PICTURE TO RECIPE AND ITS NUTRITIONAL INFORMATION USING CNN AND K-NN

A thesis (Phase-I) submitted in partial fulfillment of the requirements for the award of the degree of

M.Tech

in

Data Analytics

By

SARVASUDDI BALAGANESH (205219022)

DEPARTMENT OF COMPUTER APPLICATIONS
NATIONAL INSTITUTE OF TECHNOLOGY
TIRUCHIRAPPALLI - 620015

DECEMBER 2020

BONA FIDE CERTIFICATE

This is to certify that the project work (phase I) titled **"Picture to Recipe and It's Nutritional Information Using CNN and K-NN"** is a bonafide record of the work done by

SARVASUDDI BALAGANESH (205219022)

in partial fulfillment of the requirements for the award of the degree of **Master of Technology** in **Data Analytics** of the **National Institute of Technology, Tiruchirappalli,** during the year 2020-2021.

[illegible]
16/12/2020

Dr. R. Eswari
Project Guide

[illegible]

Dr. P.J.A Alphonse
Head of the Department

Project Viva-voce held on 17-12-2020 .

[illegible]
17/12/2020

Internal Examiner

External Examiner

ABSTRACT

In everyday lives', food plays an important role and dictates one's health in many ways. Proper diet or intake of food with knowing its nutrition improves the health condition. Usually, people enjoy the food, and some people appreciate the food by clicking the food photo. There is a complex set of ingredients and instructions for preparation(recipe) behind every dish, and each ingredient has its Nutritional Information. Still, unfortunately, people do not have any access or information about the recipe and its Nutritional Information by simply looking at the dish's photo. Pretrained CNN (Convolutional Neural Networks) Models used in this project classify and provide the encodings of each image. AlexNet, DenseNet-161, GoogleNet, ResNet-152, ResNext101_32x8d, VGG-19 are the models used, which are trained on ImageNet data set. K-NN (k-Nearest Neighbours) then used on obtained encodings to provide the required recipe along with Nutritional details from the Recipe 1M data. It results that Resent-152 and ResNext101_32x8d performed similarly in terms of loss if distance as metric taken with 0.7635 and 0.7643 loss, respectively. Whereas DenseNet 161 performed with less loss 7.96E-07 and cosine similarity with the highest loss as ~1. Resent-152 and ResNext101_32x8d also reached nearer to Densenet 161 with 0.9973 and 0.9958, respectively. ResNext101_32x8d has retrieved more zero loss images than Densenet 161. Hence, it is recommended to use the ResNext101_32x8d model for encoding and Cosine Similarity as a metric for KNN.

ACKNOWLEDGEMENT

I would like to extend my gratitude to **Dr. Mini Shaji Thomas, Director**, National Institute of Technology, Tiruchirappalli for having provided all the facilities to carry out this project work.

I would like to thank **Dr. P.J.A Alphonse, Head of the Department**, Department of Computer Applications, National Institute of Technology, Tiruchirappalli for allowing me to avail the facilities of the department during project work.

I express my sincere thanks and gratitude to my guide **Dr. R.Eswari, Assistant Professor**, Department of Computer Applications, National Institute of Technology, Tiruchirappalli who directed me from time to time throughout the project with valuable guidance, constant encouragement and suggestions.

My hearty thanks to all the members of the Project Coordination Committee, Department of Computer Applications, National Institute of Technology, Tiruchirappalli, for their valuable suggestions during the project reviews.

I am also thankful to the faculty, staff members and research scholars of the Department of Computer Applications, National Institute of Technology, Tiruchirappalli, for their support and suggestions throughout the project.

Sarvasuddi Balaganesh

(205219022)

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CNN	Convolutional Neural Networks
K-NN	k-Nearest Neighbors
3-NN	3-Nearest Neighbors
FC	Fully Connected
ReLU	Rectified linear unit
AWS	Amazon Web Services
ResNet	Residual neural network
DenseNet	Densely Connected Convolutional Networks
VGG	Visual Geometry Group at University of Oxford.
t-SNE	t-distributed stochastic neighbour embedding
BBQ	Barbecue
URL	Uniform Resource Locator
t-SNE	t-distributed stochastic neighbour embedding

CHAPTER 1
INTRODUCTION

1.1 SCOPE

Food is fundamental to human existence. Not only does it provide us with energy, but it also defines our identity and culture. There is a saying that '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 un-questionable value that food has in our society.

Moreover, eating patterns and cooking culture have been evolving. In the past, food was mostly prepared at home, but people frequently consume food prepared by third parties (e.g., takeaways, catering, bakeries, and restaurants). Thus, access to detailed information about prepared food is limited, and, therefore, it is hard to know precisely what people generally eat. Consequently, it is necessary to argue that there is a need for a specific cooking system, which can infer ingredients, cooking instructions, and nutritional information from a prepared meal.

1.2 TECHNIQUES

Studies have shown that learning transfer increases the model's overall performance when there are minimal marked training data. Transfer Learning is a technique that trains a large dataset with a neural architecture and uses the trained parameters to initialize the target model's weights. This Research begins with experimenting with the traditional machine learning algorithm K-NN which works with both distance and similarity. The next step is to create embeddings for each image from the pre-trained CNN models for the domain-specific terms and used them for the K-NN Algorithm.

1.2.1 Image Encoding

To build this system, it is necessary to determine how similar two images matched an input image to the most similar image in our dataset.

Direct pixel value comparisons are an unsatisfactory method of evaluating similarity, mainly because it is concerned with the similarity in image content, not colors and object positions. It is known that later layers in a CNN could be used as an image encoding.

Since each activation in later layers corresponds to some higher-level or abstract feature of the image. And it is known that these encodings capture the content of pictures while ignoring pixel-value differences caused by lighting, object position, discoloration, etc. To produce image encodings for every image in our dataset, these encodings needed to run on every image in the training data through various convolutional neural networks, which were pre-trained on the ImageNet dataset.

Available CNN models which are pre-trained on ImageNet dataset that are given in the official Pytorch website are:

- AlexNet
- VGG
- ResNet
- SqueezeNet
- DenseNet
- Inception v3
- GoogLeNet
- ShuffleNet v2
- MobileNet v2
- ResNeXt
- Wide ResNet
- MNASNet

Only few CNN models from the above list tha used in this Research to achieve the task of image encoding.

- **AlexNet**

AlexNet, which employed an 8-layer CNN, won the ImageNet Large Scale Visual Recognition Challenge 2012 by a considerable margin. For the first time, this network showed that learning features can transcend manually designed features, breaking the previous paradigm in computer vision.

The design philosophies of AlexNet and LeNet are very similar, but there are also significant differences. First, AlexNet is much deeper than the comparatively small LeNet5. AlexNet consists of eight layers: five convolutional layers, two fully connected hidden layers, and one fully connected output layer. Second, AlexNet used the ReLU instead of the sigmoid as its activation function.

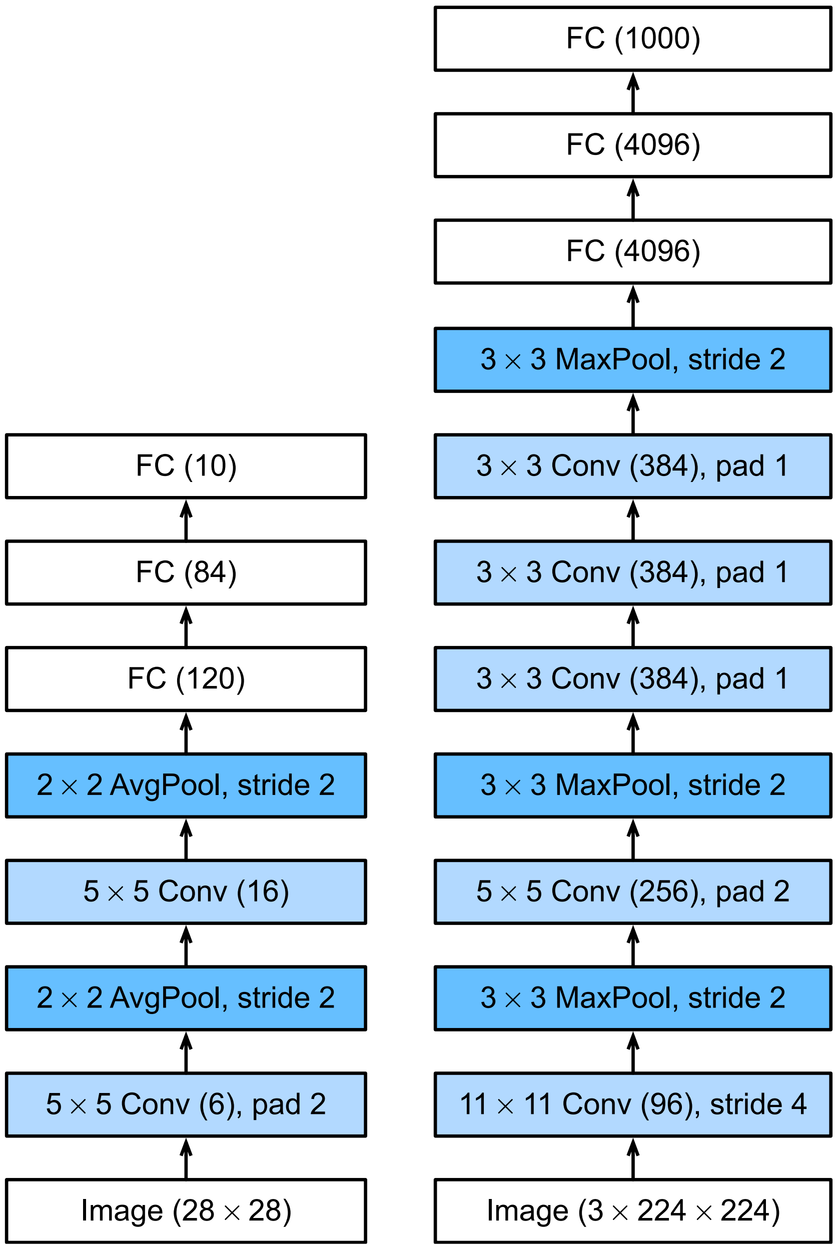


Figure 1.1: From LeNet (left) to AlexNet (right)

- **VGG**

  Like AlexNet and LeNet, the VGG Network can be partitioned into two parts: the first consisting mostly of convolutional and pooling layers, and the second consisting of fully connected layers.

The convolutional part of the network connects several VGG blocks from Figure. 1.2 (also defined in the vgg_block function) in succession. The following variable convarch consists of a list of tuples (one per block). Each contains two values: the number of convolutional layers and the number of output channels, precisely the arguments required to call the vgg_block function. The fully connected part of the VGG network is identical to that covered in AlexNet.

```
conv_arch = ((1, 64), (1, 128), (2, 256), (2, 512), (2, 512))
```

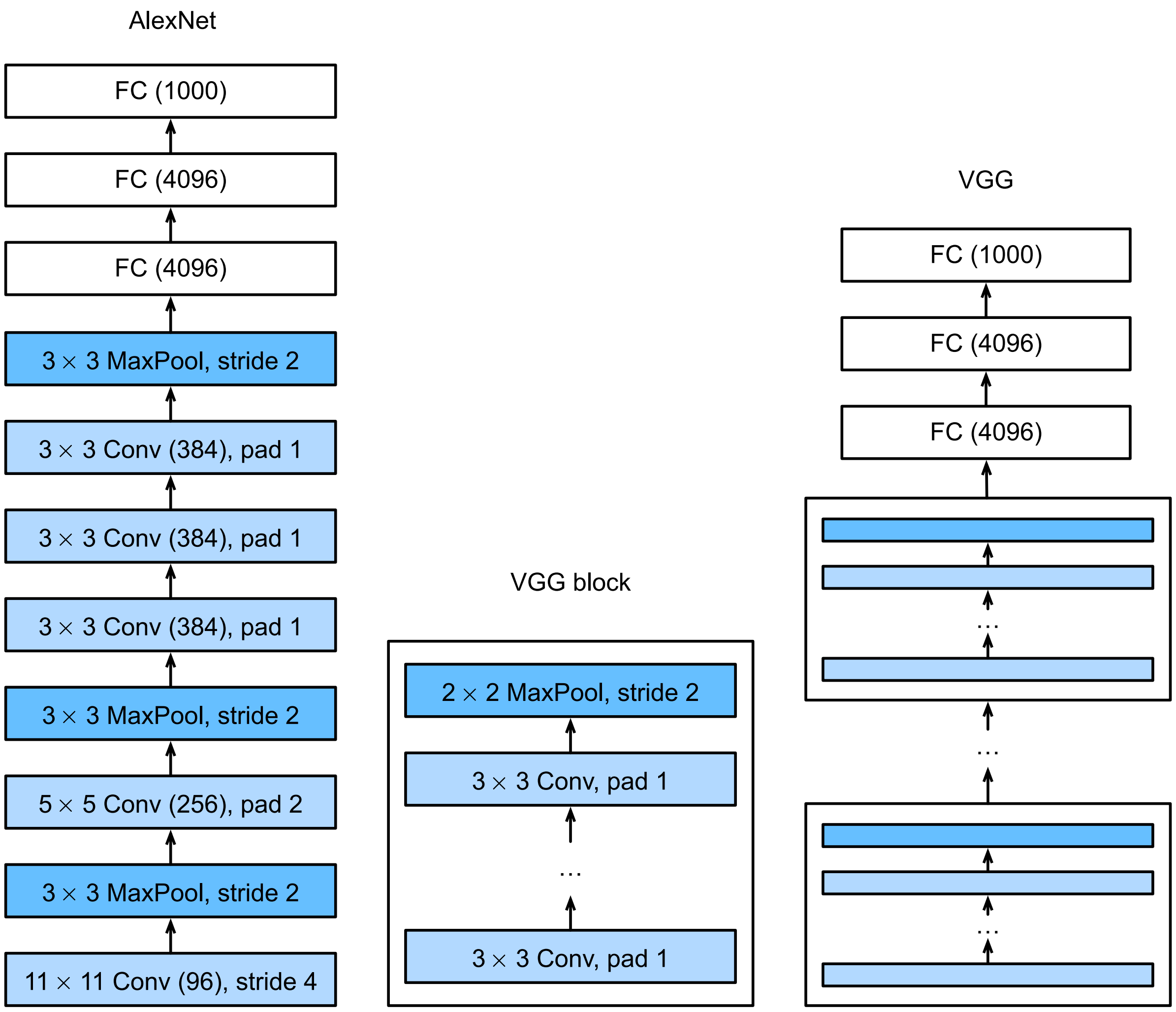


Figure 1.2: From AlexNet to VGG that is designed from building blocks.

