UNIVERSITAT DE BARCELONA

Facultat de Matemàtiques
i Informàtica

GRAU DE MATEMÀTIQUES - GRAU D'ENGINYERIA INFORMÀTICA

Treball final de grau

Using Deep Learning for Food Recognition

Autora: Ling Zhu

Directores: Dra. Petia Radeva i Bhalaji Nagarajan
Realitzat a: Departament de Matemàtiques i Informàtica

Barcelona, September 13, 2020

Abstract

Image recognition is a very challenging and important problem in the computer vision field. And food image classification is one of the most challenging branches of this field.

In real-world scenarios, it is more common for a food image to have more than one food item. As a result, the multi-label classification problem has generated significant interest in recent years. However, multi-label recognition is a much more difficult object recognition task compared to single-label recognition. In this work, we will study the multi-label food recognition problem by using deep learning algorithms, specifically Convolutional Neural Networks. We will show how redefining the loss function as well as augmenting the training dataset can leverage the multi-label food recognition problem. Extensive validation will be presented to show the strengths and limitations of multi-label food recognition.

Resum

El reconeixement d'imatges és un problema molt desafiant i important en el camp de la visió per computador. I la classificació d'imatges alimentàries és una de les branques més difícils d'aquest camp.

En els escenaris del món real, és més freqüent que una imatge alimentària tingui més d'un aliment. Com a resultat, el problema de classificació multietiquetes ha generat un interès significatiu en els darrers anys. Tanmateix, el reconeixement de diverses etiquetes és una tasca de reconeixement d'objectes molt més difícil que el reconeixement d'una sola etiqueta. En aquest treball, estudiarem el problema del reconeixement d'aliments amb etiquetes múltiples mitjançant l'ús d'algoritmes d'aprenentatge profund, específicament Convolutional Neural Networks. Mostrarem com la redefinició de la funció de pèrdua i l'augment del conjunt de dades d'entrenament poden ajudar en el problema de reconeixement d'aliments de múltiples etiquetes. Es presentarà una àmplia validació per mostrar els punts forts i les limitacions del reconeixement d'aliments multietiqueta.

Resumen

El reconocimiento de imágenes es un problema muy desafiante e importante en el campo de la visión por computadora. Y la clasificación de imágenes de alimentos es una de las ramas más desafiantes de este campo.

En escenarios del mundo real, es más común que una imagen de comida tenga más de un alimento. Como resultado, el problema de la clasificación de múltiples etiquetas ha generado un interés significativo en los últimos años. Sin embargo, el reconocimiento de múltiples etiquetas es una tarea de reconocimiento de objetos mucho más difícil en comparación con el reconocimiento de una sola etiqueta. En este trabajo, estudiaremos el problema del reconocimiento de alimentos de múltiples etiquetas mediante el uso de algoritmos de aprendizaje profundo, específicamente redes neuronales convolucionales. Mostraremos cómo la redefinición de la función de pérdida y el aumento del conjunto de datos de entrenamiento pueden ayudar en el problema del reconocimiento de alimentos de múltiples etiquetas. Se presentará una validación extensa para mostrar las fortalezas y limitaciones del reconocimiento de alimentos de múltiples etiquetas.

Acknowledgements

Firstly, I am very thankful for all the support that the supervisor of this project, Dra Petia Radeva, has offered during this period of the work. I always wanted to do a final project on Computer Vision, and she suggested the Food Recognition problem which I am very interested in, so I decided to start my research on that topic. She has helped me a lot with the structure of the project and, after the early stages of research, to define some clear goals. The door of her office was always open whenever I needed or had a question about my research.

I also want to thank the support received from Bhalaji Nagarajan, doctoral student at UB specializing in the food recognition domain, who has helped me with the technical details of the implementations and has given suggestions to improve my results.

Talking about the personal aspect, I want to express my very profound gratitude to my family and friends for providing me with unfailing support and encouragement in overwhelming times.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we are going to introduce the project, starting with the context and the motivation, following with the goals and the planning and, finally we give the description of the contents.

1.1 Context

Very recently, COVID-19 has been looked as one of the most serious global pandemic. The World Health Organization (WHO [50]) has declared the COVID-19 outbreak to be a public health emergency of international concern. As of September 1st of 2020, over 23 million people around the world have been known to be infected. People of all ages can be infected. However, older people and people with pre-existing medical conditions (such as diabetes, heart disease, obesity and asthma) appear to be more vulnerable to becoming severely ill with the COVID-19 virus, which gives prime importance to better management of health conditions.

Looking at the statistics, according to the International Diabetes Federal [48], in 2019, approximately 463 million adults (20-79 years) were living with diabetes; and by 2045, this will rise to 700 million. It is notable that 1 in 2 (approx. 232 million) people with diabetes were undiagnosed and diabetes has been the cause of approx. 4.2 million deaths.

The amount of people who are overweight is also increasing rapidly, and as a result, these people are more prone to acquire chronic diseases such as respiratory disease, heart disease, etc. According to the World Health Organization [51], the worldwide prevalence of obesity nearly tripled between 1975 and 2016. In 2016, more than 1.9 billion adults aged 18 years and older were overweight. Moreover, 38 million children under the age of 5 were overweight or obese in 2019. The

principal cause of obesity and overweight is an energy imbalance between calories consumed and calories expended.

In both cases, Diabetes and Obesity can be prevented at some point with just changing the diet. So getting adequate nutrition every day with some physical activities are essential for our healthy well-being. In the last decade, diet is turning to be one of the most significant factors in human day-to-day life. The traditional way for people with some chronic diseases is to find a professional such as a dietitian or a nutritionist (who advise people on what to eat to lead a healthy lifestyle or achieve a specific health-related goal.) to help them with food selection and quantity intake. The healthcare professionals often ask their patients to fill in a food questionnaire where they note down all the food that they consume during the day. The process of answering the questionnaire is very long and boring. This gave birth to digital diaries. With the digital Diaries such as My Fitness Pal (`www.myfitnesspal.com`), people can use these apps to track their food intake more easily by logging the food on their mobile phones.

The process of noting down the list of food taken during the day is still not the easiest way. What happens when the user goes to a restaurant or when there is a different set of ingredients on a dish. It would be much easier to take a picture of the plate, with some algorithm behind, constructing the food list automatically. With automatic food recognition systems, we take a picture of what we consume during the day and let the app track our food intake automatically. The high demand in today's society has given rise to many apps such as "Mama Calorie" and "Bite IA", which aims to record and analyze the food intake automatically.

1.2 Motivation

Food image analysis includes several Computer Vision problems such as food detection, food recognition, food image segmentation, quantity estimation, etc. In this project, we are primarily interested in the food recognition problem.

Can we automatically recognize food? People do it relatively easy because all their life, they have been seeing and eating food which gives them innate ability to recognize food. However, from a computational point of view, the detection and classification of every instance of a dish in all its variants, shapes, and positions with a large number of images is not an easy task. Most methods proposed for automatically analyzing food images include the step of recognition. Food recognition is probably one of the most popular topics in the literature about automatic diet monitoring [11] [41].

Image-based food recognition has made substantial progress thanks to advances in deep learning in the past few years. But food recognition remains a difficult problem for a variety of reasons. The main problems that we must face on are:

- Complexity and variability of the data.
- Huge amounts of data to analyze.

We know that the diversity of types of food is enormous. Even within the same food category, there is significant diversity (intra-class variation). There are thousands of ways to take a photo of a single dish changing the angles (viewpoint variation), the brightness (illumination conditions), etc. So, it is hard to do a food item recognition.

Recently, Neural Networks have revolutionized the object recognition tasks [15] [7] [21]. However, to achieve a good recognition performance, we need extensive amounts of data because of the number of parameters to be tuned by a Neural network algorithm. The problem with deep learning is that its training starts with an unfavorable initial state. Then, it uses some gradient-based optimization algorithm to converge the network to an optimal solution, which might not necessarily be the global optimum. Hence, the process requires a lot of data.

Generally, food recognition algorithms can be divided into the two categories based on the nature of images: single-label and multi-label food recognition. Fig. 1.1 shows the difference between single-label food images and multi-label food images.

- Single-label food recognition targets food images with only one food item in an image.
- Multi-label food recognition and detection of food images analyze multiple food items in a single image.

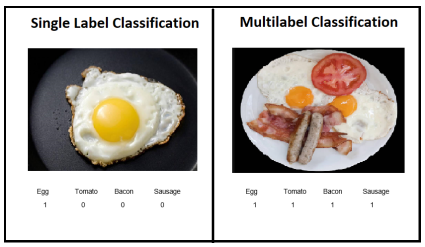

Figure 1.1: Single Label vs Multilabel

1.3 Goals

The main objective of this project is to explore Deep learning algorithms, in our case "Convolutional Neural Network" (explained in the chapter 3.2) for Multi-label Food Recognition Problem. More specifically, we want to find a model that results in high performance of multi-label food recognition on a multi-label dataset. We will therefore study in detail, the effectiveness of Convolutional neural networks in multi-label food recognition problems by training different deep learning models.

To study the main objective of this project, we will dive into: To study the main objective of this project, we will dive into:

1. Exploration of the multi-label food recognition problem.
2. Exploration of the dependence problem on the number of trained datasets.
3. Consideration of a new loss function to improve the classification
4. Realization of an extensive validation of the multi-label food recognition problem.

Why we choose Multi-label Food Recognition problem instead of Single-label Classification? As we all know, in the real world scenario, the proportion of dishes with more than one type of food is higher than single class dishes. In general we know that a single-label classification problem is simpler to solve than a multi-label one due to the use of a softmax layer that forces the algorithm to give a solution. Besides, we know that the multi-label annotations are more complicated than the single label annotations because multi-label annotations might have more annotator errors, confusion of classes, partial annotations, etc. For these reasons, it is more useful but challenging to focus on the Multi-label Classification Problem.

During the course of the memory, we will explain the necessary concepts and explore the classification of multiple labels using the Convolutional Neural Network.

1.4 Planning of the project

This project took about a year to complete. My first meeting with my supervisor was on 02/08/2019. Our planning is as follows:

- During the first months: the principal tasks are searching useful information, reading articles about Food Recognition, and learning necessary theories and technologies.

- During the next three months: Developing deep learning models for multi-label food recognition using Google Colab Pro.
- In the following months: trying to improve our model developed previously and solve the overfitting problem.
- In the last phase: the tasks are continuing with our experiments, cleaning codes, getting conclusions after comparing the result, and elaborating the project memory and defense presentation.

