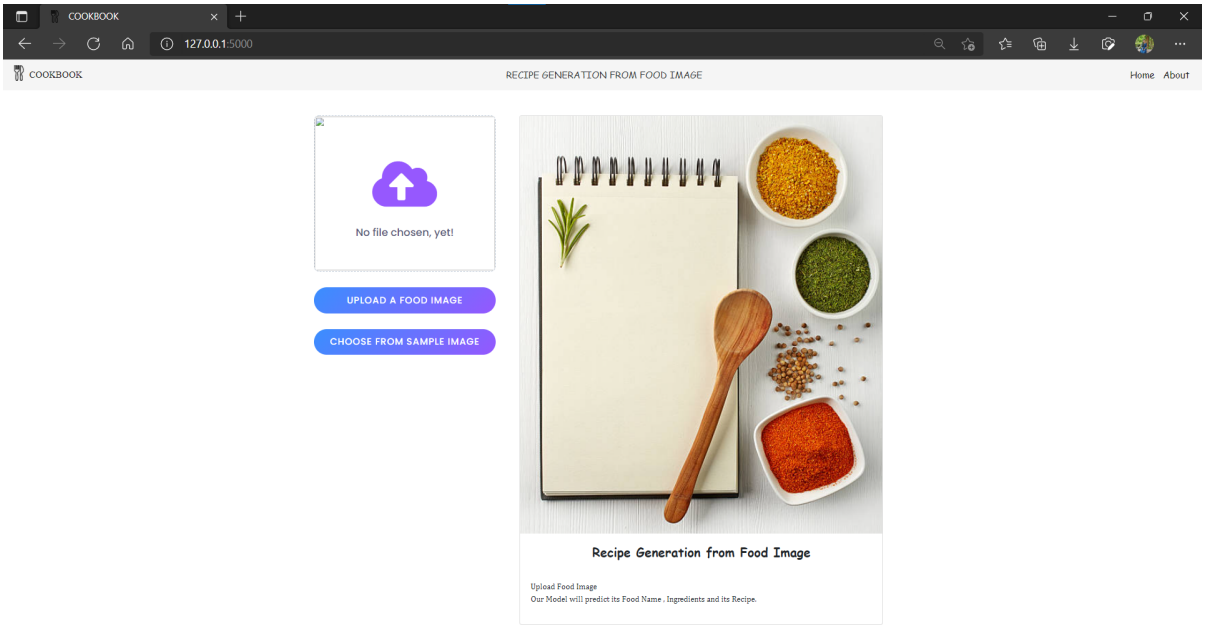


Fig. 5.36: User Interface web application

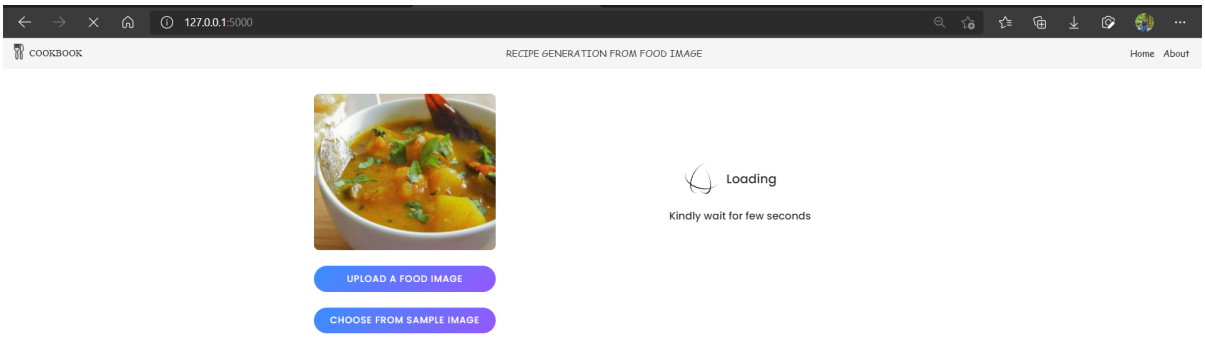


Fig. 5.37: Input 1

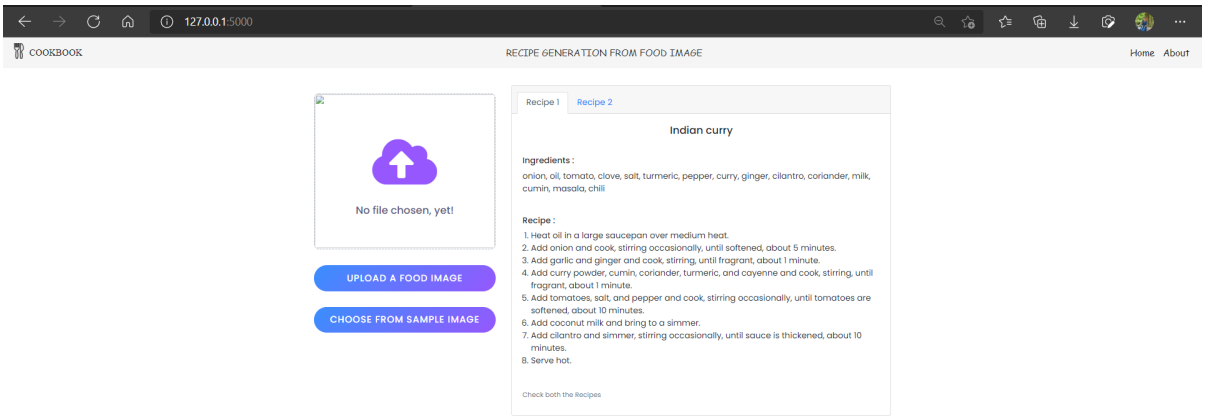


Fig. 5.38: Output 1

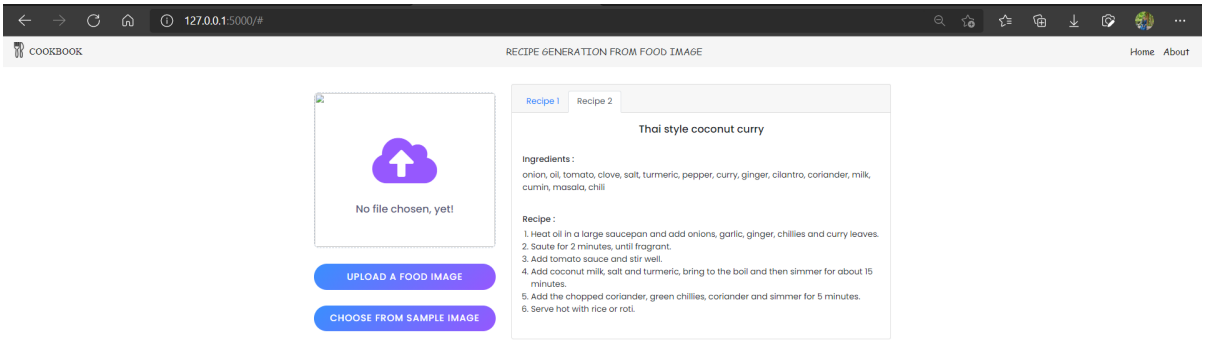


Fig. 5.39: Output 2

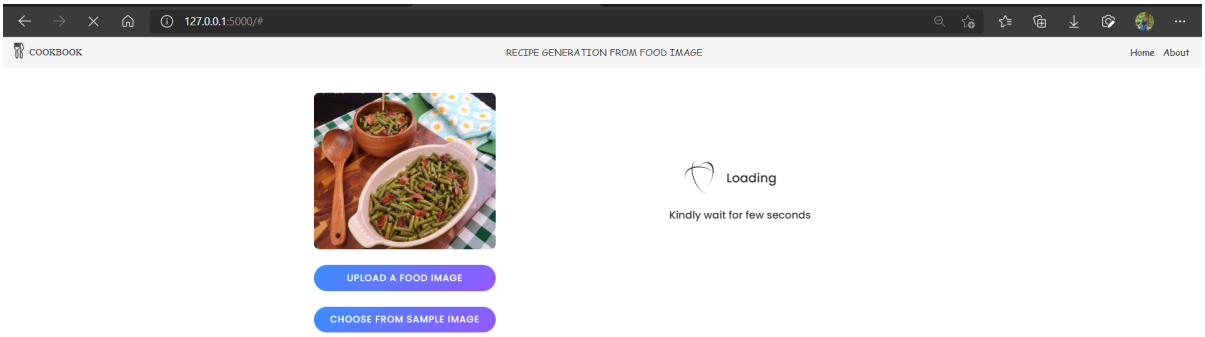


Fig. 5.40: Input 2

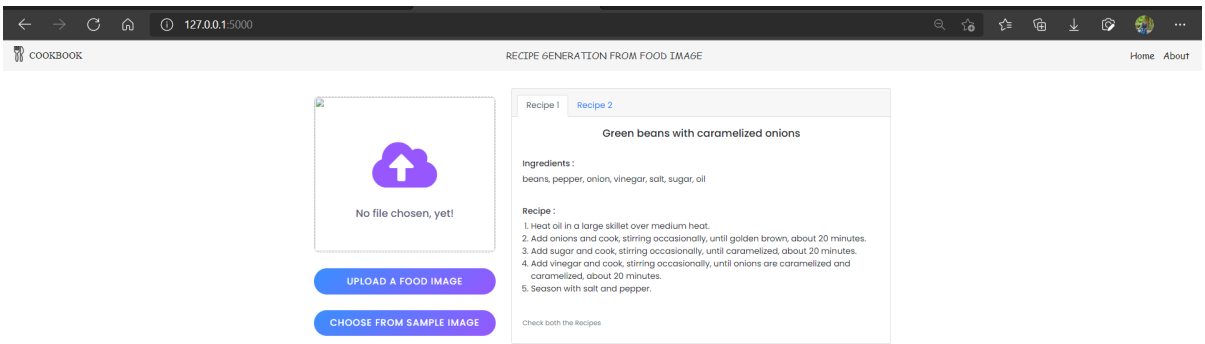


Fig. 5.41: Output 1

CHAPTER 6
CONCLUSION

Food is fundamental to human existence. Not only does it provide us with energy—it also defines our identity and culture. As the old saying goes, we are what we eat, and food related activities such as cooking, eating and talking about it take a significant portion of our daily life.

Food culture has been spreading more than ever in the current digital era, with many people sharing pictures of food they are eating across social media. Querying Instagram for food leads to at least 300M posts; similarly, searching