- **Google Net**

  The basic convolutional block in GoogLeNet is called an Inception block, likely named due to a quote from the movie Inception ("We need to go deeper"), which launched a viral meme.

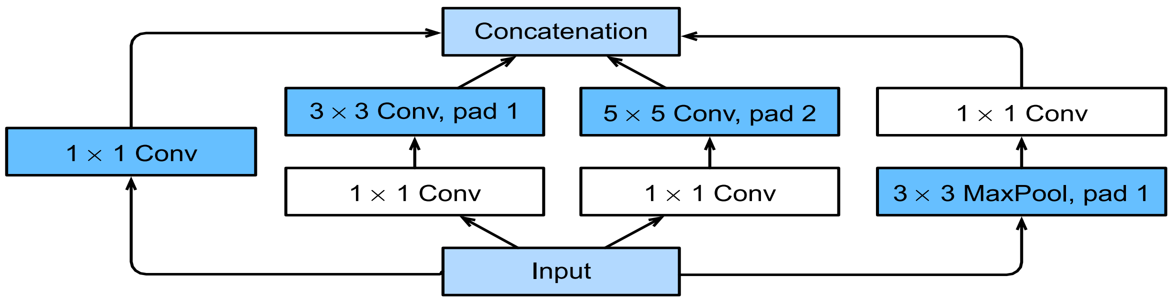


Figure 1.3: Structure of the Inception block.

As depicted in Figure. 1.3, the inception block consists of four parallel paths. The first three paths use convolutional layers with window sizes of 1×1, 3×3, and 5×5 to extract information from different spatial sizes. The middle two paths perform a 1×1 convolution on the input to reduce the number of channels, reducing the model's complexity. The fourth path uses a 3×3 maximum pooling layer, followed by a 1×1 convolutional layer to change the number of channels. The four paths all use appropriate padding to give the input and output the same height and width. Finally, the outputs along each path are concatenated along the channel dimension and comprise the block's output. The commonly tuned hyperparameters of the Inception block are the number of output channels per layer.

As shown in Figure. 1.4, GoogLeNet uses a stack of a total of 9 inception blocks and global average pooling to generate its estimates. Maximum pooling between inception blocks reduces the dimensionality. The first module is like AlexNet and LeNet. The stack of blocks is inherited from VGG and the global average pooling avoids a stack of fully connected layers at the end.

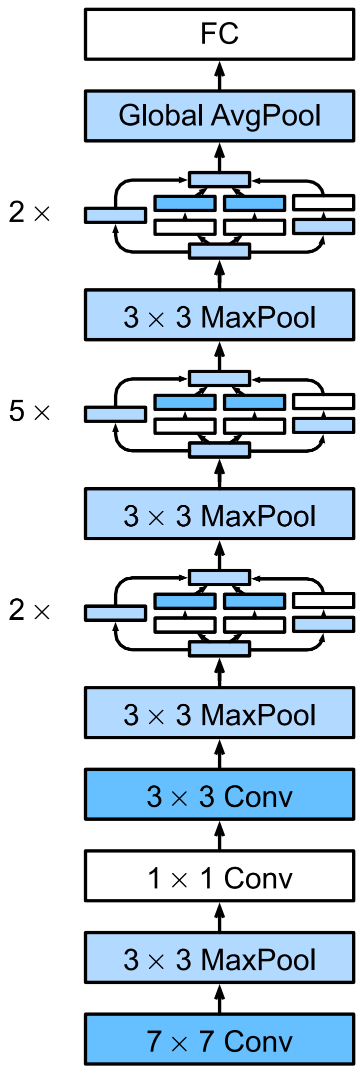


Figure 1.4 The GoogLeNet architecture:

- **ResNet**

  Residual Blocks: Let x is denoted as input, then the general assumption is that the desired underlying mapping and it is needed to obtain by learning is f(x), to be used as the input to the activation function on the top. On the left of Figure 1.5, the dotted-line box's portion must directly learn the mapping f(x). On the right, the amount within the dotted-line box needs to understand the residual mapping f(x) - x, which is how the residual block derives its name. Suppose the identity mapping f(x) = x is the desired underlying mapping. In that case, the residual mapping is more comfortable to learn. It is needed to push the weights and biases of the upper weight layer (e.g., fully connected layer and convolutional layer) within the dotted-line box zero. The right Figure in Figure. 1.5 illustrates the residual block of ResNet, where the solid line carrying the layer input x to the addition operator is called a residual connection (or shortcut connection). With residual blocks, inputs can forward propagate faster through the residual connections across layers.

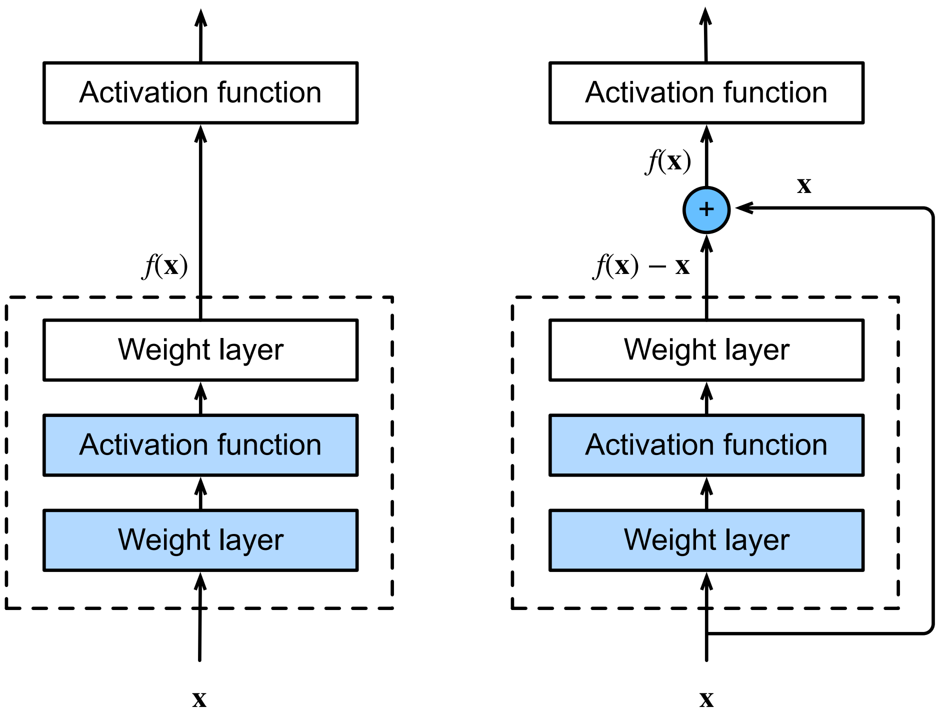


Figure 1.5: A regular block (left) and a residual block (right)

The first two layers of ResNet are the same as those of the GoogLeNet it. As described before: the 7×7 convolutional layer with 64 output channels and a stride of 2 is followed by the 3×3 maximum pooling layer with a stride of 2. The difference is the batch normalization layer added after each convolutional layer in ResNet.

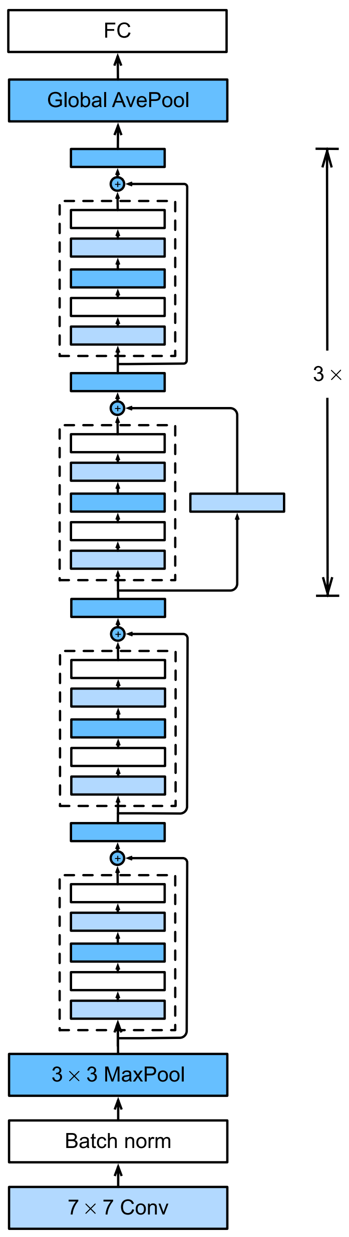


Figure 1.6: The ResNet-18 architecture

GoogLeNet uses four modules made up of Inception blocks. However, ResNet uses four modules made up of residual blocks, each of which uses several residual blocks with the same number of output channels. The number of channels in the first module is the same as the number of input channels. Since a maximum pooling layer with a stride of 2 has already been used, it is unnecessary to reduce the height and width. In the first residual block for each of the subsequent modules, the number of channels is doubled compared with that of the previous module, and the height and width are halved.

There are four convolutional layers in each module (excluding the 1×1 convolutional layer). Together with the first 7×7 convolutional layer and the fully connected layer, there are 18 layers. Therefore, this model is commonly known as ResNet-18. Configuring different numbers of channels and residual blocks in the module can create other ResNet models, such as the deeper 152-layer ResNet-152. Although the leading architecture of ResNet is similar to that of GoogLeNet, ResNet's structure is more straightforward and more comfortable to modify. All these factors have resulted in the rapid and widespread use of ResNet.

- **DenseNet**

Recall the Taylor expansion for functions. For the point *x*=0, it can be written as

$$f(x) = f(0) + f'(0)x + \frac{f''(0)}{2!}x^2 + \frac{f'''(0)}{3!}x^3 + \dots .$$

The critical point is that it decomposes a function into increasingly higher-order terms. In a similar vein, ResNet decomposes functions into

$$f(\mathbf{x}) = \mathbf{x} + g(\mathbf{x}).$$

That is, ResNet decomposes f into a simple linear term and a more complex nonlinear one. What if it is a need to capture (not necessarily add) information beyond two terms? One solution was DenseNet.

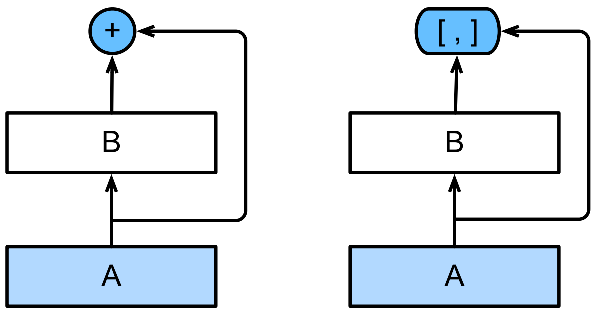


Figure 1.7: The main difference between ResNet (left) and DenseNet (right).

As shown in Fig. 1.7, the key difference between ResNet and DenseNet is that in the latter case outputs are concatenated (denoted by [,]) rather than added. As a result, perform a mapping from **X** to its values after applying an increasingly complex sequence of functions:

$$\mathbf{x} \to \left[\mathbf{x}, f_1(\mathbf{x}), f_2([\mathbf{x}, f_1(\mathbf{x})]), f_3([\mathbf{x}, f_1(\mathbf{x}), f_2([\mathbf{x}, f_1(\mathbf{x})])]), \ldots\right].$$

In the end, all these functions are combined in MLP to reduce the number of features again. In terms of implementation this is quite simple: rather than adding terms, lets concatenate them. The name DenseNet arises from the fact that the dependency graph between variables becomes quite dense. The last layer of such a chain is densely connected to all previous layers. The dense connections are shown in Fig. 1.8.

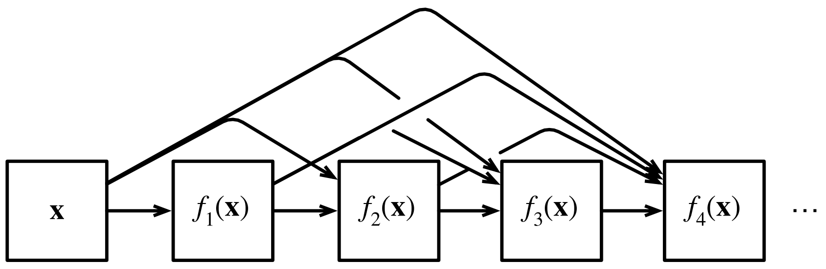


Figure 1.8: Dense connections in DenseNet.

The main components that compose a DenseNet are dense blocks and transition layers. The former defines how the inputs and outputs are concatenated, while the latter control the number of channels so that it is not too large.