1.5 Organization of the memory

The rest of the memory is organized as follows:

- **Related works chapter:** discusses recent work in the area of Food recognition using deep learning.
- **Methodology chapter:** explains the concepts required to address the Convolutional Neural Networks.
- **Implementation of Multi-label Food Recognition chapter:** explains the concepts required and implementations to address the Multi-label Food Recognition problem.
- **Validation of Multi-label Food Recognition chapter:** exposes the results of all the experiments.
- **Conclusions and future lines chapter:** concludes the project with an analysis of the work as well as considers the future work.
- **Bibliography chapter:** websites, articles and literature consulted.

Chapter 2

Related work

In this chapter, we will present some previous work done in the field of Food Recognition. We will discuss the area of Single-label Food Recognition and Multi-label Food Recognition to get a notion of the tendencies in this field.

2.1 Single-label Food Recognition

The main application of Food analysis is to improve people's lives, more clearly, to improve people's diet. One of the main tasks of Food Analysis is Food Recognition. Most research in the food recognition area considers that each image sample contains only one class food. So, we can assume food recognition as an image classification problem. For all the existing works in the food recognition area, there are two different approaches:

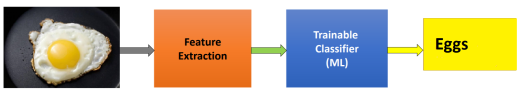

Figure 2.1: Classical Approach

Figure 2.2: Deep Learning Approach

2.1.1 Food Recognition Using Classical Approach

The seminal paper on food recognition done by Bossard et al. [22] proposed a significant work in the area of Food Recognition. They created a new food database called Food-101. This dataset contains 101 food categories, and for each class, there were 1000 images. They achieved a 58% accuracy on the test set. The steps that they followed were:

1. Firstly, they extracted the color SURF [3] features for super-pixels on each image.

2. Then, they clustered all the super-pixels into groups using Random Forest based on their respective Fisher vector [38] encoded feature vectors to obtain discriminative components across all the images.

3. Finally, they trained a binary SVM using the top N mined components of each category.

Joutou et al. [46] proposed a Multiple Kernel Learning (MKL) method combining different features: SIFT-based bag-features, color histogram, and Gabor Texture features (it is a linear filter used for texture analysis). They created a new Japanese food dataset with 50 classes (100 images of each one). The accuracy of their new dataset was 61.3%. A follow-up study by Hoashi et al. [17] achieved an accuracy rate of 62.5% using the same method mentioned previously on an extended dataset of 85 classes.

Chen et al. [10] created the Pittsburgh food database which contained 101 classes of American fast food images taken in a controlled environment. The researchers benchmarked the dataset using two standard approaches, color histogram and bag of SIFT features in conjunction with a discriminative classifier. Yang et al. [57] proposed a new representation for food items that calculates pairwise statistics between local features computed over a soft pixel-level segmentation of the image sample into eight ingredient types. They tested on the subset of Pittsburgh dataset [10] and they achieved an accuracy rate of 28.2%.

Bettadapura et al. [4] proposed to leverage the context of where the picture was taken (in their case, with additional information about the restaurant). They combined 6-feature descriptors (2 color-based and 4 SIFT-based) and SMK-MKL Sequential Minimal Optimization to train an SVM classifier. They tested on a dataset consisting of 3750 food images of 75 categories (50 images per category) and achieved an accuracy of 63.33%.

2.1.2 Food Recognition Using Deep Learning Approach

Recently, Convolutional Neural Networks [2] have been widely used for feature extraction in object recognition and have achieved much better performance than the classical ones. This fact can be observed also for the food recognition problem.

Kagaya et al. [22] trained convolutional neural network for food recognition and also non-food detection. They created a food database of 170,000 images containing 10 popular food items. They used 6-fold cross-validation to get the optimal hyperparameters. To test the performance they used their own dataset and they showed that CNN outperformed all the other baseline classical approaches by achieving an accuracy rate of 73.7% for 10 classes.

Kawano et al. [24] proposed the idea of using CNN. In this case, AlexNet proposed by Krizhevsky et al. [26] was used as a feature extractor. The authors achieved an accuracy of 72.3% on the UEC-FOOD100 dataset (contains 100 classes of Japanese food: `http://foodcam.mobi/dataset.html`).

Yanai et al. [56] examined the effectiveness of the Alexnet network (which was pretrained on 2000 categories of the ImageNet dataset including 1000 food-related categories) for food recognition task by fine-tuned the AlexNet. They achieved the best results on public food datasets so far (with a top-1 accuracy of 78.8% for UEC-FOOD100 [28] dataset and 67.6% for UEC-FOOD256 [23]).

Wu et al. [54] proposed a visual food recognition framework that integrates the inherent semantic relationships among fine-grained classes.

Martinel et al. [27] proposed a new CNN structure that combines slice convolution block to capture specific information with extracted visual features from the Wide Residual Networks (WRNs), proposed by Sergey et al. [58]). They achieved the highest Top-1% and Top-5% on Food-101 (90.27%, 98.71%), UECFood-256 [23] (83.15%, 95.45%), and UECFood-100 [28] (89.58%, 99.23%).

Most recently, Myers et al. [31] proposed the Im2Calories system for food recognition using CNN-based approaches. They used the GoogleLeNet [44] as a base architecture and fine-tuned the pre-trained model on Food101. They achieved top-1 accuracy rate of 79% on the Food-101 test set.

2.2 Multi-label Food Recognition

In real-world scenarios, the proportion of the multi-food images is higher than one item food images. Marsuda et al. [30] proposed the first work of multiple food recognition from food images. They proposed a new method with two steps:

1. Detect several candidate regions by fusing outputs of region detectors. In the paper they used four kinds of candidate region detection method:
    - Whole image: this method assumes that one image contains only one food item.
    - Deformable part model method (DPM): this model was proposed by Felzenszwalb et al. [14]. The DPW contains a global root filter and several part models. Each part model specifies a spatial model (defines allowed placements for a part relative to a detection window and its cost). Both root and part filters are scored by computing the dot product between a set of weights and HoG features [12].
    - Circle detector: detects regions of dishes by extracting circular contours (using Canny Edge Detector [9]) from an image.
    - JSEG region segmentation: divide an image into several pieces of regions using JSEG Algorithm (proposed by Deng et al. [55]) as a region segmentation algorithm.
2. Apply a feature-fusion-based food recognition method for bounding boxes of the candidate regions with various kinds of visual features, such as histogram of oriented gradient (HoG [12]).

They used trained SVM classifiers by multiple kernel to compute the evaluation values of each region belonging to all the given classes. These values were sorted and the system only outputs the top 10 classes for the user to choose.

Matsuda et al. [29] proposed a method to recognize multiple-food meal images which include multiple food items considering co-occurrence statistics of food items.

As representative work, Aguilar et al. [1] proposed a semantic food detection framework, which consists of three parts, namely food segmentation, food detection, and semantic food detection. Food segmentation uses the fully CNNs to produce the binary image and then adopts the Moore-Neighbor tracing algorithm to conduct boundary extraction. Food detection consists of retraining YOLOv2

(Redmon et al. [35]). Semantic food detection removes errors from object detection by combining results of segmentation and detection to obtain final food detection results.

Zhang et al. [60] presented a new mobile food recognition system which is capable to do multiple-food recognition of 15 food categories on the phone. The photo taken by users will be uploaded to the server. On the server side, the food image was firstly segmented into possible salient regions, and these regions were further grouped based on the similarity of their color, HOG and SIFT feature vectors. Then they trained a linear multiple-class SVM classifier for each class using the Fisher vector encoded feature vectors (including SIFT and color features) of salient regions. They have achieved an accuracy rate of 85% in their experiments.

In the Im2Calories [31] system mentioned earlier, the researchers proposed a new multi-label dataset (known as Food-201) which contained 201 food items. They developed a new multi-label classifier by replacing the last softmax layer to a multiple-label classifier named logistic nodes. An average top1 accuracy rate of 50% were achieved.

There is also some work done on the area of multi-label ingredient recognition. For example, Pan et al. [32] proposed a deep learning method for automatic multi-class classification of food ingredients.

Zhou et al. [61] proposed a novel approach to exploit ingredient and label relationships through bipartite-graph labels, and then combined that with a Convolutional neural network in a unified framework for multi-label ingredient recognition and dish recognition.

Marc Bolaños et al. [5] proposed a deep multi-ingredients recognition method. They used Inception-V3 [45] and ResNet-50 [16] as basic deep architectures. For these models, they modified the last layer to apply multi-label classification over N possible outputs to predict the list of ingredients in a food image.

Chapter 3

Methodology

In this chapter, we will explain the concepts required to address the Convolutional Neural networks. And then, we will present some relevant explanations for better understanding.

3.1 Deep Learning

Artificial Intelligence (AI) is a general field that encompasses Machine Learning (ML) and deep learning. We can define it as an academic discipline devoted to the theory and development of computer systems able to perform tasks requiring human intelligence. And referring to machine learning, it is a subfield of AI that uses algorithms to learn from data and enables machines to improve with experience.

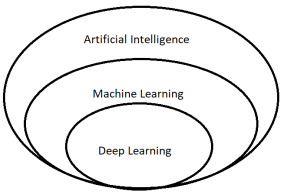

Figure 3.1: Artificial Intelligence, Machine Learning and Deep Learning

Deep Learning (DL) is a subfield of ML algorithms that permits software to train itself to perform tasks, such as image recognition, by exposing multilayered neural networks to cast amounts of data. Most modern DL models base on artificial neural networks, specifically, Convolutional Neural Networks (explained in section 3.2).

3.2 Convolutional Neural Networks

The Convolutional Neural Networks are a type of Artificial Neural Networks (ANN: based on neural networks that make up the nervous system of the human being), which have become a research focus in the field of image analysis and recognition.

Convolutional neural networks are very similar to ordinary neural networks: they are composed of neurons with learnable weights and biases. Each neuron receives some input and then performs dot product. The entire neural network still represents a differential score function: mapping from the original image pixel to the class score. In the last layer (the fully connected layer) there is also a loss function (explained in the section 3.2.4) that assures the optimization of the model in terms of training from a pre-annotated training dataset.

3.2.1 Layers

We can say that a convolutional neural network (also known as convnet), consists of an input, multiple hidden layers, and an output layer. And importantly, it takes as input tensors of shape (image Height, image width, image channels).

The hidden layers of a convnet typically consist of a series of convolutional layers. The activation function is commonly a ReLu layer, and subsequentially it follows by additional convolutions such as pooling layers, fully connected layers, and normalization layers.

Convolutional Layer

A Convolutional Layer: consists of a set of learnable filters. Every filter is spatially small in width and heights. We apply the filters to the original image or other feature maps in convnets.