for foodie results in at least 100M posts, highlighting the unquestionable value that food has in our society. Moreover, eating patterns and cooking culture have been evolving over time.

In the past, food was mostly prepared at home, but nowadays we frequently consume food prepared by thirdparties (e.g. takeaways, catering and restaurants). Thus, the access to detailed information about prepared food is limited and, as a consequence, it is hard to know precisely what we eat. Therefore, we argue that there is a need for inverse cooking systems, which are able to infer ingredients and cooking instructions from a prepared meal.

People enjoy food photography because they appreciate food. Behind each meal there is a story described in a complex recipe and, unfortunately, by simply looking at a food image we do not have access to its preparation process. The Food Scanner Application which takes a food image as input and produces a recipe consisting of a title, ingredients and sequence of cooking instructions.

CHAPTER 7
REFERENCES

[1] Recipe1M+: A Dataset for Learning Cross-Modal Embeddings for Cooking Recipes and Food Images – MIT

[2] Food Detection with Image Processing Using Convolutional Neural Network (CNN) Method — IEEE Conference Publication — IEEE Xplore

[3] (PDF) Food Image Recognition by Using Convolutional Neural Networks (CNNs) (researchgate.net)

[4] implified-recipes-1M Dataset - Dominik Schmidt Slide 1 (nvidia.com)

[5] [PDF] Deep Learning Based Food Recognition — Semantic Scholar

[6] Food Ingredients and Recipes Dataset with Images — Kaggle

[7] https://www.tutorialspoint.com/flask/index.htm

[8] https://flask.palletsprojects.com/en/2.0.x/tutorial/

[9] https://www.javatpoint.com/deep-learning

[10] https://www.tensorflow.org/tutorials/images/cnn

[11] https://papers.nips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[12] https://pytorch.org/