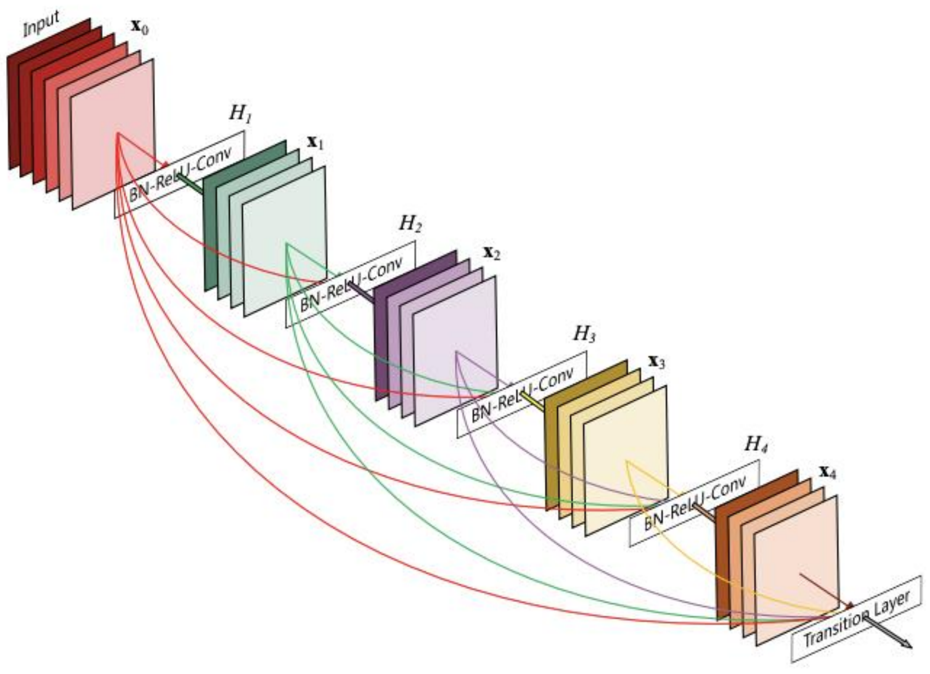


Figure 1.9: The DenseNet architecture

- **ResNext**

  ResNeXt won 2nd place in ILSVRC 2016 classification task and also showed performance improvements in Coco detection and ImageNet-5k set than their ResNet counterpart.

  This is a very simple model and the paper related to this model is simple to read which introduces a new term called "cardinality". The paper simply explains this term and make use of it in ResNet networks and does various ablation studies.

  The paper made several attempts to describe the complexity of Inception networks and why ResNeXt architecture is simple.

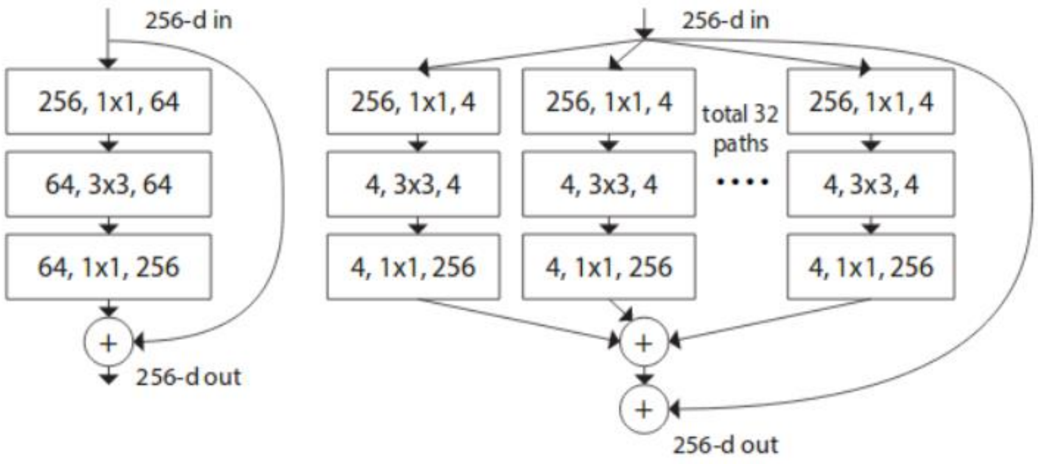
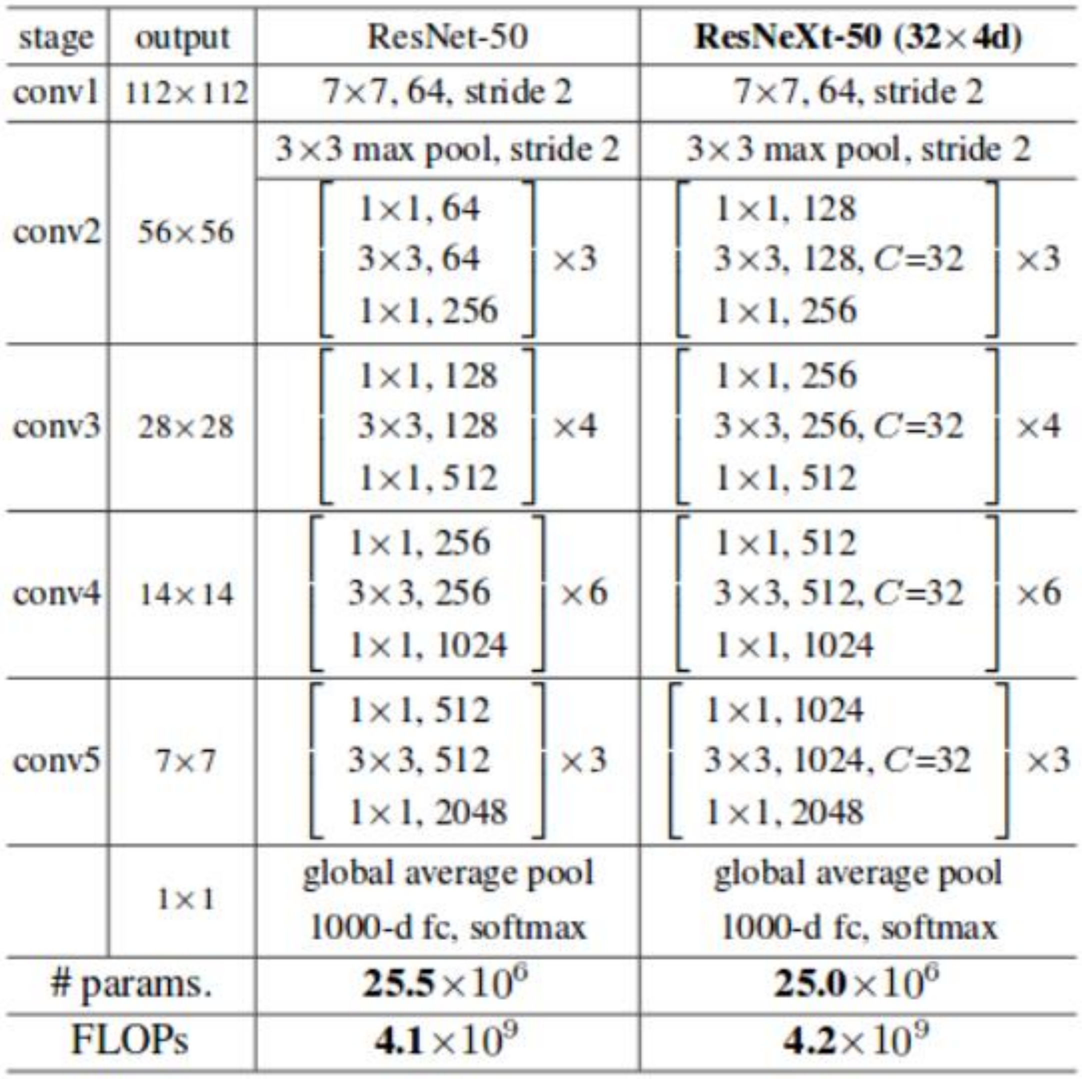


Figure 1.10: Left: A Block of ResNet. Right: A block of ResNeXt with cardinality = 32, with roughly the same complexity.

- The above diagram distinguishes between a simple ResNet block and ResNeXt blog.
- It follows a split-transform-aggregate strategy.
- The number of paths inside the ResNeXt block is defined as cardinality. In the above diagram C=32.
- All the paths contain the same topology.
- Instead of having high depth and width, having high cardinality helps in decreasing validation error.
- ResNeXt tries embedding more subspaces compared to its ResNet counterpart.

- Both the architectures have different width. Layer-1 in ResNet has one conv layer with 64 width, while layer-1 in ResNext has 32 different conv layers with 4 width (32*4 width). Despite the larger overall width in ResNeXt, both the architectures have the same number of parameters(~70k) (ResNet 256*64+3*3*64*64+64*26) (ResNeXt C*(256*d+3*3*d*d+d*256), with C=32 and d=4)



stage	output	ResNet-50	ResNeXt-50 (32×4d)
conv1	112×112	7×7, 64, stride 2	7×7, 64, stride 2
conv2	56×56	3×3 max pool, stride 2	3×3 max pool, stride 2
		$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128, C{=}32 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3	28×28	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256, C{=}32 \\ 1\times1, 512 \end{bmatrix} \times 4$
conv4	14×14	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512, C{=}32 \\ 1\times1, 1024 \end{bmatrix} \times 6$
conv5	7×7	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 1024 \\ 3\times3, 1024, C{=}32 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
# params.		**25.5**$\times10^6$	**25.0**$\times10^6$
FLOPs		**4.1**$\times10^9$	**4.2**$\times10^9$

Figure 1.11: (Left) ResNet-50 vs (Right) ResNeXt-50.

1.3 PROBLEM STATEMENT

This Research deals with the identifying ingredients, recipe process, and their nutritional information form the food image. Food is fundamental to human experience. It has deep ties to our health, our livelihood, our emotions, and our culture. More and more these days, people want to learn about what they are eating to make better nutritional decisions, but many struggles to find a place to start.

This Research aims to empower people to understand better and control their food intake to help achieve their health goals. This project aims to create a system that accepts an image of a meal as input and outputs closely related recipes and nutritional information.

1.3.1 Objectives

- To use the existing online could source (AWS) to handle, analyze massive data and collecting the part of the vast data.
- To understand the information Available in an image.
- To understand and modify the pre-trained CNN models to provide encodings to the image.
- **To Construct the K-NN model to retrieve the closest images by using distance or similarity as a metric.**
- To extract the possible ingredients, recipe, and nutritional information from the retrieved image.

1.4 REPORT OUTLINE

This chapter summarized the work scope, an introduction to some of the CNN models, which are helpful for image encodings used in this work. The rest of the report is organized as follows:

- Chapter 2 presents the already existing solutions and their drawbacks.
- Chapter 3 presents the proposed solution.
- Chapter 4 presents the implementation details, along with the results obtained.
- Chapter 5 concludes the work done.
- Chapter 6 presents the possible future work.

CHAPTER 2
LITERATURE SURVEY

This section discusses some of the earlier works related to this Research, It is well known that investigating on projects aimed at accomplishing similar tasks will be extremely helpful to guide in the right direction and gain a better sense of what came before us. In the past decade, there has been significant progress in the collection of food-recipe datasets. In the early stages, most works followed a classical recognition pipeline, focusing on feature combination and specialized datasets (mostly Asian cuisine). However, in 2014, Bossard et al. took a step forward and introduced the holistic Food-101 visual classification dataset with 101k images divided equally among 101 classes, setting a baseline accuracy of 50.8% [1].

Two years later in 2016, Chen and Ngo presented another large dataset with 65k recipes and 110k images, but it only covered Chinese cuisine [2]. In 2017, Salvador et al. released the Recipe1M+ dataset, which was a new large-scale, structured corpus of over one million cooking recipes and 13 million food images [3]. To date, this is the largest publicly available collection of food and recipe data and allows for the ability to train high-capacity models on aligned, multi-modal data. Therefore, this is the dataset chosen to use for this Research. There have also been a variety of approaches used to tackle the food recognition issue.

In 2010, Yang et al. proposed to learn spatial relationships between ingredients using pairwise features [4]. However, this was bound only to work for standardized meals. Along with the Food-101 dataset, Bossard et al. introduced a novel method to mine discriminative parts using Random Forests. Random Forests was used to discriminatively cluster super pixels into groups (leaves) and then used the leaf statistics to create features. The researchers applied a distinctiveness measure to the leaves to merge similar leaves, then used a support vector machine (SVM) to classify the food images into categories.

A few years later, Herranz et al. proposed an extended multi-modal framework that explores how visual content, context, and external knowledge can be integrated into food-oriented applications. [5]. Around the same time, Min et al. presented a multi-attribute theme modeling approach that incorporates attributes such as cuisine style, course type, flavors or ingredient types [6]. The model learns a common space between different food attributes and their corresponding food images. Finally, there have some interesting papers published specifically in the realm of extracting recipes from images. Chen et al. found that the manner of cutting and cooking ingredients play substantial roles in the food's appearance, and therefore attempted to predict ingredient, cutting, and cooking attributes given a food image [7]. On the other hand, Chang et al. looked at the possibility of several different preparations for one dish by clustering recipes based on their distance [8].

2.1 SUMMARY

This chapter enlisted the recent works and techniques related to image data, detecting ingredients from the image, image classifications, and clustering. This section helps to get the information on some of the earlier works related to this Research, finally found it extremely helpful to investigate projects aimed at accomplishing similar tasks to guide us in the right direction and gain a better sense of what came before us.

CHAPTER 3
PROPOSED WORK

Our project aims to get the details of similar ingredients, recipe steps, and query images' nutritional information. To achieve this project's aim, there should be some way to determine how two images are similar that would help retrieve the most similar image to the input image in the dataset. Similarity does not mean the similarity of pixel values, the similarity in objects or content, not in colors or object positions. It is well known that the layers in CNN architecture are used to identify from the lines to objects. The deep layer has the strength to recognize the objects. Therefore, encoding CNN would help capture the content of images while ignoring pixel-value differences caused by lighting, object position, discoloration, etc. To produce image encodings for every image in our dataset, the Recipe 1M training data runs through various convolutional neural networks, Alex Net, Dense Net 161, Google Net, ResNet 152, ResNext101_32x8d, and VGG 19, each trained on the ImageNet dataset.

With our entire dataset encoded through the various CNNs, the next step is to process the user input. The first step is to generate an encoding of the input image, bypassing the input image through the same CNN used to encode the dataset. Once this encoding is collected, next search linearly through the encoded dataset and calculate the k nearest neighbors to the input image from among the encoded datasets. The k is determined by the user depending on how many recipes they would like to see. This system supports running nearest neighbors with either cosine similarity or Euclidean distance as the distance metric between two encodings. Once the nearest neighbors are found, the recipes related to these images are displayed, along with nutritional information about these recipes.

3.1 Architecture:

The following Figure shows the step-by-step process of our Research as in the form of architecture of the proposed model. Figure 3.1 is the architecture of our Research.