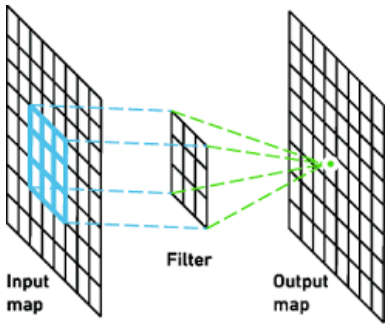

Figure 3.2: Convolutional Layer

The convolution layers learn the local pattern. So, this key characteristic gives convnets two interesting properties:

- The patterns that they learn are translation invariant: after learning a certain pattern in the top right corner of a picture, a convnet can recognize it anywhere.

- They can learn spatial hierarchies of pattern: a first convolution layer will learn small ones such as edges, a second convolution layer will learn larger ones made of the features of the first layers, and so on.

Pooling Layer

The pooling Layer: also known as a subsampling layer, is used to reduce the dimensions of the feature maps. So, it reduces the number of parameters to learn and the amount of computation performed in the network.

These layers are needed because the number of features obtained from a convolution layer can be very high. For example, an RGB image convoluted with a mask of size $HxWxC$ would have $HxWxC$ (Number of Filters) features and would increase significantly the output of the convolution. Training on such a high number of features would result in problems like over-fitting.

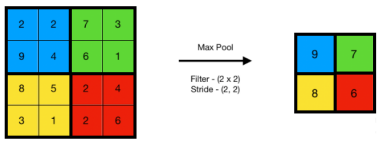

Figure 3.3: Max Pooling Layer

Fully-connected layer

The fully-connected layer: uses the feature information to reach a final class label. Every neuron in one layer connects to every neuron in another layer.

The nodes in the last layer of fully-connected layers have a specific activation function that will generate a probabilistic result for each label. In the case of a multi-label classification problem, the activation function is generally a Sigmoid function.

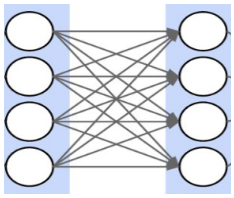

Figure 3.4: Fully Connected Layer

Activation Layer

The activation layer: is an activation function that decides the final value of a neuron. So we can say that it is a function used in a layer of a neural network to define the output of that node given as input the result of applying all previous layers. There are numerous activation functions. We will only explain some of them used in this project:

- **ReLu**: is the abbreviation of the rectified linear unit, which removes negative values from an activation map by setting them to zero. It applies the following non-saturating activation function

$$ReLu(x) = max(0, max) \tag{3.1}$$

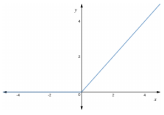

Figure 3.5: ReLu Function

- **Sigmoid**: limits the output to a range between 0 and 1. This function is used in the prediction of probabilities. The formula is as follows:

$$S(x) = \frac{1}{1 + e^{-x}} \tag{3.2}$$

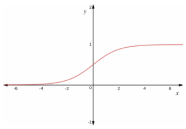

Figure 3.6: Sigmoid function

3.2.2 Optimizers

In the learning of the model, we also need to specify optimizers that define how the network will be updated based on the loss function. We will now explain the ones that we will use in our project: Stochastic Gradient Descent (SGD), Adaptive Moment Estimation (Adam), and AdaDelta.

- **Stochastic Gradient Descent**: is an iterative optimization method based on Gradient Descent. We can define the gradient descent like a first-order optimization algorithm used to find a local minimum of a function, so all we have to do is find all the points where the derivative goes to 0.

  To find a local minimum using gradient descent, we have to take steps (known as learning rate) proportional to the negative of the gradient of the function at the current point. In this case, it uses the entire dataset to calculate the gradient.

  We can define Stochastic Gradient Descent as a stochastic approximation of Gradient descent, where instead of using the actual gradient, it uses an estimated gradient calculated from a randomly selected subset of the data.

- **Adaptive Moment Estimation (Adam)**: was published for the first time in 2014 by Diederik P. Kingma[25]. It is a first-order gradient-based optimization algorithm of stochastic objective functions. It combines the advantages of the following two extensions of stochastic gradient descent:

  – Adaptive Gradient Algorithm (AdaGrad): it maintains a per-parameter learning rate instead of a fixed value.
  – Root Mean Square Propagation (RMSProp): it maintains a per-parameter leaning rate that is adapted based on the average of the last magnitudes of the gradients for the weight.

  This method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients.

  Moreover, it calculates an exponential moving average of the gradient and squared gradient, and the parameters $\beta 1$ and $\beta 2$ control the decay rates of these averages.

- **AdaDelta**: was proposed for the first time in 2012 by Matthew D.Zeiler[59]. It is an extension of Adagrad that adapts learning rates based on a moving window of gradient updates.

Adagrad has the following two drawbacks:

1. It has a continually decaying learning rate throughout the training.
2. It requires a manual selection of the global leaning rate.

With these two drawbacks in mind, Adadelta implements two new ideas:

1. Accumulates the sum of squared gradients over a restricted time window rather than overall time.
2. Corrects the mismatch in units that exist in most gradient descent based algorithms.

3.2.3 Normalization

Generally, the datasets used in deep learning are massive, so the input of the network can vary significantly. For this reason, we usually need some techniques like data normalization and batch normalization.

We can define normalization as a method that makes different samples look more similar to each other in the machine learning model, which helps the model learn and generalize well to new data.

There are different forms of data normalization. The most common one is: centering the data on 0 by subtracting the mean from the data, and giving the data a unit standard deviation by dividing the data by its standard deviation.

In 2015, Ioffe and Szegedy[20] introduced the new concept of "Batch normalization". This layer can normalize data during the training. It works by maintaining an exponential moving average of the batch-wise mean and variance of the data during training. Many architectures use this new layer, such as ResNet50 [16].

3.2.4 Loss Function

We can define the loss function as the quantity that will be minimized during training. It represents a measure of success for the task at hand, in other words, how well are the classifiers doing the recognition task.

Choosing the right loss function for the right problem is extremely important: the network will try everything to minimize the loss; so if the loss does not fully correlate with success for the task at hand, sometimes the network will end up doing unexpected things.

It exists different loss functions and the chose of the loss function depends on the learning problem. Now, we will explain some of them:

- **Softmax Loss:** is a commonly used loss function which is essentially a combination of multinomial logistic loss and softmax.

  Given a training set $\{(x^{(i)}, y^{(i)}; i \subseteq 1, ..., N, y^{(i)} \subseteq 1, ..., C\}$, where $x^{(i)}$ is the i-th input image patch, and $y^{(i)}$ is its target class among the C classes. The prediction of j-th class for i-th input is computed with softmax function:

$$p_j^{(i)} = \frac{e^{z_j^{(i)}}}{\sum_{l=2}^{C} e^{(z_l)^i}} \tag{3.3}$$

  The softmax loss is defined as:

$$\mathcal{L}_{softmax} = -\frac{1}{N}[\sum_{i=1}^{N}\sum_{j=1}^{N} 1y^i = j \log{(p_j)^i}] \tag{3.4}$$

- **Binary cross-entropy:** this loss is commonly used for multi-class problems. It calculates a loss between two distributions:

$$L(p, q) = -\sum_{i=0}^{n} pi \ln qi \tag{3.5}$$

  where p is the target distribution and q is the predicted distribution(computed by the neural network).

Chapter 4

Implementation of Multi-label Food recognition

In this chapter, we will introduce some useful concepts and implementations to address the Multi-label Food Recognition Problem. We will first explain the difference between single-label recognition and multi-label recognition. Then, we will dive into the common techniques used in the multi-label recognition problem.

4.1 Single-label vs Multi-label recognition

Multi-label image classification is a classification task where each input sample can be assigned multiple labels. The number of labels per image is usually variable. And for a single-label classification task, each sample can only be labeled as one class.

Besides the challenges shared with single-label image classification (e.g., large intra-class variation caused by viewpoint, scale, occlusion, illumination), multi-label classification is more difficult because both the input images and output label spaces are more complex. Multi-label datasets usually have many features that do not exist in single-label datasets such as high dimensionality, unbalanced data, and the correlation between labels.

Furthermore, collecting a multi-label dataset is more difficult and less scalable than collecting a single-label dataset, because collecting a consistent and exhaustive list of labels for every image requires significant effort.

4.1.1 Main difference

To compute the loss we need two inputs, the true labels for the images and the predicted labels. Cross-entropy is the default loss function to use for multi-class classification problems. We understand cross-entropy as the difference between two probability distributions for a given variable or set of events. The cross-entropy $H(p, q)$ of the distribution "true labels"(q) relative to a distribution "predicted labels"(p) over a given set is defined as:

$$H(p, q) = -Ep[\log q] \tag{4.1}$$

where $Ep[*]$ is the expected value operator with respect to the distribution p.

In the single-label classification model, we use a softmax activation function in the last layer. For each image, we want to maximize the probability of a single class. As the probability of one class increases, the probability of the other class decreases. So, we can say that the probability of each class is dependent on the other classes.

In our case, we focus on the multi-label food classification problem. The input can be classified into one or more classes. Using the softmax as the activation function will not be appropriate because we want the probabilities to be independent of each other to avoid obtaining, 0, 1, or many recognized classes. Instead, we can use the sigmoid activation function. This will predict the probability of each class independently. So by using sigmoid, we will turn a multi-label classification problem into several binary classification problems. Since we have converted it into several binary classification problems, we will use the binary_cross-entropy as our loss function.

In Keras, there are different types of losses available (all losses are available via a class handle and via a function handle) such as probabilistic losses and logistic losses. We have used the binary_cross-entropy loss.

4.2 Transfer Learning to the Multi-label Food Recognition

A common approach to deep learning on a small dataset is to use a pre-trained model on a bigger dataset although from a different domain, know as fine-tuning. This process is called Transfer learning and presents one of the main advantages of deep learning. In this context, we get a pre-trained network as a saved network trained previously on a different big dataset, typically on a large-scale generic image-classification task.

The majority of the pre-trained networks are trained on a subset of the ImageNet database, which is used in the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC [37]). These networks have been trained on more than a million images and can classify images into 1000 object categories.

The main reasons that we opted the pre-trained model were:

- By using a pre-trained model, we will save more training time and computation cost than train a new model from scratch.
- All pre-trained models have been trained with more data: in our case, we have limited data.
- Better performance of neural networks (in most cases).

There are two ways to use a pre-trained network: feature extraction and fine-tuning. We will cover the technique used in our project: feature extraction.

4.3 CNN Learning with Partial Labels

We also need to address the concept of partially-labeled multi-class classification. A partially-labeled food recognition dataset contains image samples where some of the images are fully labeled while other images are only partially labeled.

During the development of this project, we have seen the necessity of collecting new image samples for the Food-201 dataset. Due to the complexity of multi-label annotations, we only have single-labeled them. Training convnets by treating missing labels as negatives [6] [42] results in a significant performance drop as many ground-truth positive labels are falsely labeled [19].