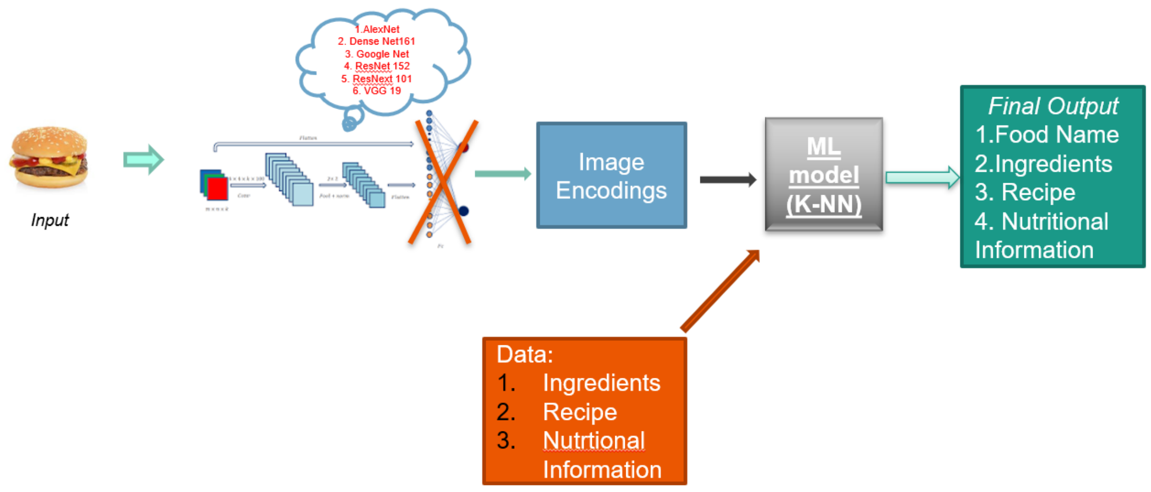


Figure 3.1: Architecture of the Proposed Model.

3.2 Encoding of the Images:

To identify or retrieve the similar image to the query image, it is needed to determine how similar the two images were so that an input image matched to the most similar image in our dataset can be retrieved. Encoding for an image must be done in a way that it should not be concerned with direct pixel value for evaluating similarity. In contrast, it should be concerned with the similarity in image content but not with colors and object positions. To do that, CNN could be used as an image encoding. Since each activation in later layers corresponds to some higher-level or abstract feature of the image, these encodings to capture the content of images while ignoring pixel-value differences caused by lighting, object position, discoloration, etc. Intending to produce image encodings for every image in our dataset, Recipe 1M training data ran through various convolutional neural networks, Alex Net, Dense Net 161, Google Net, ResNet 152, ResNext101_32x8d, and VGG 19, each trained on the ImageNet dataset. The next step is to check how to create an encoded vector by each Pretrained CNN model and analyze them.

Firstly, the CNN models are taken, which are pre-trained on the ImageNet Dataset from the official website of Pytorch. These models can also be downloaded from the python (pytorch) package '*torchvision.models*'. The following figures are the simplified architecture of those models (Alex Net, Dense Net 161, Google Net, ResNet 152, ResNext101_32x8d, and VGG 19, respectively).

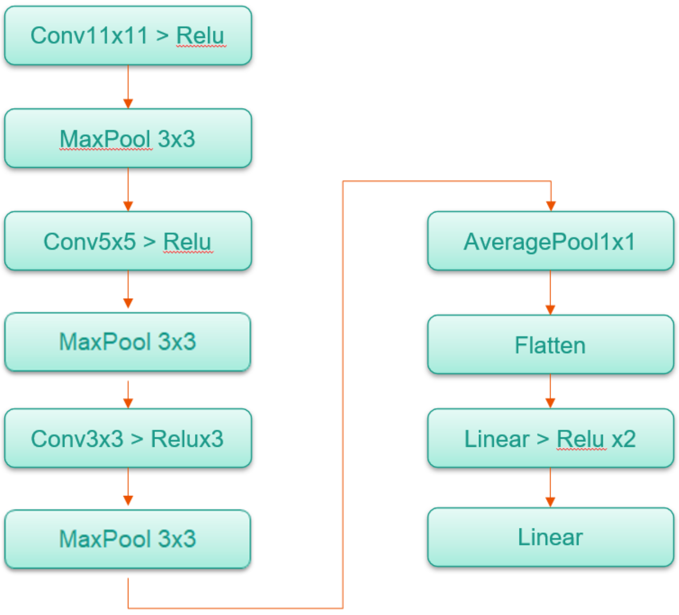
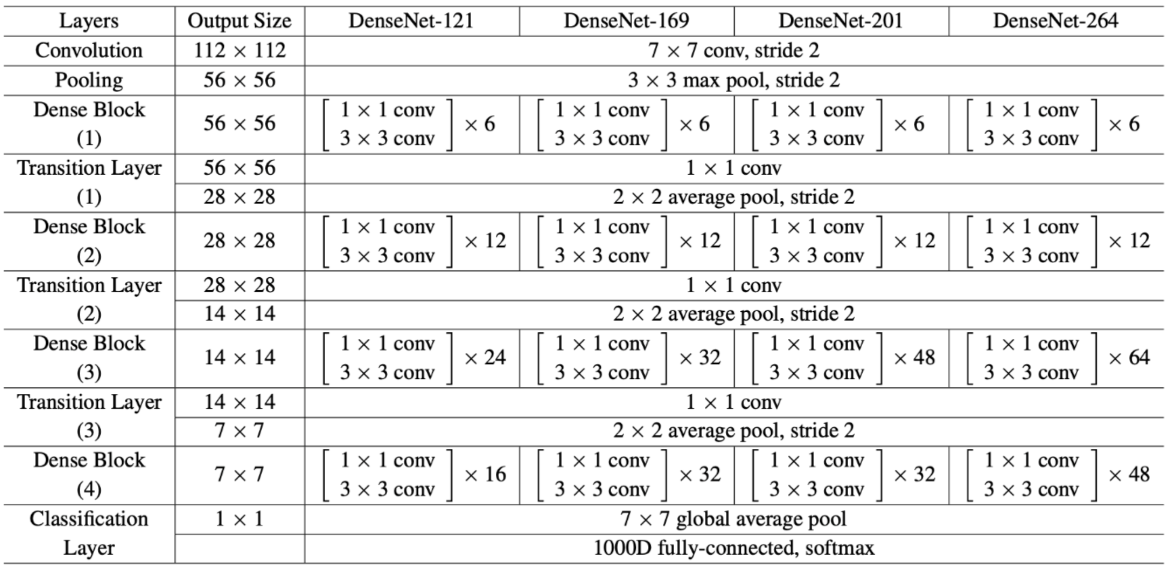
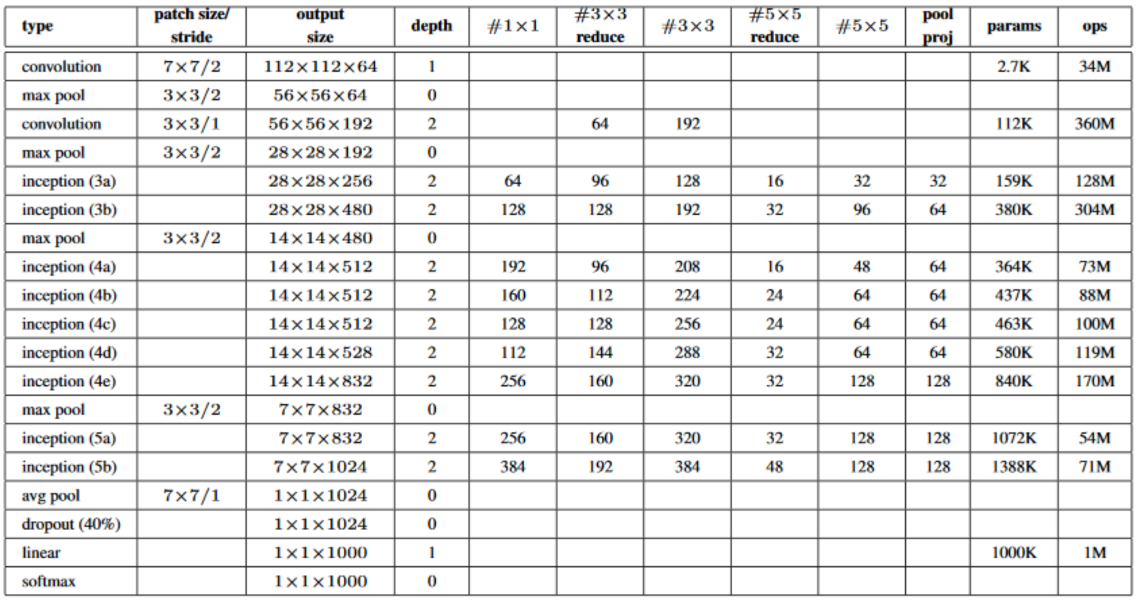


Figure 3.2. Layer Architecture of AlexNet obtained from Hidden Layer.



Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112×112	7×7 conv, stride 2			
Pooling	56×56	3×3 max pool, stride 2			
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer	56×56	1×1 conv			
(1)	28×28	2×2 average pool, stride 2			
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer	28×28	1×1 conv			
(2)	14×14	2×2 average pool, stride 2			
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer	14×14	1×1 conv			
(3)	7×7	2×2 average pool, stride 2			
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification	1×1	7×7 global average pool			
Layer		1000D fully-connected, softmax			

Figure 3.3: DenseNet Layer Architecture



type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Figure 3.4: GoogleNet Layer Architecture details.

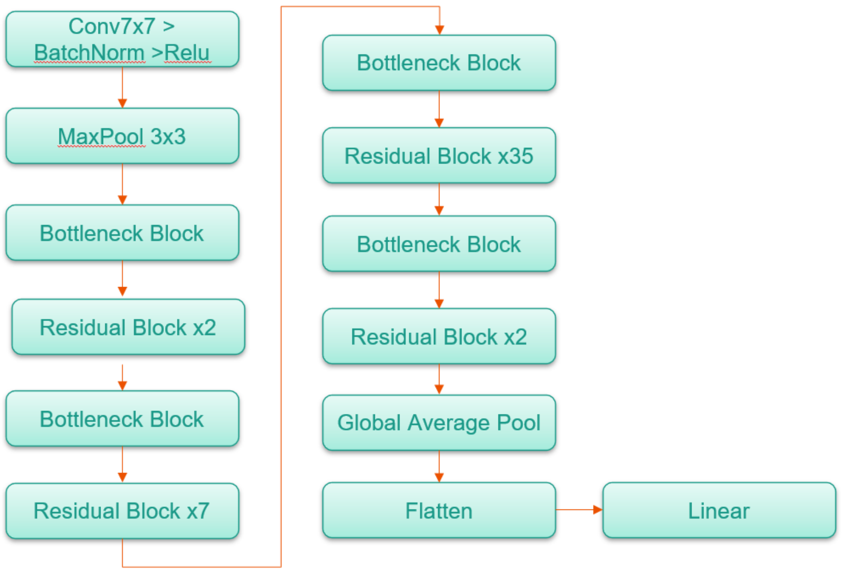


Figure 3.5: ResNet Layer Architecture

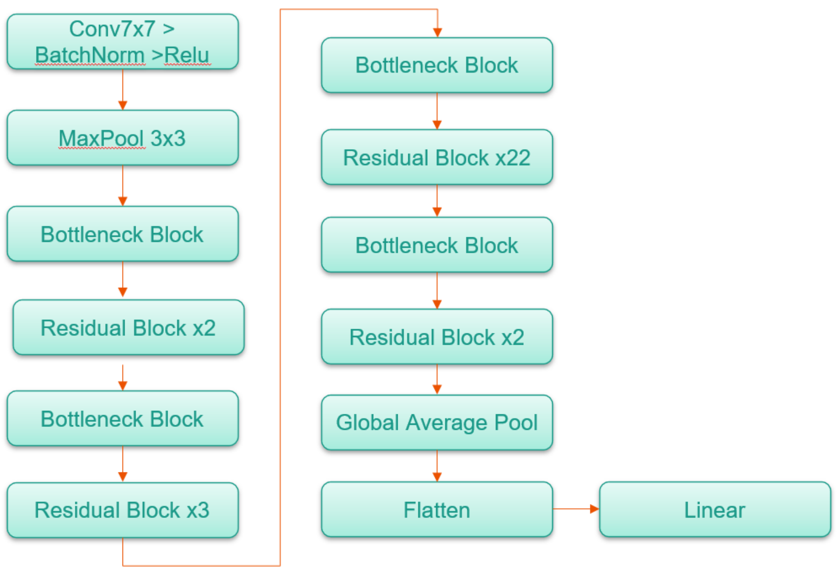


Figure 3.6: ResNeXt Layer Architecture

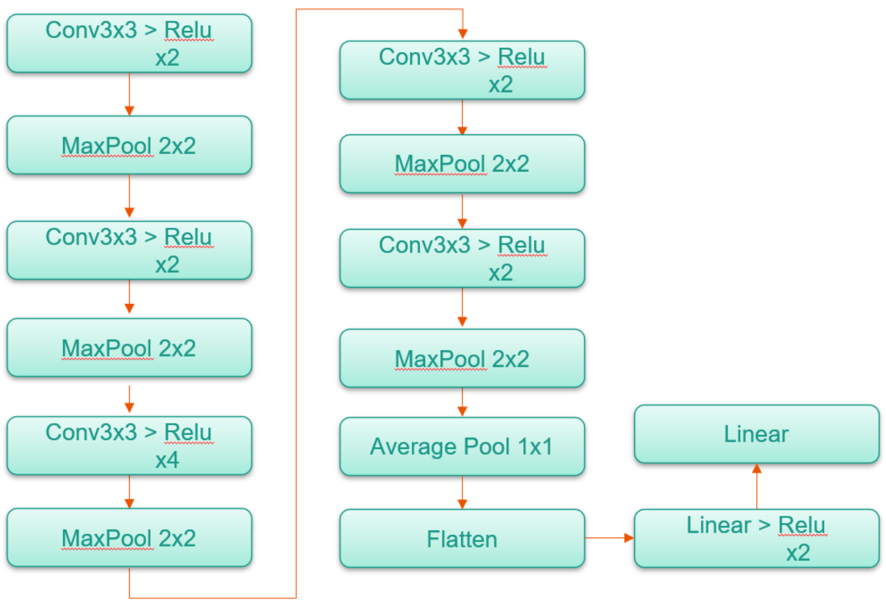


Figure 3.7: VGG 19 Layer Architecture

A further step in this Research is to provide encodings for each image, but these models are designed for Classification; therefore, it contains the fully connected layers and SoftMax layers, which are not needed for this task. So, the Fully Connected layers (Linear) and the SoftMax layers are removed and attached average pooling layer. Then, the modified models' obtained weight values are flattened to make the vector known as encoding for an image.

3.3 Producing encoding vector for Query Image:

After constructing each image's encodings in the dataset for each pre-trained models (Alex Net, Dense Net 161, Google Net, ResNet 152, ResNext101_32x8d, and VGG 19). The vector dimensions or shape of each encoding with respect to the CNN models may be different. As in Research, it is observed that vector size will be different for different encodings produced by different CNN models.

3.4 K-NN:

K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on the Supervised Learning technique. It assumes the similarity between the new case/data and available cases and puts the new case into the most similar category to the available categories. It stores all the available data and classifies a new data point based on the similarity. When new data appears, it can be easily classified into a well suite category by using the K- NN algorithm.

K-NN algorithm can be used for Regression and Classification, but mostly it is used for Classification problems. K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data. It is also called a lazy learner algorithm because it does not learn from the training set immediately. Instead, it stores the dataset, and at the time of Classification, it acts on the dataset. KNN algorithm at the training phase stores the dataset, and when it gets new data, it classifies that data into a category that is much similar to the new data.

3.4.1 Working procedure of KNN

The K-NN working can be explained based on the below algorithm:

Step-1: Select the number K of the neighbors.

Step-2: Calculate the Euclidean distance / Cosine Similarity of K number of neighbors.

Step-3: Take the K nearest neighbors as per the calculated Euclidean distance or cosine similarity value.

Step-4: Among these k neighbors, count the number of the data points in each category.

Step-5: Assign the new data points to that category for which the number of the neighbor is maximum.

Instead of doing Classification, this system will retrieve the top k nearest neighbors similar images to the query image with less distance value in the case of distance as metric and having high value in case of similarity as a metric.

Euclidean distance: The Euclidean distance between two points in either the plane or -dimensional space measures the length of a segment connecting the two points. It is the most obvious way of representing the distance between two points.

The Pythagorean Theorem can be used to calculate the distance between two points, as shown in the Figure below. If the points (x1, y1)
and (x2, y2) are in 2-dimensional space, then the Euclidean distance between them is

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$

For points (x1, y1, z1) and (x2, y2, z2) in 3-dimensional space, the Euclidean distance between them is

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}.$$

Cosine Similarity: Cosine similarity measures the similarity between two vectors of an inner product space. It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction.

Let x and y be two vectors for comparison. Using the cosine measure as a similarity function, have

$$sim(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{||\boldsymbol{x}||\,||\boldsymbol{y}||},$$

where ||x|| is the Euclidean norm of vector x = (x1, x2, …, xp), defined as $\sqrt{x_1^2 + x_2^2 + \cdots + x_p^2}$. Conceptually, it is the length of the vector.

3.5 DATASET

Because the Recipe1M+ dataset contained 1M recipes with 13M associated images, For this model, it is better to take a subset of the dataset (the Recipe1M dataset) instead due to computational and time constraints. Our Recipe1M dataset includes 402, 760 recipes with 887, 536 associated images, which was the perfect size for our project. Below are some statistics on the Recipe1M+ dataset, which gives a general idea of our Recipe1M subset.

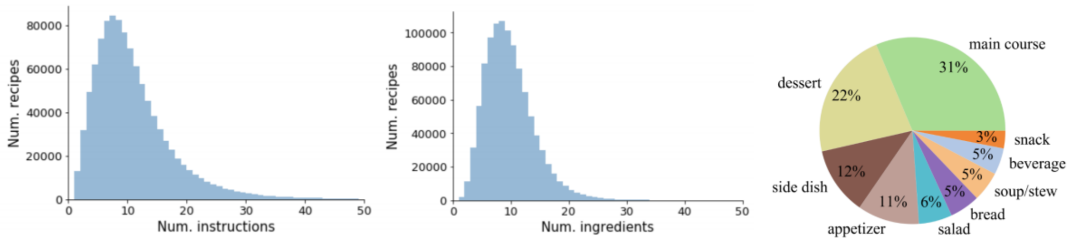


Figure 3.8: General Idea regarding recipe 1M dataset

Among those 402,760 further divided 280,747 images as the training set for encoding and 14,236 images for testing.

Besides the training images, the Recipe1M dataset was split into two dataframes. The first dataframe contains the id, ingredients, title, instructions, and original URL location of all 1M recipes in the Recipe1M+ set. The second dataframes have the ids of only the recipes in our Recipe1M set and a list of their associated image ids. Pre-processed the first and second dataset such that all the individual image ids were extracted to a new dataframe matched the image id to their respective recipes. There is also a third separate dataset created by the authors of Recipe1M that contained nutritional information on a subset of about 51k recipes. Later these images are pre-processed so that they were resized to have the shape (256, 256, 3) before those images ran through any of the CNN models. For processing all the images, AWS sage maker and AWS S3 are helpful to do that.

3.6 SUMMARY

This chapter enlisted the details of the proposed framework for constructing different encoding techniques in computer vision, which are applied to the food images. It clearly explains all the encodings used, architecture, and the dataset used.

CHAPTER 4
IMPLEMENTATION AND RESULTS

4.1 IMPLEMENTATION DETAILS

The system used for the experiments has the following configuration:

● Processor	Intel i5 8th Generation 3.9 GHz
● RAM	16 GB
● System Type	64-bit
● Operating System	Windows 10
● GPU	NVIDIA GeForce GTX 1050

4.2 SOFTWARE ENVIRONMENT

The following software tools and libraries were used:

● IDE	Jupiter Notebook Google Collaboratory, Amazon Sage Maker
● Programming Language	Python
● FrameWork	Pytorch
● Libraries	Numpy, PyTorch, SciPy, Matplotlib

4.3 DATA PROCESSING

First, the data is taken from the MIT Recipe 1M+. Then handled all the data on the AWS website. They were then renamed and resized each image in the train and test dataset and renamed each image with id (id for whole data) and image id. With the help of image id, all data is gathered. The size of each image is 256x256. Then all the processed images are downloaded into the local system.

● Train	280,747 images (shape: 256x256, size: 6.38Gb)
● Test	14,236 images (shape: 256x256, size: 326Mb)

4.4 ENCODING

All these 280,747 images passed to each pertained CNN model and generated the encodings for each image. Due to the size issue, the encoding of 10,000 images is saved into each dataframe and then merged into one data frame for each pre-trained CNN model. Since six models are taken, six encoding files are obtained. These encoding files contain the id, image id, and the encoded vectors. Each encoding has different dimensions and sizes.

MODEL	VECTOR DIMENSION	FILE SIZE
ALEX NET	1024 + 2	4.01 GB
DENSENET 161	1000 + 2	5.46 GB
GOOGLENET	1024 + 2	4.28 GB
RESNET 152	1000 + 2	4.96 GB
RESNEXT 101_32X8D	1000 + 2	4.97 GB
VGG 19	1024 + 2	5.22 GB

Table 4.1: Encodings Details

Visualization of Encoding using t-SNE: Once the encoding part is done, visualization of the model's encoding will help draw some conclusions. Since the data taken is vast, initially visualization 1% of complete data shown. The algorithm used for visualization is dimensionality reduction algorithm t-SNE; since it will take a tremendous amount of time to run on this vast data, run fast, and utilize some time, this data is on t-SNE should run on GPU, google colab is helped to achieve it. The following Figure shows the encoding of 1% encodings provided by some of the pre-trained models.

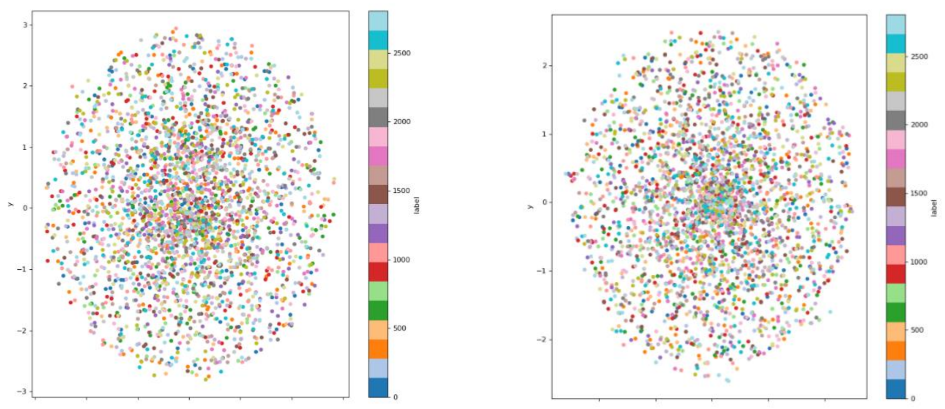


Figure 4.2: t-SNE visualization of 1% (left) DenseNet & (right) googleNet encodings.

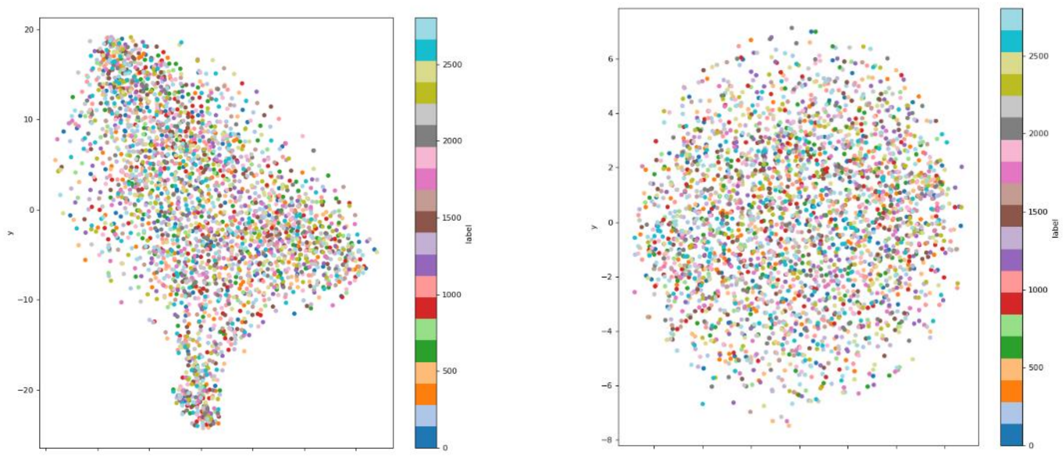


Figure 4.3: t-SNE visualization of 1% (left) ResNext & (right) VGG encodings

the visualization of whole data encodings even though it's difficult to draw some conclusions.

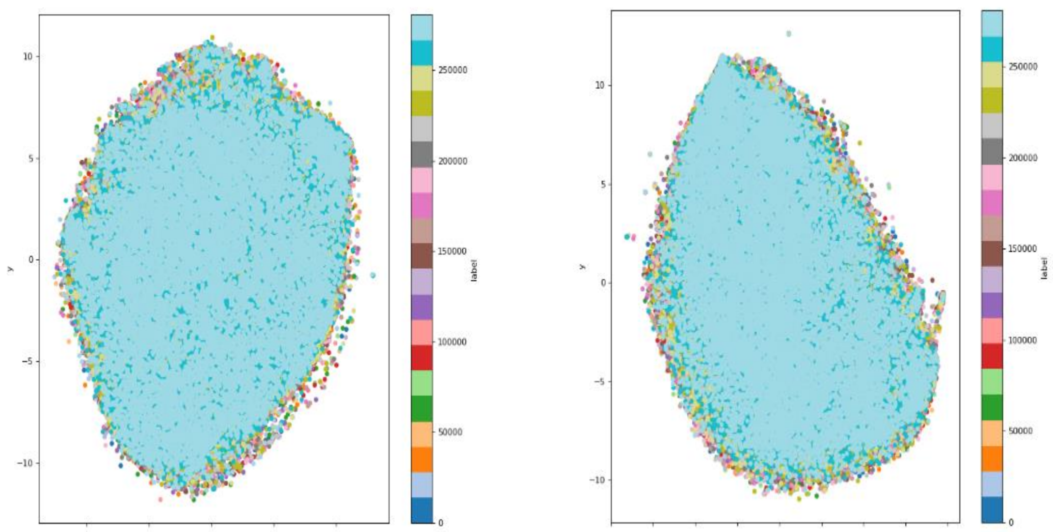


Figure 4.4: t-SNE visualization of (left) AlexNet & (right) ResNet encodings.

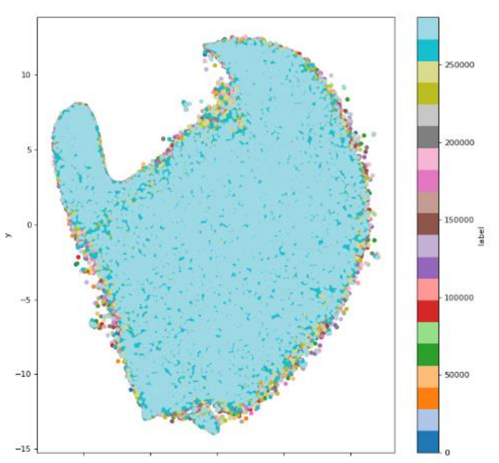
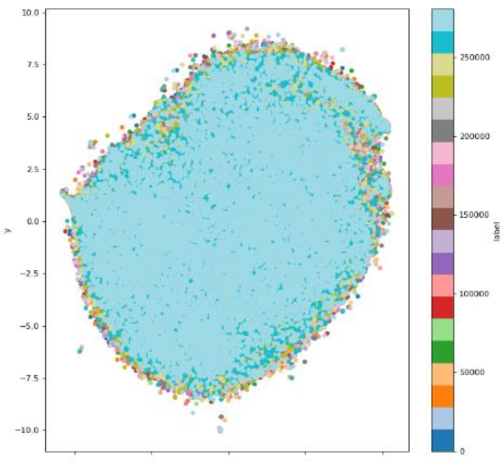



Figure 4.5: t-SNE visualization of (left) ResNext & (right) VGG 19 encodings.