4.3.1 New loss function: partial Binary Cross-Entropy

The most popular loss function to train a model for multi-label classification is binary cross-entropy (BCE). To be independent of the number of categories, the BCE loss is normalized by the number of classes.

This becomes a drawback for partially labeled data because the back-propagated gradient becomes small. So, when partial labeling is considered, the proportion of false negatives will increase by adding single-labeled images.

In order to solve this problem we have opted for a new loss function, known as Binary Cross-Entropy (BCE) for partial labels (partial BCE) defined in [13].

The partial BCE normalizes the loss function by the proportion of known labels. Let us consider the following Notation:

- Number of classes: C
- Number of training samples: N
- Training data: $D = (I^1, y^1), ..., (I^N, y^N)$ where I^i is the i^{th} image and $y^i = [y_1^i, ...y_C^i] \in \gamma \subseteq \{-1, 0, 1\}^C$ where {-1 = absent, 0 = present and -1 = unknown}
- Proportion of known labels: $P_y \subseteq [0, 1]$

The new loss function is defined as follows:

$$\ell(x, y) = \frac{g(P_y)}{C} \sum_{c=1}^{C} \left[1_{[y_c=1]} \log \left(\frac{1}{1 + \exp(-x_c)} \right) + 1_{[y_c=-1]} \log \left(\frac{\exp(-x_c)}{1 + \exp(-x_c)} \right) \right] \tag{4.2}$$

As you can observe in the equation above, the partial-BCE loss ignores the categories for unknown labels (yc = 0). In the standard BCE loss, the normalization function is g(py) = 1. Unlike the standard BCE, the partial-BCE gives the same importance to each example independent of the number of known labels, which is useful when the proportion of labels per image is not fixed. This loss adapts itself to the proportion of known labels.

The function g normalizes the loss function with respect to the label proportion. They proposed the normalization function below which has the same behavior as the BCE loss when all the labels are present i.e. g(1) = 1.

$$g(P_y) = \alpha P_y^{\gamma} + \beta \tag{4.3}$$

According to the paper mentioned previously, for the new loss function, they introduced several hyper-parameters to their loss function and the default values are: $\alpha = 4.45$, $\beta = 5.45$ and $\gamma = 1$.

4.4 Feature Extraction and Architecture to Analyse

Convolutional neural networks used for image classification comprise two parts: a series of pooling and convolution layers as the first part, known as a convolutional base, and a densely connected classifier as a second part.

Feature extraction consists of taking the convolutional base of the previous network, running the new data through it, and training a new classifier on top of the output. We only reuse the convolutional base because information learned by the convolutional base are likely to be more generic.

In this project, we will extend the convolutional base model by adding dense layers on top and running the whole model end-to-end on the input data. This technique is slow and expensive but allows us to use data augmentation.

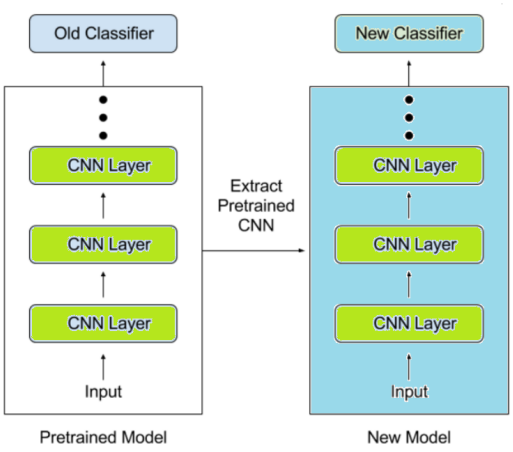

Figure 4.1: Feature Extraction

```
con_base = efn.EfficientNetB5(weights='imagenet', include_top=False,input_shape=(224, 224, 3))

x = con_base.output
x = SpatialDropout2D(0.5)(x)
x = GlobalAveragePooling2D()(x)
x = Dense(2048, activation='relu')(x)
predictions= Dense(201, activation='sigmoid')(x)

model= Model(inputs=con_base.input, outputs=predictions)
```

Figure 4.2: Adding layers to convolutional base model

4.4.1 Architectures used

Different learning models are made available in the literature. Referring to the image classification, the VGG-16 [39](which was trained for the ImageNet “Large Visual Recognition Challenge") is one of the most popular pre-trained models for that. Also, the ResNet50 [16] Inceptionv3 [45] and EfficientNet [47]. We can use these models for prediction, feature extraction, and fine-tuning.

We have used the following models as the convolutional base models in our experiments:

- **ResNet50 [16]**: this model was proposed by Kaiming He et al. who was the winner of ILSVRC 2015. The model allowed us to train extremely deep neural networks with 150+ layers successfully. It introduced the concept of skip connection. On the model there is a heavy use of batch normalization. And there is no fully connected layer at the end of the architecture.

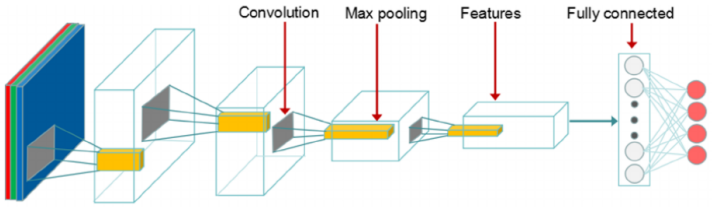

Figure 4.3: Resnet50 Architecture [33]

- **InceptionResNetV2 [43]**: this model combines the Inception [44] architecture, with residual connections. In the InceptionResnet block, multiple sized convolutional filters are combined with residual connections.

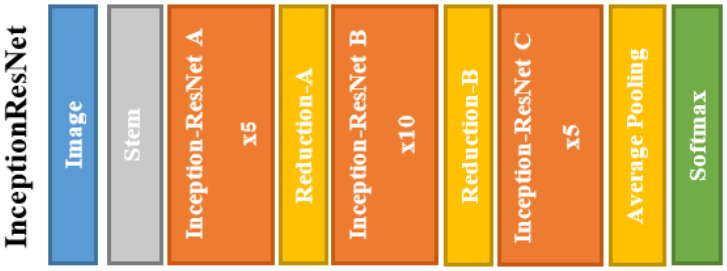

Figure 4.4: InceptionResnetV2 Core Architecture [43]

- **DenseNet169 [18]**:this network connects all layers in such a way each layer obtains additional inputs from all preceding layers and passes its own feature-maps to all subsequent layers.

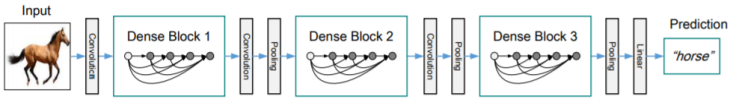

Figure 4.5: A deep DenseNet with three dense blocks.[18]

- **EfficientNetB5 [47] and EfficientNetB7 [47]**: Efficient model was proposed by Mingxing Tan et al. They proposed a new scaling method that uniformly scales all dimensions of depth, width and resolution of the network. They used the neural architecture search to design a new baseline network and scaled it up to obtain the EfficientNet.

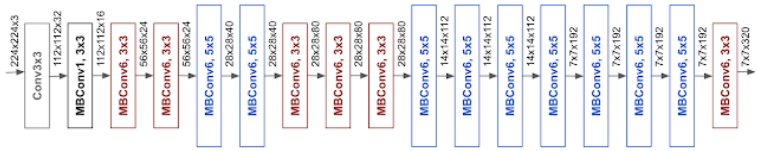

Figure 4.6: Architecture of EfficientNetB0

In Keras, there is a module called Keras applications [49]. This module provides pre-trained models (all pretrained on the ImageNet dataset) like Resnet50 [16] (also it variants), EfficientNet [47] (also it variants) InceptionRestNetV2 [43] for deep neural networks.

4.5 Over-fitting Problem

The over-fitting problem is a common problem in machine learning. It is a modeling error that occurs when our neural network fits closely with the training set, but it might not generalizes and makes predictions for new data correctly. In other words, we can say that the training accuracy is higher that validation/test accuracy. So what can we do to reduce the over-fitting problem?

Generally talking, the problem of overfitting can reduce by adding new training data. But in the case of multi-label image recognition, this process is not easy. The label annotations and image collections are very challenging due to its complexity. So, instead of collecting new image samples, we can use techniques to generate images from existing ones.

4.5.1 Data Augmentation

The data augmentation technique consists of generating more training data from the existing ones by augmenting images via some transformations. The goal is that at training time, our model will never see the same picture twice. Given that our network is seeing new data slightly modified versions of the input data, it can learn more robust features.

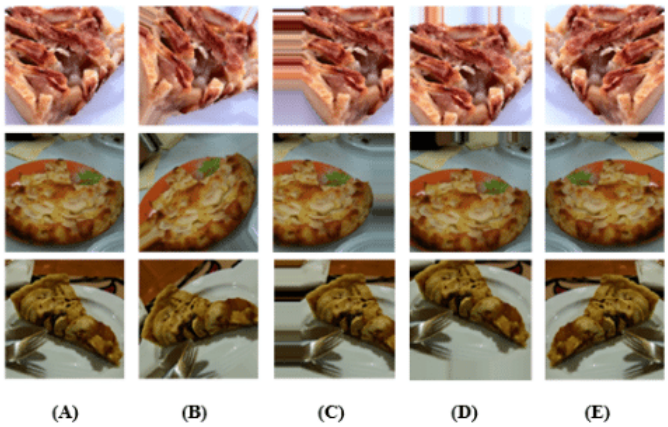

Figure 4.7: Example of the data augmentation images: (A) original, (B) rotation, (C) width shift, (D) height shift, and (E) horizontal flip images. [34]

In our case, we have used ImageDataGenerator from Keras and Albumentations library from Github[8]:

- **ImageDataGenerator:** In Keras, we can do random transformations using ImageDataGenerator. As you can see in the following figure 4.8, we can apply different geometric changes such as rescale, random horizontal flips, etc.

```
train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_dataframe(
        dataframe=train_df,
        directory= train_data_dir,
        x_col="filename",
        y_col=lista,
        target_size=(224, 224),
        batch_size=12,
        class_mode='raw')

validation_generator = test_datagen.flow_from_dataframe(
        dataframe=valid_df,
        directory=validation_data_dir,
        x_col="filename",
        y_col= lista,
        target_size=(224, 224),
        batch_size=12,
        class_mode='raw')
```

Figure 4.8: ImageDataGenerator

- **Albumentations:** is a flexible image augmentation library based on NumPy, OpenCV, and Imgaug and is written in Python. This library is widely used

in deep learning search, machine learning competitions, and open source projects. We have chosen this library due to its numerous set of transformations.