4.5 K-NN IMPLEMENTATION

For testing, each query image is going through the respective pre-trained CNN model and generates the query vector, encoding to the respective image. This vector is named as a Query vector, Now using K-NN Algorithm retrieved the top 'K' similar image from the dataset and produce their data, i.e., ingredients, recipe process, and nutritional information, as the output. For this project, the K value is taken as 3.

4.6 RESULTS

To check the results, first, it is necessary to know what type of images the model able to retrieve the images from the dataset, for it is necessary to check and compare the predicted image with ground truth values/image for each pre-trained model encoding on K-NN algorithm with different metrics. Since this Research, K's value is taken as three; hence, there will be top 3 similar images in this system. those three retrieved images and compare them with ground truth images. To do that, it is better to take five test images for distance and similarity as a metric.

4.6.1 Distance as Metric for K-NN

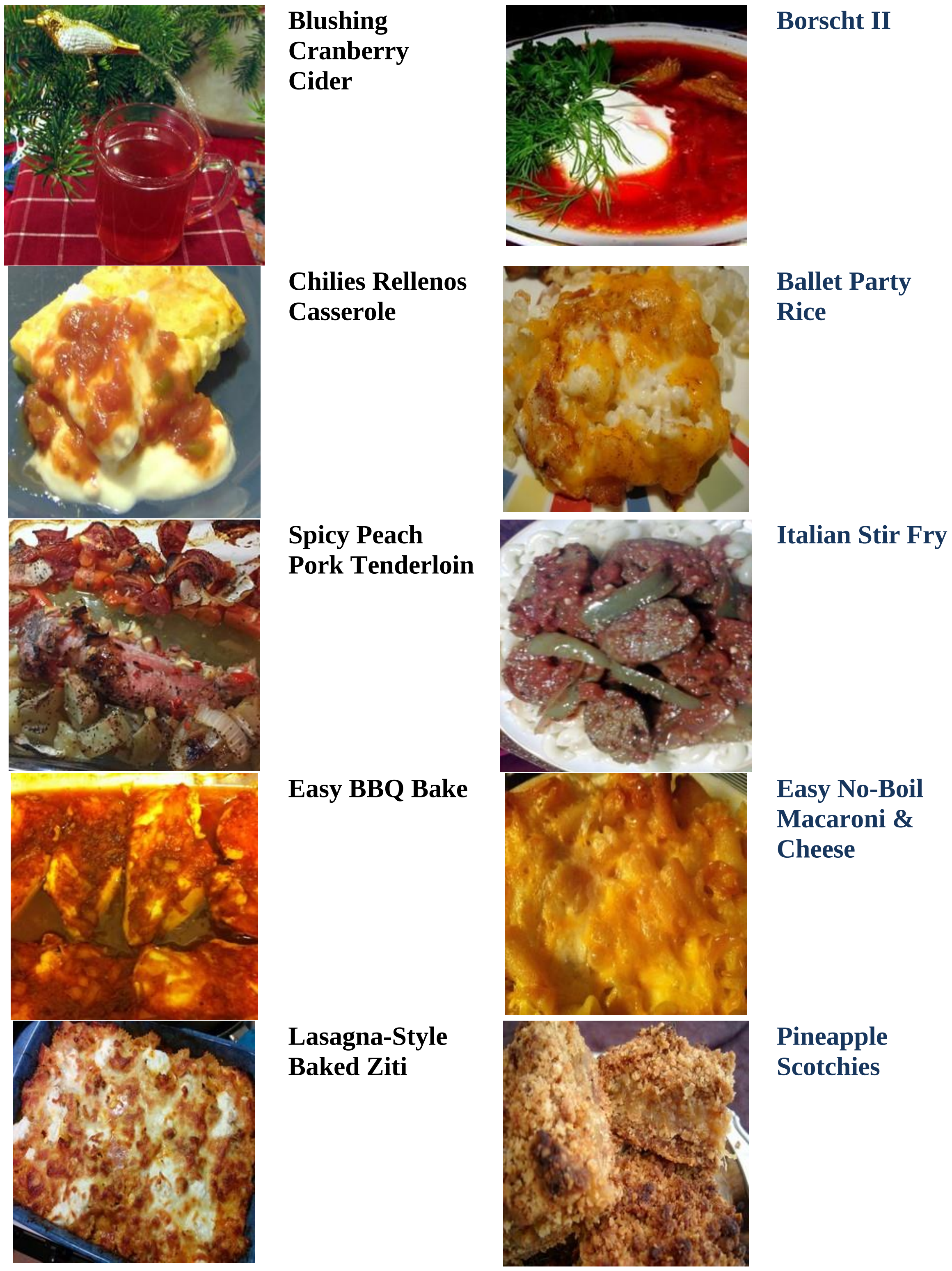


Fig 4.6: comparison between (left)ground truth and (right)Top 1st similar image retrieved by distance as metric for 3-NN with AlexNet encoding.

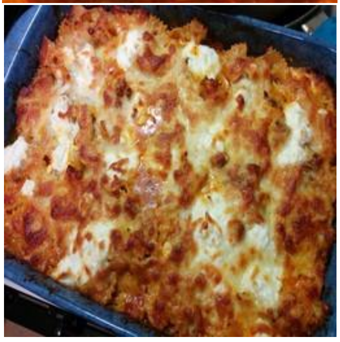
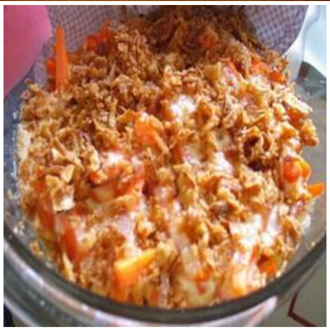
	Blushing Cranberry Cider		**Children's Raspberry 'mojito' (Nonalcoholic)**
	Chilies Rellenos Casserole		**Apricot Applesauce (Jewish)**
	Spicy Peach Pork Tenderloin		**Slow-Cooker Pepper Steak**
	Easy BBQ Bake		**sunshines home style meat loaf**
	**Lasagna-Style Baked Ziti**		**Sister Mary's Zesty Carrots**

Figure 4.7: comparison between (left)ground truth and (right)Top 2nd similar image retrieved by distance as metric for 3-NN with AlexNet encoding.

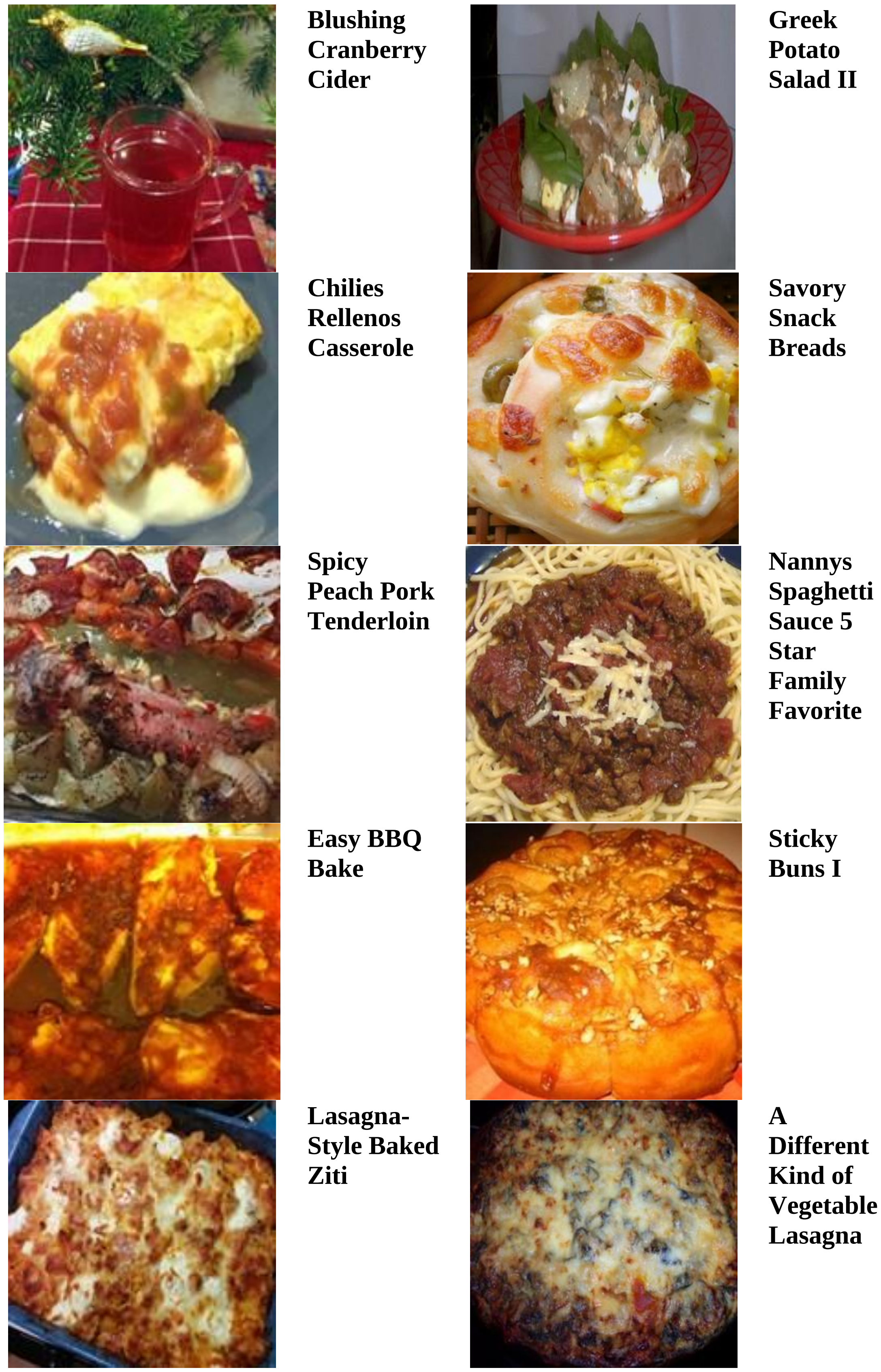

Blushing Cranberry Cider
Greek Potato Salad II
Chilies Rellenos Casserole
Savory Snack Breads
Spicy Peach Pork Tenderloin
Nannys Spaghetti Sauce 5 Star Family Favorite
Easy BBQ Bake
Sticky Buns I
Lasagna-Style Baked Ziti
A Different Kind of Vegetable Lasagna

Figure 4.8: comparison between (left)ground truth and (right) top 3rd similar image retrieved by distance as metric for 3-NN with AlexNet encoding.

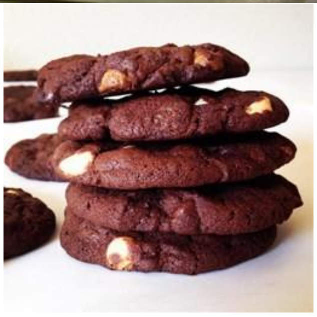
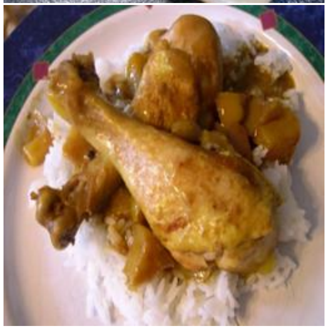
	Modenese Pork Chops		**Blueberry Buttermilk Pancakes**
	Asparagus Salad		**Asparagus Salad**
	Churros		**Cap'n Crunch Cod (Fish) Tacos by Food Dudes**
	Sausage Egg Casserole		**Snobahr's Baked Mac 'n Cheese**
	**Chocolate Chocolate Chip Cookies I**		**Chicken From Hell**

Figure 4.9: comparison between (left)ground truth and (right) top 1st similar image retrieved by distance as metric for 3-NN with ResNext101 encoding.