Referring to our implementations, we have applied the following transformations in some of our experiments (you can find the explanation of each one in their official Github webpage):

```
AUGMENTATIONS = albumentations.Compose([albumentations.HorizontalFlip(p=0.5),
                                        albumentations.Rotate( p=0.5),
                                        albumentations.Blur( p=0.5),
                                        albumentations.RGBShift(r_shift_limit=20, g_shift_limit=20, b_shift_limit=20, p=0.5),
                                        albumentations.HueSaturationValue(hue_shift_limit=5, sat_shift_limit=20,  val_shift_limit=10, p=.9),
                                        albumentations.RandomBrightness(p=0.5),
                                        albumentations.ShiftScaleRotate( p=0.5)])
```

Figure 4.9: Some transformations of Albumentations

Notation

To distinguish between both techniques, we use the following notations:

- Type 1: data augmentation using ImageDataGenerator.
- Type 2: data augmentation using Albumentations.
- Type 3: data augmentation using both techniques.

4.5.2 Regularization

The common way to avoid over-fitting is to put constraints on the complexity of a network by forcing its weights to take only small values, which makes the distribution of weight values more regular. We called this as the weight regularization, and it is done by adding to the loss function of the network a cost associated with having large weights. The cost comes in two types:

- **L1 regularization:** the cost added is proportional to the absolute value of the weight coefficient(the L1 norm of the weights: W)

$$Cost = Loss_Function + \lambda \sum_{j=0}^{M} |W_j| \tag{4.4}$$

- **L2 regularization:** the cost added is proportional to the square of the value of the weight coefficients.

$$Cost = Loss_Function + \lambda \sum_{j=0}^{M} W_j^2 \tag{4.5}$$

Dropout

Dropout was presented for the first time in 2014 by Nitish Srivastava et al. [40]. It is one of the most effective and most commonly used regularization techniques for neural networks. Dropout, applied to a layer, consists of randomly dropping out some output features of that layer during training. The dropout rate is the fraction of the features that are drop out and it's usually set between 0.2 and 0.5.

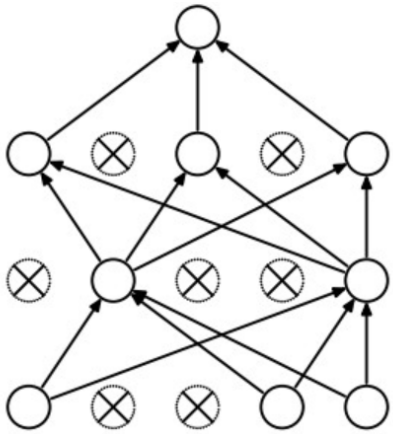

Figure 4.10: Neural Network with Dropout

We have used 'SpatialDropout2D' [52] from Keras that performs the same function as Dropout. However, it drops entire 2D feature maps instead of individual elements.

4.6 Visualization of Food Recognition

It is often said that deep-learning models are "black boxes" because the learning representations are difficult to extract in a human-readable form.

Fortunately, Convolutional Neural Networks have inputs (images) that are visually interpretable by humans so we have various techniques for understanding what do they learn, how they work, and why they work in a given manner while for other deep neural network architectures visualizations are very more difficult. In this project we used heatmaps as a technique to visualize the results:

- **Visualizing heatmaps of class activation in an image:** very useful for understanding which parts of an image were identified as belonging to a given class.

4.6.1 Visualizing heatmaps of class activation in an image

This technique is useful for understanding which parts of a given image led a convnet to its final classification decision, and also for debugging the decision process of a convnet.

This general category of techniques is called class activation map visualization. It consists of producing heatmaps of class activation over input images. A heatmap is a 2D grid of scores associated with a specific output class, computed for every location in any input image indicating how important it is for that class.

The activation heatmaps may differ for different layers in the network, as all layers view the input image differently, creating a unique abstraction of image based on their filters. In this project, we have focused on the final layer of model, as the class prediction label will largely depend on it.

We computed the heatmaps by using the one describe in “Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization" [36].

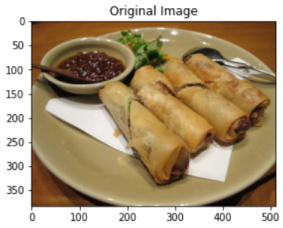

Figure 4.11: Original Image

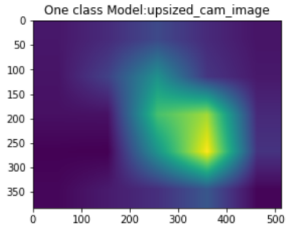

Figure 4.12: Heatmap

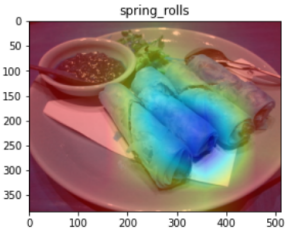

Figure 4.13: Final Image

4.6.2 Visualizing Feature Extraction

In machine learning, feature extraction starts from an initial set of measured data(images) and builds derived values (features) intended to be informative, facilitating the subsequent learning and leading to better human understandings. We have followed the steps below:

1. Generate features for each class from activation maps.
    - Create the activation maps.
    - Convert the activation map to a binary mask and multiply with the original image.

- Use the masked image on the classifier and extract features from the last but one Full-Connected layer.

2. Apply visualization techniques as PCA and t-SNE (see bellow) to visualize the features.

Principal Component Analysis

Principal component analysis (known as PCA) is a method that projects a higher-dimensional space into a lower-dimensional space by maximizing the variance of each dimension. It is one of the most important methods of dimensionality reduction for visualizing data.

For visualization, the first and second component can be plotted against each other to obtain a two-dimensional representation of the data that captures most of the variance (most of the relevant information), useful to analyze and interpret the structure of a data set.

T-distributed stochastic neighbor embedding

T-distributed stochastic neighbor embedding, known as t-SNE, is a nonlinear dimensionality reduction algorithm that uses the local relationships between points to create a low-dimensional mapping. It was developed by Laurens van der Maaten and Geoffrey Hinton in 2008 [53].

The algorithm creates a probability distribution using the Gaussian distribution that defines the relationships between the points in high-dimensional space and then uses the Student t-distribution to recreate the probability distribution in low-dimensional space. This technique finds clusters in data by making sure that an embedding preserves the meaning in the data.

Embedding Projector

Embedding Projector is a web application for interactive visualization and analysis of high-dimensional data. We used the standalone version at `projector.tensorflow.org`, where users can visualize their high-dimensional data without the need to install and run TensorFlow. The Embedding Projector offers three methods of reducing the dimensionality of a dataset:

- **PCA:** the Embedding Projector can computes the top 10 principal components.

- **T-SNE:** the Embedding Projector offers two-dimensional and three-dimensional t-SNE views.
- **Custom:** users can construct specialized linear projections based on text searches.

Chapter 5

Validation of Multi-label Food Recognition

In this chapter, we begin discussing the experimental part of this project. First, we will introduce our experiment settings. Then we will describe our experiments. Finally, we will dive into the results of each of them.

5.1 Experimental Settings

In this section we will discuss about the environment, dataset used and evaluation criteria.

5.1.1 Environment

We have implemented all the experiments using Keras 2.0 with TensorFlow 1.0 as the back-end and have run all of the experiments on Google Colab Pro.

Keras is a deep-learning framework written in Python that provides a convenient way to create and train a deep-learning model. We have picked up Keras as it has a user-friendly API that makes it easy to prototype deep-learning models and it has built-in support for convolutional networks.

TensorFlow is an end-to-end open source platform for machine learning by Google. It is based on data flow graphs where each edge is a multidimensional array, and each node represents an operation with this array. We have used TensorFlow backend using Keras for our model creation.

Google Colab Pro is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud. We have used Google Colab Pro instead of

Google Colab (free version one) because Colab Pro provides faster GPUs, longer run-times, and more memory.

5.1.2 Dataset

The dataset Food-201 [31] is provided with a total size of 2.58GB (with 50374 images and annotation files). We have split the whole dataset into training, validation, and testing sets. There are 31718 images for training, 15132 images for validation, and 3524 images for testing. (you can find for more detail about the dataset in 6).

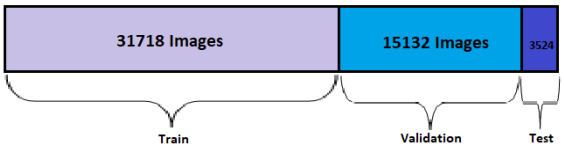

Figure 5.1: Visualization of the splits

The dataset represents 201 different food classes. For each image sample, remind that it might contain more than one category (multi-labeled).

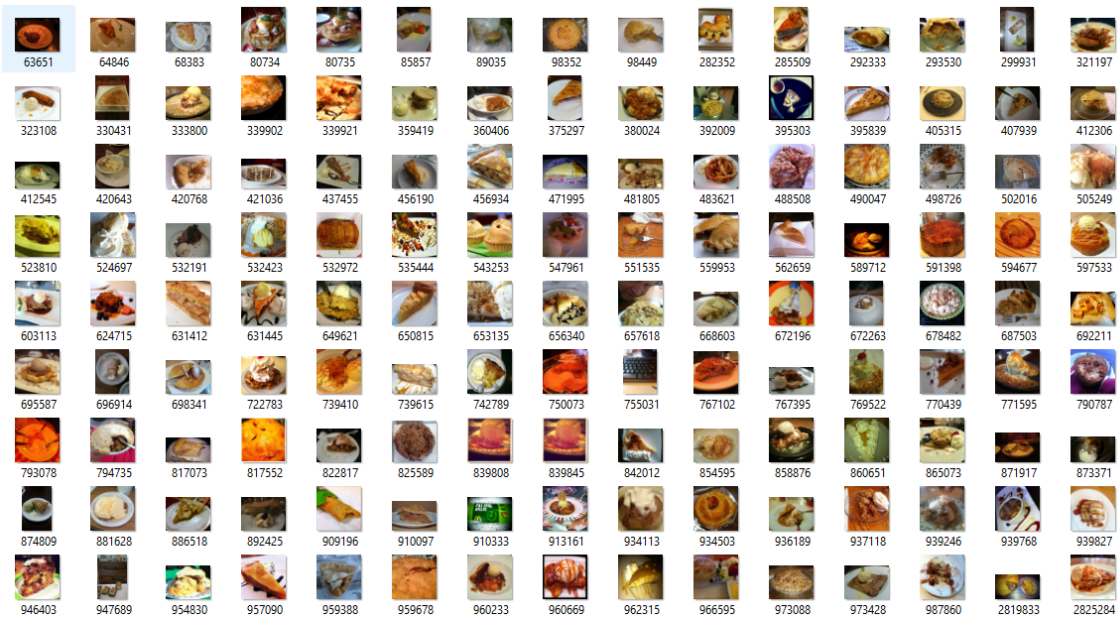

Figure 5.2: Food-201 dataset

5.1.3 Evaluation Criteria

To evaluate the performances of our model, we use the following metrics:

- **Micro F1-score:** we can interpret as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. In this case, it shows the global performance:

$$F1 = 2 * \frac{(Precision * Recall)}{(Precision + Recall)} \tag{5.1}$$

- **Per-class precision:** is defined as the number of true positives over the number of true positives plus the number of false positives:

$$Precision = \frac{True\ Positives}{(True\ Positives + False\ Positives)} \tag{5.2}$$

- **Per-class recall:** is defined as the number of true positives over the number of true positives plus the number of false negatives:

$$Recall = \frac{True\ Positives}{(True\ Positives + False\ Negatives)} \tag{5.3}$$

- **Per-class F1-score:** shows the F1-score of each class.

Another important thing that we want to point out is that in our implementations, we have adapted the F1-score function to evaluate the model in each epoch because it is not available in Keras.

```
def f1(y_true, y_pred):
    def recall(y_true, y_pred):
        """Recall metric: only computes a batch-wise average of recall.
        Computes the recall, a metric for multi-label classification of
        how many relevant items are selected.
        """
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
        recall = true_positives / (possible_positives + K.epsilon())
        return recall

    def precision(y_true, y_pred):
        """Precision metric:only computes a batch-wise average of precision.
        Computes the precision, a metric for multi-label classification of
        how many selected items are relevant.
        """
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
        precision = true_positives / (predicted_positives + K.epsilon())
        return precision

    precision = precision(y_true, y_pred)
    recall = recall(y_true, y_pred)

    return 2*((precision*recall)/(precision+recall+K.epsilon()))
```

Figure 5.3: Method of F1-score

The reason for this selection is that in the Food-201 dataset, there are imbalanced classes and an imbalanced dataset usually causes the convnet model to be more biased towards the majority class and poor classification of the minority class. Because of that, we have decided to use F1-score instead of accuracy.

In the library Scikit-learn, there is a function called "Classification Report" which shows the representation of the main classification metrics per-class basis.

5.2 Find the best model architecture

This experiment aims to identify the best model architecture to suit in multilabel food recognition problem. For this reason, we have trained different models evaluated in the Food-201 dataset.

All the experiments were carried out with input shape = 224 x 224 x 3 and batch size = 12 as our fixed parameters. The first model was executed using the hyperparameters described below:

- Pretrained model: Resnet50
- Batch size: 12
- Input shape: 224 x 224 x 3
- Without data augmentation
- Without dropout
- Loss function: binary_cross-entropy
- Optimizer: Stochastic Gradient Descent with learning rate 0.01

From figure 5.4 and 5.5, which plot the F1-score and loss curves of the first model, we can observe a clear example of an overfit model. Therefore, we decided to train new models modifying the following hyper-parameters:

- **Optimizers and learning rates:** is important to choose a suitable optimizer to train deep models because the optimizers are used to update and calculate network parameters that affect model training and the output, so as to approximate or reach the optimal value, thereby minimizing the loss function. We had tried with several optimizers such as Adam with default values, Adam with lr=0.01, and Adadelta with default values. These changes aimed to see the implication of using different optimizers in a convnet model.
- **Regularization:** the aim of adding regularization techniques in the convnets is to reduce the overfitting problem of the first trained model.

- **Data augmentation techniques:** We also applied some data augmentation techniques to handle with overfitting problem. You can find the techniques used in section 4.5.1.

- **Base architecture:** by changing the base architecture, we aimed to see the performance of each pre-trained model in our Food-201 dataset.

- **New data:** after analyzing the per-class performance, we decided to add new image samples into the dataset because in some specific classes there was not enough data that enables our model to learn. So, retraining the algorithm on a bigger, richer, and more diverse data set should improve its performance.

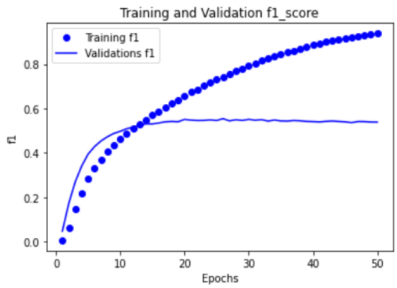

Figure 5.4: F1-score of Model1

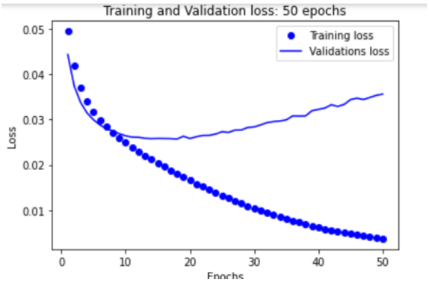

Figure 5.5: Loss of Model1

According to the loss and F1 curves of the first model shown previously, we were able to detect the overfitting problem on the model where the validation loss is much higher than the training loss. Also, the training F1-score is higher than the validation F1-score.

5.2.1 Results

We have trained eleven different models changing or combining some of the hyper-parameters mentioned early. Table 5.1 reports the detail architecture of each model.

In the initial stage of this experiment, we had focused on the evaluation of different techniques to reduce the overfitting effect (especially the first five models). These architectures have been extracted from the result of the following experiment, explained in 5.3. When this problem has been reduced, we were able to explore and analyze the performance of model by using different pre-trained architectures.

	Base Architecture	Optimizer	Dropout	Data Augmentation	Regularizer	Val F1	Test F1
Model1	ResNet50	SGD(lr=0.01)	NO	NO	NO	0.5403	0.1789
Model2	ResNet50	Adam()	NO	NO	NO	0.3981	0.1021
Model3	ResNet50	SGD(lr=0.01)	Dropout(0.4)	NO	NO	0.5012	0.1203
Model4	ResNet50	Adam(lr=0.01, rho=0.95)	SpatialDropout2D(0.5)	Type1	L2(0.0001)	0.4634	0.1239
Model5	ResNet50	Adam(lr=0.01, rho=0.95)	SpatialDropout2D(0.5)	Type1	L1(0.0001)	0.4012	0.1120
Model6	InceptionResnetV2	Adadelta()	SpatialDropout2D(0.4)	Type1	No	0.4292	0.4086
Model7	EfficientNetB7	Adadelta()	SpatialDropout2D(0.4)	Type1	No	0.5217	0.3812
Model8	DenseNet169	Adadelta()	SpatialDropout2D(0.4)	Type2	No	0.5052	0.4038
Model9	DenseNet169	Adadelta()	SpatialDropout2D(0.4)	Type3	No	0.4825	0.4246
Model10	EfficientB5	Adadelta()	SpatialDropout2D(0.4)	Type1	No	0.5838	0.3012
Model11	**EfficientB5**	**Adadelta()**	**SpatialDropout2D(0.4)**	**Type3**	**No**	**0.5006**	**0.4812**

Table 5.1: Architecture of trained models

As expected, the results of our models proved that:

- The effectiveness of adding dropout (in our case, SpatialDroupout2D) and data augmentation techniques into the model architecture for reducing the overfitting effect.
- The F1-score of the models using both techniques (ImageDataGenerator and Albumentations) are better than the models with a single data augmentation technique.

As shown in table 5.1, the best performing model with highest test F1-score was Model 11. We can observe that the F1-score of the validation and test sets are still really low. With this problem in mind, we have computed the per-class validation F1-score of the model.

	support	f1-score	class	order
192	60.0	0.040816	192	175
132	107.0	0.034483	132	176
11	141.0	0.028571	11	177
133	120.0	0.027778	133	178
169	661.0	0.020566	169	179
144	310.0	0.000000	144	180
180	230.0	0.000000	180	181
98	165.0	0.000000	98	182
43	117.0	0.000000	43	183
12	105.0	0.000000	12	184
47	104.0	0.000000	47	185
9	98.0	0.000000	9	186
175	95.0	0.000000	175	187
143	92.0	0.000000	143	188
170	92.0	0.000000	170	189
90	88.0	0.000000	90	190
48	84.0	0.000000	48	191
62	84.0	0.000000	62	192
87	76.0	0.000000	87	193

Figure 5.6: Per-class Validation F1-score

If we look at the validation F1-score and the training support of the classes shown in figure 5.6, some F1-score was 0, and also, the training support was very low. This would appear to indicate that we need to study deeply on these low-performing classes (explained in section 5.4).

5.3 Overfitting problem handling

This experiment was developed in parallel with the first experiment. In order to reduce the overfitting problem of the first model, we carried out several sub-experiments.

To overcome the overfitting problem, we did the following steps and in this experiment we focused on the first two steps.:

1. Adding dropout
2. Augmentation strategies

Because of the long runtime needed on the learning process with the complete Food-201 dataset and the necessity of study deeper the behaviour of the low performing classes, we had extracted the dataset into seven small datasets,

and we had applied data augmentation strategies mentioned in 4.5.1 to solve the over-fitting problem.

We extracted the seven dataset based to the per-class F1-score of the first base model, and we had picked the lowest F1-score classes.

- C1: image samples of class Sauce.
- C2: image samples of class Parsley plus C1.
- C3: image samples of class Spinach plus C2.
- C4: image samples of class Greens plus C3.
- C5: image sample of class chicken plus C4.
- C6: image samples of class beef plus C5
- C7: image samples of class chives plus C6.

The setting of the base architecture is shown as below:

- Pretrained model: Resnet50
- Batch size: 12
- Input shape: 224 x 224 x 3
- Without data augmentation
- Without dropout
- Loss function: binary_cross-entropy
- Optimizer: Stochastic Gradient Descent with learning rate 0.01

We did different sub-experiments with each dataset mentioned previously:

- Sub-experiment 1: train the base architecture model using the previous seven datasets.
- Sub-experiment 2: train the base architecture model adding dropout using the previous seven datasets.
- Sub-experiment 3: train the base architecture model adding dropout and some data augmentation techniques (using ImageDataGenerator) using the previous seven datasets.
- Sub-experiment 4: train the base architecture model adding dropout (using SpatialDropout2D(0.4)) and some data augmentation techniques (using Albumentations) using the previous seven datasets. In this case, we use Adadelta as the optimizer.

- Sub-experiment 5: train the base architecture model adding dropout and some data augmentation techniques (using Albumentations and ImageDataGenerator) using the previous seven datasets.