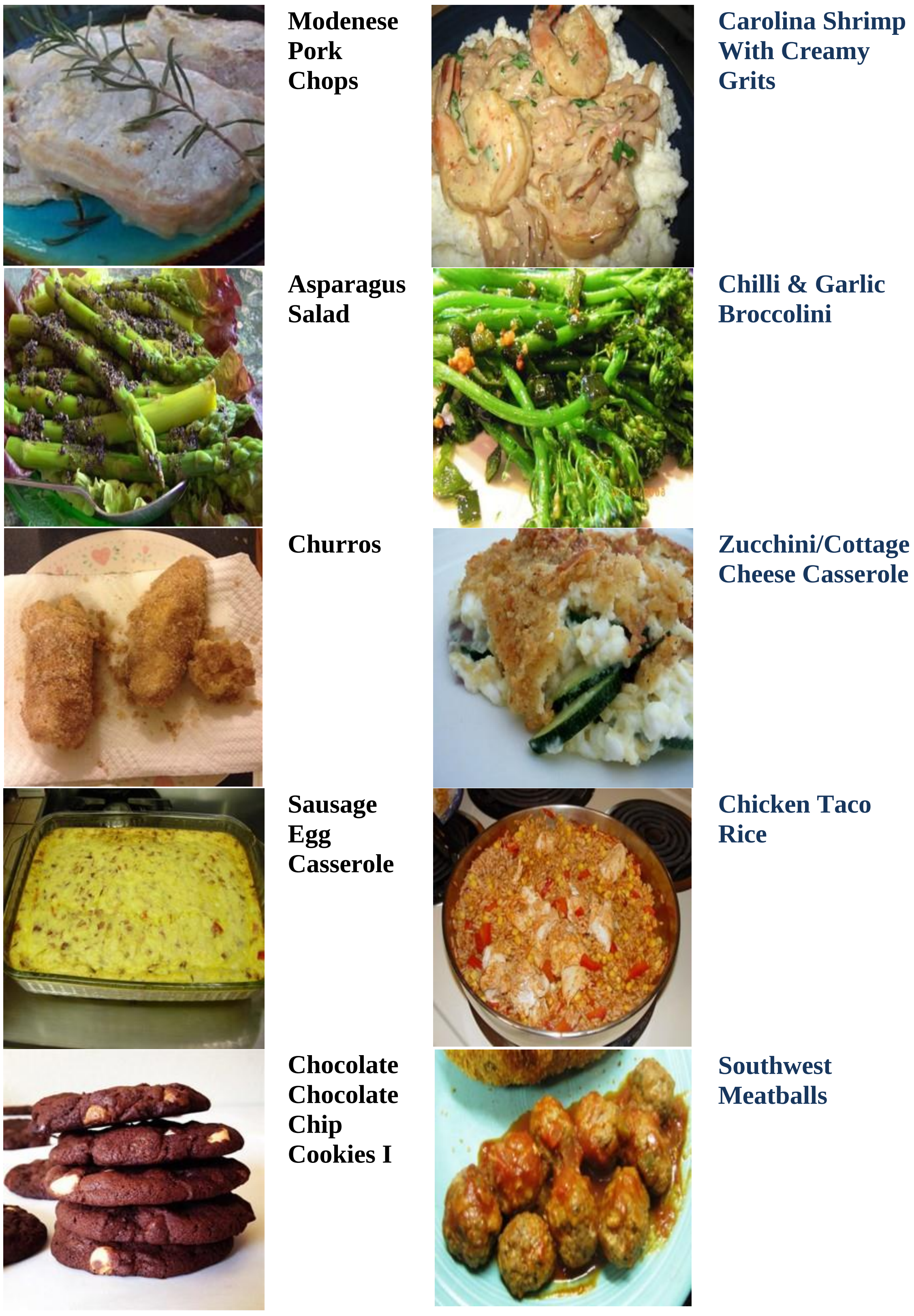
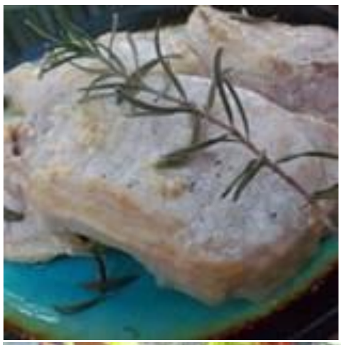
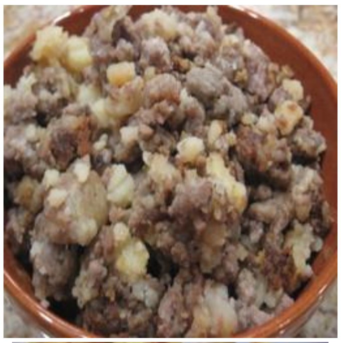

Modenese Pork Chops
Carolina Shrimp With Creamy Grits
Asparagus Salad
Chilli & Garlic Broccolini
Churros
Zucchini/Cottage Cheese Casserole
Sausage Egg Casserole
Chicken Taco Rice
Chocolate Chocolate Chip Cookies I
Southwest Meatballs

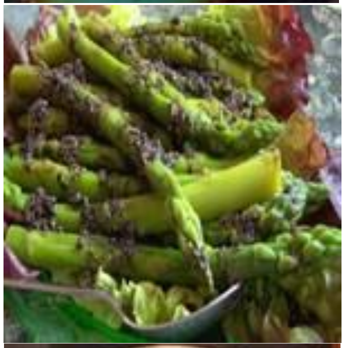
Figure 4.10: comparison between (left)ground truth and (right) top 2nd similar image retrieved by distance as metric for 3-NN with ResNext 101 encoding.



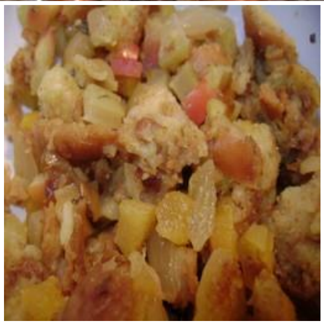
Modenese Pork Chops



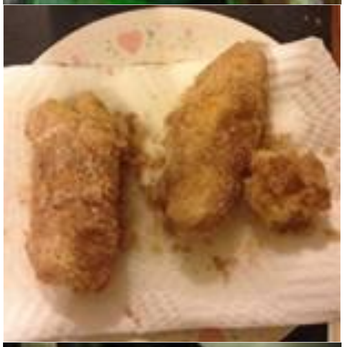
Janet's Thanksgiving Meat Stuffing



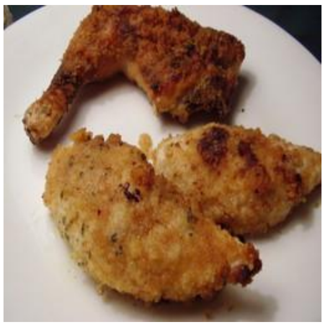
Asparagus Salad



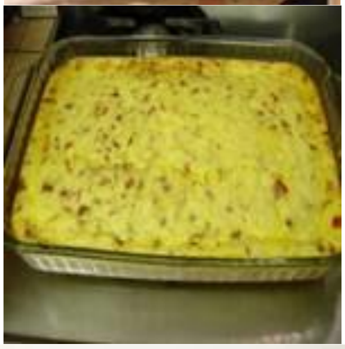
Autumn Harvest Stuffing (Aka Throw Away the Box of Stove Top!)



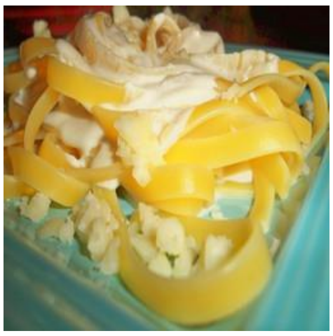
Churros



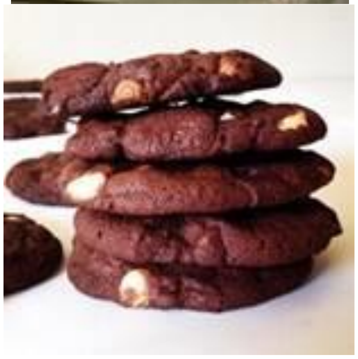
Crunchy Chicken



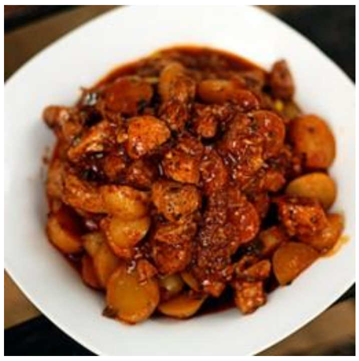
Sausage Egg Casserole



The Best Alfredo Sauce



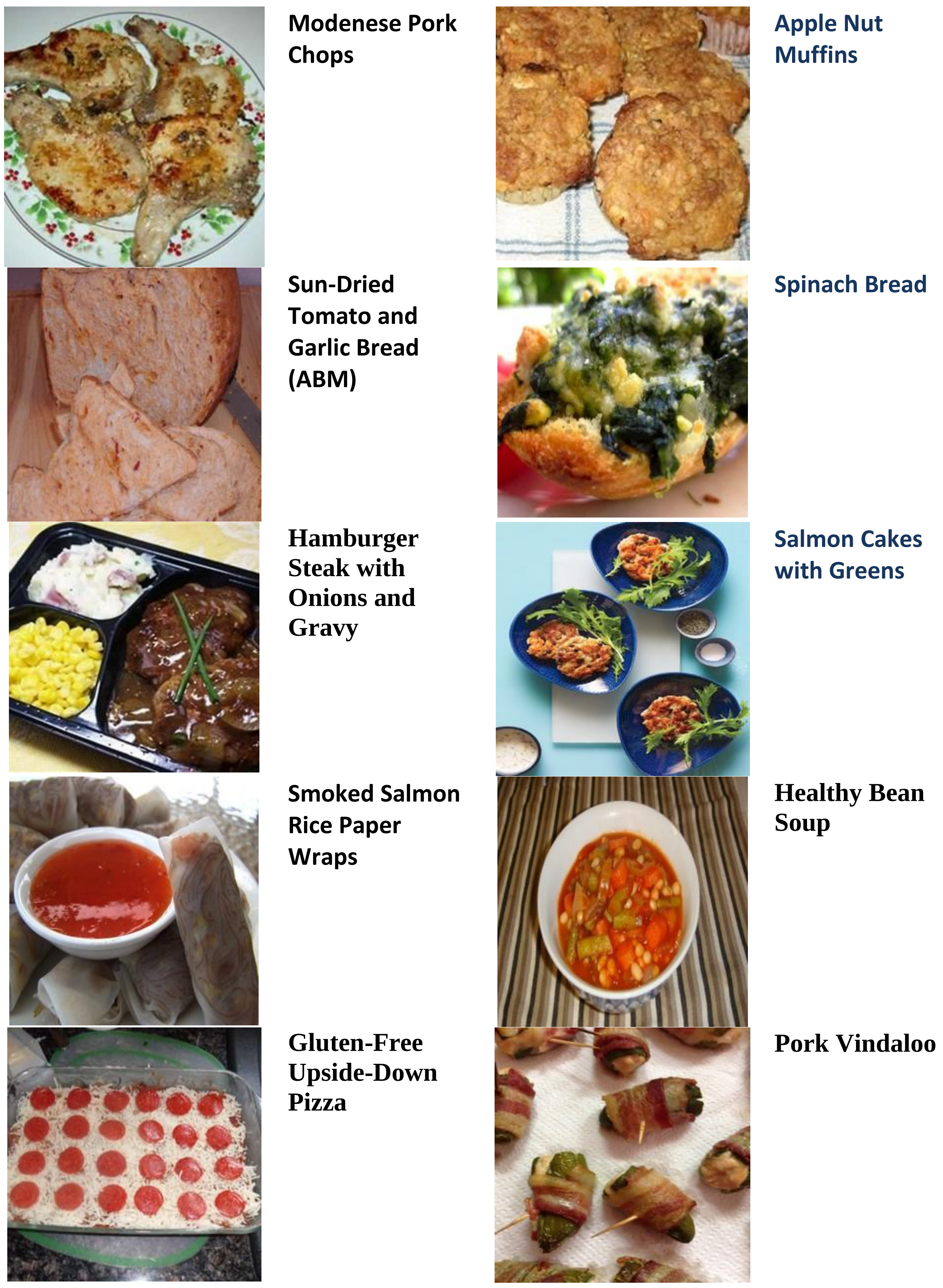
Chocolate Chocolate Chip Cookies I



Pork Vindaloo

Figure 4.11: comparison between (left)ground truth and (right)Top 3rd similar image retrieved by distance as metric for 3-NN with ResNext101 encoding.

4.6.2 Cosine Similarity as Metric for K-NN:



Modenese Pork Chops
Apple Nut Muffins
Sun-Dried Tomato and Garlic Bread (ABM)
Spinach Bread
Hamburger Steak with Onions and Gravy
Salmon Cakes with Greens
Smoked Salmon Rice Paper Wraps
Healthy Bean Soup
Gluten-Free Upside-Down Pizza
Pork Vindaloo

Figure 4.12: comparison between (left)ground truth and (right) top 1st similar image retrieved by cosine similarity 3-NN with ResNet encoding.

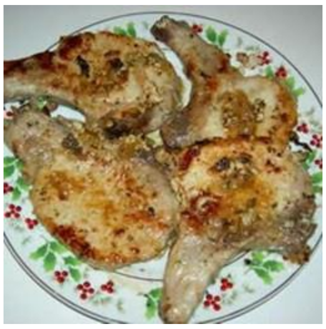
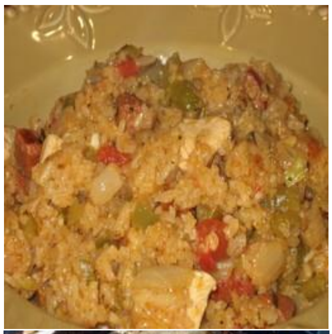
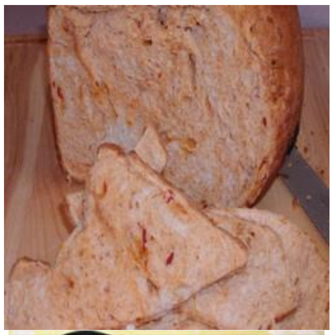
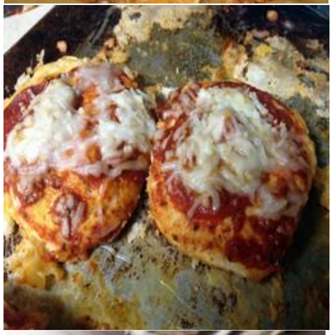
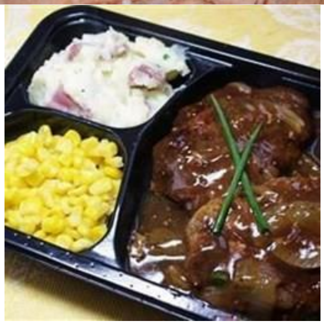
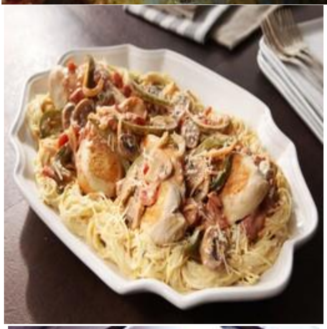
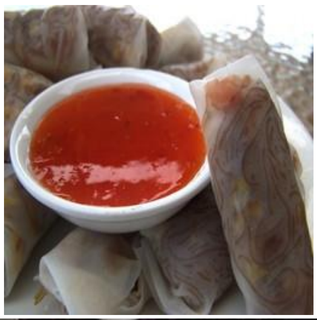
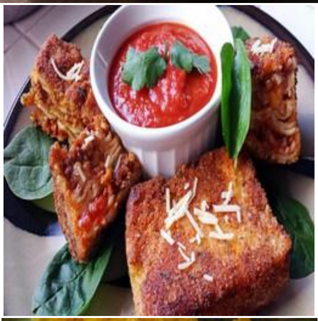
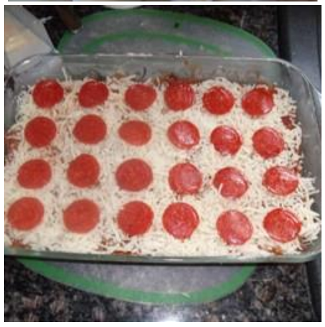
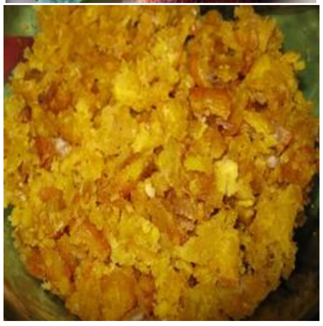
	**Modenese Pork Chops**		**Louisiana Sausage Jambalaya**
	**Sun-Dried Tomato and Garlic Bread (ABM)**		**Pizza Dillies**
	**Hamburger Steak with Onions and Gravy**		**Chicken Cacciatore with Creamy Angel Hair**
	**Smoked Salmon Rice Paper Wraps**		**Ray's' ~Fried Lasagna Squares~**
	**Gluten-Free Upside-Down Pizza**		**Smashed Garlic Plantains - Mofongo**

Figure 4.13: comparison between (left)ground truth and (right) top 2nd similar image retrieved by cosine similarity 3-NN with ResNet encoding.