All the sub-experiments mentioned before use Resnet50 as the base architecture. To explore more about the over-fitting issue, we decided to train the model using InceptionRestnetV2 as the pretrained model:

- Sub-experiment 6: train the InceptionRestnetV2 architecture model adding dropout and some data augmentation techniques (using ImageDataGenerator) using the previous seven datasets.
- Sub-experiment 7: train the InceptionRestnetV2 architecture model adding dropout and some data augmentation techniques (using Albumentations) using the previous seven datasets. In this case, we use Adadelta as the optimizer.
- Sub-experiment 8: train the InceptionRestnetV2 architecture model adding dropout and some data augmentation techniques (using ImageDataGenerator + Albumentations) using the previous seven datasets.

Activation Maps and Feature Visualization

Once we discovered the overfitting problem, we thought that it is interesting to visualize the activation maps and extract features to make sure if the convnets can identify different classes of a given image.

With this in mind, we did different sub-experiments using different models (EfficientNetB5 and InceptionResnetV2) to visualize the activation maps and extract features from these models. In order to evaluate the result, we used the datasets C1, C2, C3, C4, C5, and Food-201.

And for each selected model, we did the following steps:

1. Create activation maps for the specific class (check if the activation maps can localize the position of that class).

   The first thing we have to do is to get the index of the prediction. After that, we need the final convolutional layer of the trained model to obtain the gradients. Once we get that, we will finally be able to compute the heatmap with some more operations.

```
#for each predicted labels, we extract the activation maps
for i in range(len(pred_label)):
  output = model.output[:, pred_label[i]]
  last_conv_layer = model.layers[-7]
  gradients = K.gradients(output, last_conv_layer.output)[0]
  mean_gradients = K.mean(gradients, axis=(0, 1, 2))
  extractor_fn = K.function([model.input], [mean_gradients, last_conv_layer.output[0]])
  mean_grad_vals, conv_output = extractor_fn([image])



  for j in range(1536):
      conv_output[:, :, j] *= mean_grad_vals[j
  heatmap = np.mean(conv_output, axis=-1)
  heatmap = np.maximum(heatmap, 0)
  heatmap /= np.max(heatmap)

  upsized_cam_image = cv2.resize(heatmap, original.shape[:2][::-1])
  heatmap = cv2.applyColorMap(np.uint8(255*upsized_cam_image), cv2.COLORMAP_JET)
  # display the cam image
```

Figure 5.7: How we get the heatmap

2. Convert the activation map to a binary mask and multiply with the original image.

```
mask = np.zeros( ( np.array(upsized_cam_image).shape[0], np.array(upsized_cam_image).shape[1], 3 ) )
mask[:,:,0] = upsized_cam_image # same value in each channel
mask[:,:,1] = upsized_cam_image
mask[:,:,2] = upsized_cam_image
```

Figure 5.8: How we compute the mask

```
original =cv2.cvtColor(original, cv2.COLOR_BGR2RGB)
maskedImage = original * (mask)
maskedImage =np.array(maskedImage,np.int64)
image= resize(maskedImage,(224, 224,3), order=1, mode='constant', cval=0, clip=True, preserve_range=True)
image = np.expand_dims(image, axis=0)
```

Figure 5.9: How we get the final image

3. Use the masked image on the classifier and extract features from the last but one FC layer.

```
model_features = tf.keras.Model(inputs=model.input, outputs=model.get_layer('dense_1').output)
features = model_features.predict(image)
```

Figure 5.10: How we get the last dense layer to extract features

4. Apply PCA and t-SNE to visualize the features: we have used "Projector Embedding".

5.3.1 Result

Since the dataset C1 contains only images of class sauce, for this experiment, we decided to show up only the F1-score of this class using the "Classification Report".

- Base Architecture training:

	C1	C2	C3	C4	C5	C6	C7
Training	0.99	1.0	1.0	1.0	1.0	0.99	1.0
Validation	0.39	0.49	0.45	0.4	0.4	0.19	0.16
Test	0.41	0.45	0.36	0.37	0.26	0.1	0.15

Table 5.2: F1-score of the Sub-experiment 1

- With dropout:

	C1	C2	C3	C4	C5	C6	C7
Training	1.0	1.0	0.97	0.95	0.84	0.81	0.81
Validation	0.71	0.66	0.56	0.47	0.47	0.25	0.2
Test	0.58	0.51	0.47	0.38	0.41	0.28	0.23

Table 5.3: F1-score of the Sub-experiment 2

- With dropout and data augmentation Type1:

	C1	C2	C3	C4	C5	C6	C7
Training	0.88	0.83	0.81	0.82	0.78	0.69	0.69
Validation	0.53	0.50	0.48	0.46	0.44	0.28	0.24
Test	0.51	0.49	0.47	0.42	0.38	0.22	0.26

Table 5.4: F1-score of the Sub-experiment 3

- With dropout and data augmentation Type2:

	C1	C2	C3	C4	C5	C6	C7
Training	0.97	0.97	0.96	0.96	0.96	0.96	0.95
Validation	0.65	0.59	0.53	0.52	0.47	0.21	0.23
Test	0.61	0.55	0.59	0.51	0.46	0.27	0.31

Table 5.5: F1-score of the Sub-experiment 4

- With dropout and data augmentation Type3:

	C1	C2	C3	C4	C5	C6	C7
Training	0.99	0.97	0.97	0.96	0.90	0.86	0.87
Validation	0.72	0.68	0.61	0.50	0.35	0.28	0.22
Test	0.66	0.61	0.53	0.46	0.42	0.34	0.29

Table 5.6: F1-score of the Sub-experiment 5

As expected, in each case of the previous five tables, the F1-score has decreased by adding new group images. This fact is understandable because by adding new group images, the convnets will be more complex.

Moreover, these sub-experiments revealed that with the same pretrained model, the performance has increased by applying dropout and data augmentation technique.

We also have tested with InceptionResnetV2 as pretrained model. And the results of these three sub-experiments were in line with the previous results.

- With dropout and data augmentation Type1:

	C1	C2	C3	C4	C5	C6	C7
Training	0.99	0.99	0.99	0.99	0.99	0.98	0.98
Validation	0.70	0.68	0.60	054	0.47	0.26	0.28
Test	0.67	0.61	0.57	0.56	0.49	0.32	0.22

Table 5.7: F1-score of the Sub-experiment 6

- With dropout and data augmentation Type2:

	C1	C2	C3	C4	C5	C6	C7
Training	0.99	0.99	0.98	0.99	0.99	0.89	0.97
Validation	0.75	0.73	0.62	0.53	0.44	0.27	0.23
Test	0.68	0.64	0.61	0.48	0.36	0.26	0.19

Table 5.8: F1-score of the Sub-experiment 7

- With dropout and data augmentation Type3:

	C1	C2	C3	C4	C5	C6	C7
Training	0.99	0.99	0.99	0.99	0.99	0.98	0.96
Validation	0.77	0.72	0.62	0.53	0.49	0.29	0.27
Test	0.70	0.66	0.64	0.54	0.41	0.38	0.32

Table 5.9: F1-score of the Sub-experiment 8

As we can observe in the previous sub-experiments, the performance have increased by adding dropout and data augmentation techniques. Due to the results, we extracted the following approach to create new models for the first experiment:

1. The importance of adding dropouts into our models.
2. The effect of applying ImageDataGenerator techniques with Albumentation's library techniques that we explained in subsection 4.5.1 into our models.

Activation maps and Features Visualization

Activation Maps: as shown in the two images in Figures 5.11 and 5.12, we were able to see that in some of the cases, the activation maps were correct, such as class "Dumplings" and class "Spring Rolls".

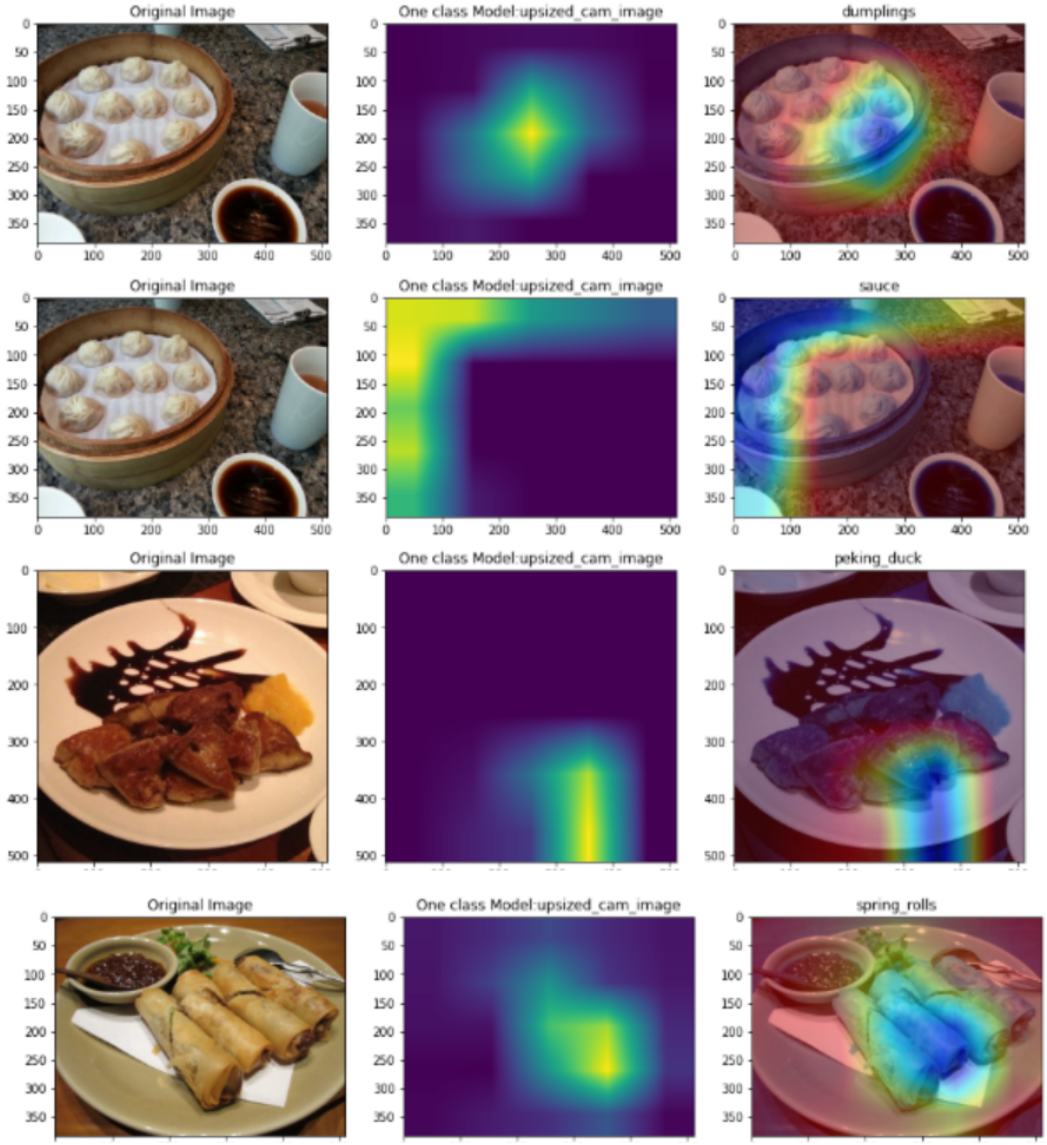

Figure 5.11: Activation map of some images

Despite that, in some cases like images with class "Sauce", the model could not localize it. The F1-score of this class was very low. We were not surprised about that result because class "Sauce" was one of the ten classes with the lowest F1-score.