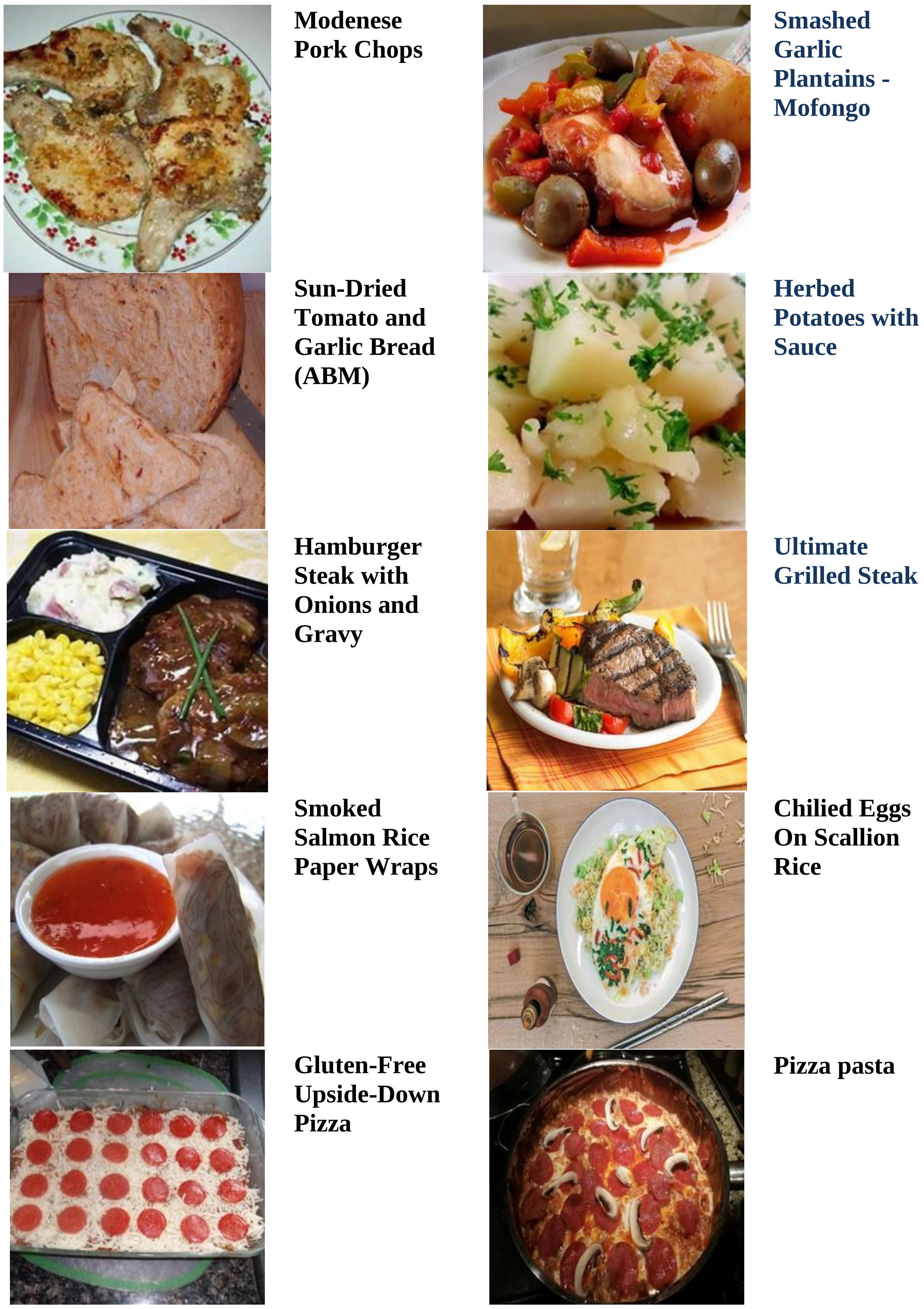

Modenese Pork Chops
Smashed Garlic Plantains - Mofongo
Sun-Dried Tomato and Garlic Bread (ABM)
Herbed Potatoes with Sauce
Hamburger Steak with Onions and Gravy
Ultimate Grilled Steak
Smoked Salmon Rice Paper Wraps
Chilied Eggs On Scallion Rice
Gluten-Free Upside-Down Pizza
Pizza pasta

Figure 4.14: comparison between (left)ground truth and (right) top 3rdst similar image retrieved by cosine similarity 3-NN with ResNet encoding.

The next step is to check the overall loss and number of images retrieved with zero loss. For evaluation, randomly 14236 images are taken. To evaluate this model, this model is tested with the relative accuracy (number of images retrieved with zero losses)/similarity of the encoding of images to the other images attached to the same recipe. Average cosine similarity and average Euclidean distance, where values of 1 and 0 are perfect matches of two images, were two metrics used to quantify this. Using these metrics to measure the similarity of an image to others associated with the same recipe helps quantify how accurate our found encodings are for an image. Thus the relative accuracy can be measured as the number of images retrieved with zero.

MODEL	Distance_loss_top1	Distance_loss_top2	Distance_loss_top3
Alex Net	35.6281	36.8806	37.2691
Dense Net 161	7.96E-07	8.10E-07	8.14E-07
Google Net	7.8552	7.893	7.9125
ResNet 152	0.7635	0.7779	0.7779
ResNext101_32x8d	0.7643	0.778	0.7813
VGG 19	3.1642	3.2286	3.2448

Table 4.2: Distance Loss

MODEL	Cosine_loss_top1	Cosine_loss_top2	Cosine_loss_top3
Alex Net	0.7538	0.7388	0.7331
Dense Net 161	1	1	1
Google Net	0.8374	0.8357	0.8349
ResNet 152	0.9988	0.9988	0.9988
ResNext101_32x8d	0.9986	0.9986	0.9986
VGG 19	0.9034	0.9004	0.8995

Table 4.3: Cosine Similarity Loss

Number of Images Matched with Zero Loss:

MODEL	Distance_loss_top1	Distance_loss_top2	Distance_loss_top3
Alex Net	569	60	35
Dense Net 161	101	0	1
Google Net	0	0	0
ResNet 152	230	19	7
ResNext101_32x8d	262	28	20
VGG 19	307	22	14

Table 4.4: Number of images matched with zero Loss with Distance Metric

MODEL	Distance_loss_top1	Distance_loss_top2	Distance_loss_top3
Alex Net	646	57	31
Dense Net 161	1	0	0
Google Net	0	1	0
ResNet 152	239	14	9
ResNext101_32x8d	307	25	14
VGG 19	311	10	10

Table 4.5: Number of images matched with zero Loss with Cosine similarity Metric

Finally, the system will give the output or result (The result is restructured into a table.) for the query image shown in the following table.

Query image	**Title**	**ingredients**	**recipe**	**Nutrition status**
	Simple Butter milk Biscuits	cup all-purpose flour '1/4 tsp baking soda' '1 tbsp baking powder '1 tsp salt' '6 tbsp butter,	Preheat oven to 400 degees. 'Place flour, baking soda & powder and salt in food processed. 'I keep my butter in the freezer to get extra cold while gathering my other ingredients. 'Add butter to food processer and pluse 1-3	**fat_status**: Orange **salt_status:** Red **saturates_status:** Red **Sugar_status:** Green

		cubed and very cold 1 cup buttermilk	times until the mixture resembles coarse sand. 'Add buttermilk in batches, pulse after each addition of buttermilk. 'Stop pulsing when dough is just combined. 'Dump dough onto floured surface. 'Pat dough together, handling it as little as possible. 'Cut out biscuits, reuse the leftover dough by kneading a few times and patting out again. 'The biscuits made from rerolled dough will not be as fluffy as the first batch. 'Place biscuits on a non-stick baking sheet. 'Bake for 10-12 mins or until golden & fluffy. 'You can freeze these biscuits on baking sheet until frozen through. 'Place in storage bag to use at a later date. 'To bake from freezer, preheat oven to 400F and bake for 20 mins.	Energy : 270.4816 cal Fat: 12.26649 Protein: 5.763952 Salt: 1.907074 Saturates 7.44323380187973 Sugar: 2.076021

Table 4.6: Output for query image

4.7 SUMMARY

This chapter enlisted all the observations and results from the experiments. Figures 4.6.10 and 4.6.11 lists the loss with distance and similarity as a metric, respectively. It results that Resent-152 and ResNext101_32x8d performed similarly in terms of loss; if distance as metric for K-NN was taken, then the loss obtained with 0.7635 and 0.7643 respectively. Whereas DenseNet 161 performed with less loss 7.96E-07 and cosine similarity with the highest loss as ~1. Resent-152 and ResNext101_32x8d also reached nearer to DenseNet 161 with 0.9973 and 0.9958, respectively. ResNext101_32x8d has retrieved more zero loss images than DenseNet 161; therefore, from the results obtained from all model encodings, it is recommended to take the ResNext101_32x8d model for encoding and Cosine Similarity as metric for the K-NN.

CHAPTER 5

CONCLUSION

The limitations of time and computing power are ultimately the main constraints in this project. Without such restrictions, future steps could be taken towards implementing models and algorithms that take advantage of these absent limitations in return for higher accuracy in image classification and closer recipe recommendations. Our model relied on utilizing pre-trained image classification models such as Alex Net, Dense Net 161, Google Net, ResNet 152, ResNext101_32x8d, and VGG 19 to handle the initial Classification of images. Integrating the models with the lowest validation errors would be the obvious next step to guaranteeing improvements to our existing model. Implementing these versions would expand the range of images our model can accurately classify.

It could also train on the entire Recipe1M+ dataset instead of the Recipe 1M subset, which would give us a larger variety of recipes to choose from and improve our performance for higher values of k. Finally, it is possible to expand our project such that users could opt to input a recipe and get a corresponding image. This would likely require the recipes' specific instructions and ingredients to be encoded and work alongside our image encodings.

REFERENCES

1] L. Bossard, M. Guillaumin, and L. Van Gool. Food-101– mining discriminative components with random forests. In European Conference on Computer Vision, pages 446–461. Springer, 2014.

[2] C.-w. N. Jing-jing Chen, "Deep-based ingredient recognition for cooking recipe retrival," ACM Multimedia, 2016.

[3] Salvador, Amaia, Hynes, Nicholas, Aytar, Yusuf, Marin, Javier, Ofli, Ferda, Weber, Ingmar, and Toralba, Antonio. Learning cross-model embbeddings for cooking recipes and food images. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017.

[4] Yang, S.L., Chen, M., Pomerleau, D., Sukthankar, R.: Food recognition using statistics of pairwise local features. In: CVPR (2010)

[5] L. Herranz, W. Min, and S. Jiang, "Food recognition and recipe analysis: integrating visual content, context and external knowledge," CoRR, vol. abs/1801.07239, 2018. [Online]. Available: http://arxiv.org/abs/1801.07239

[6] W. Min, S. Jiang, S. Wang, J. Sang, and S. Mei, "A delicious recipe analysis framework for exploring multimodal recipes with various attributes," in Proceedings of the 2017 ACM on Multimedia Conference, ser. MM '17. New York, NY, USA: ACM, 2017, pp. 402–410. [Online]. Available: http://doi.acm.org/10.1145/3123266.3123272

[7] J.-j. Chen, C.-W. Ngo, and T.-S. Chua, "Cross-modal recipe retrieval with rich food attributes," in Proceedings of the 2017 ACM on Multimedia Conference, ser. MM '17. New York, NY, USA: ACM, 2017, pp. 1771–1779. [Online]. Available: http://doi.acm.org/10.1145/3123266.3123428

[8] M. Chang, L. V. Guillain, H. Jung, V. M. Hare, J. Kim, and M. Agrawala, "Recipescape: An interactive tool for analyzing cooking instructions at scale," in Proceedings of the 2018 CHI Conference on Human Factors
in Computing Systems, ser. CHI '18. New York, NY, USA: ACM, 2018, pp. 451:1–451:12. [Online]. Available: http://doi.acm.org/10.1145/3173574.3174025

[9] Manish Chablani," DenseNet", Towards Data Science. https://towardsdatascience.com/densenet-2810936aeebb

[10] Gao, Hao." The Efficiency of Densenet." Medium. Last modified August 15, 2017. https://medium.com/@smallfishbigsea/densenet-2b0889854a92.

[11] Krizhevsky, Alex. "One weird trick for parallelizing convolutional neural networks." *arXiv preprint arXiv:1404.5997* (2014).

[12] Huang, Gao, et al. "Densely connected convolutional networks." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.

[13] Szegedy, Christian, et al. "Going deeper with convolutions." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015.

[14] He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

[15] Xie, Saining, et al. "Aggregated residual transformations for deep neural networks." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.

[16] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556* (2014).