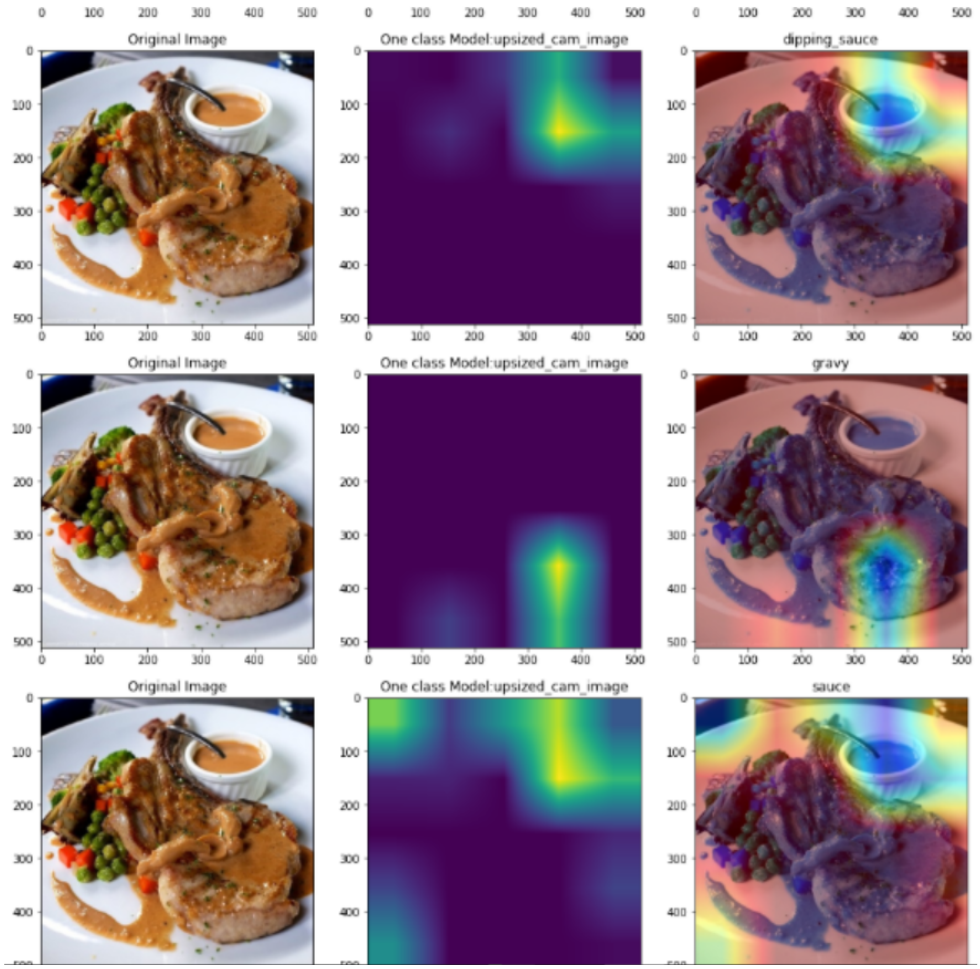

Figure 5.12: Activation map of some images

Binary masked Images: now we will show some binary masked images that we used for extract features. Masking involves setting some of the pixel values to zero.

In our case, we are interested in the pixels with a value number close to 1. In other words, we want just the regions in the image in which the pixels are relevant to one specific class.

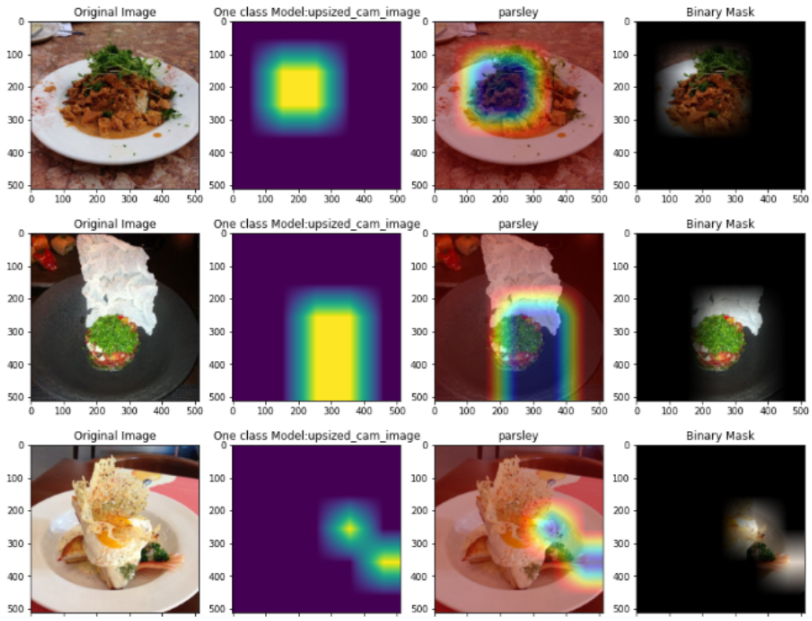

Figure 5.13: Some binary masked images

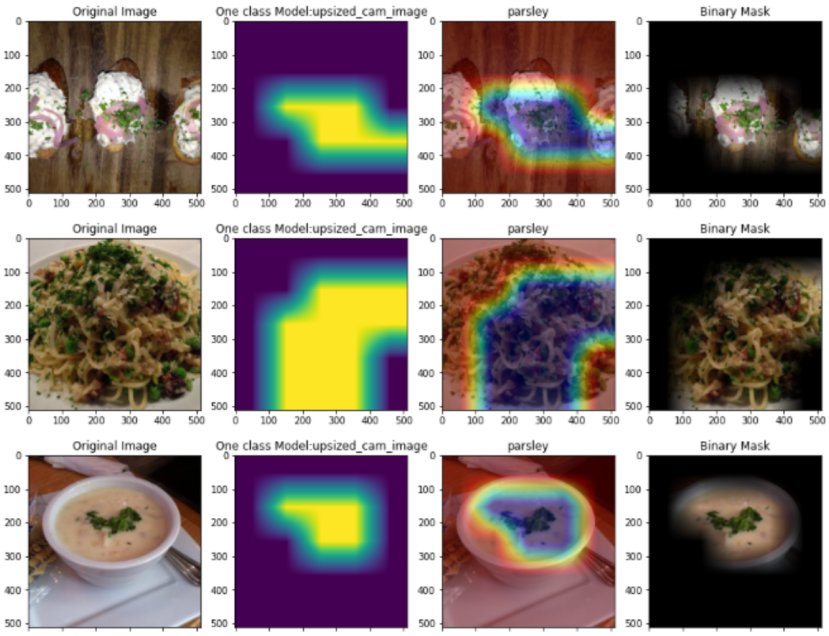

Figure 5.14: Some binary masked images

PCA and T-sne:

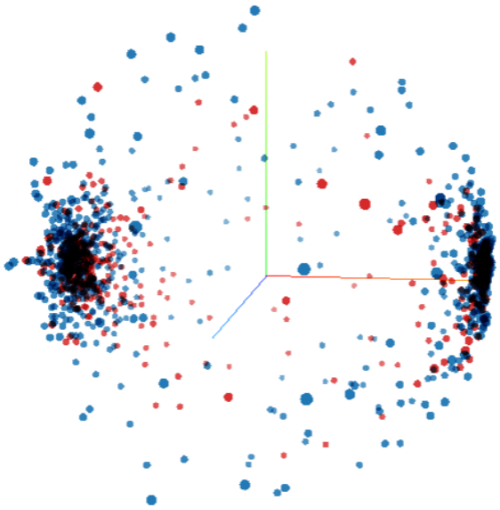

Figure 5.15: PCA visualization of C2

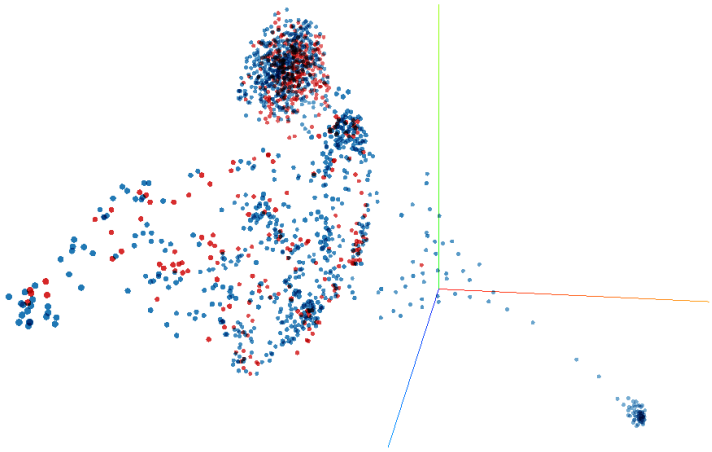

Figure 5.16: T-sne visualization of C2

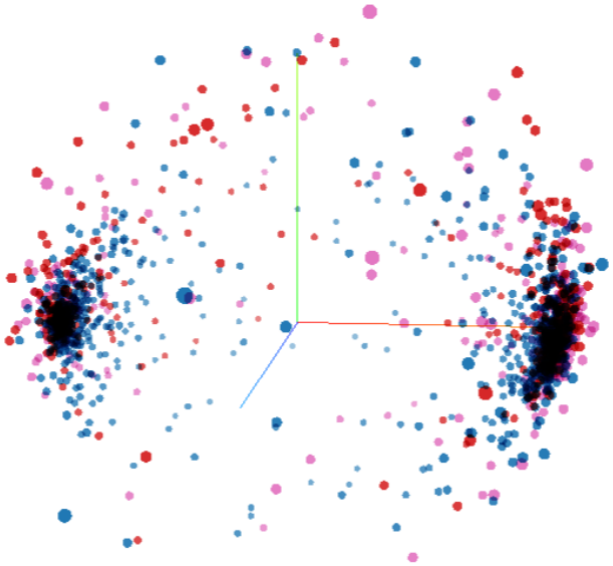

Figure 5.17: PCA visualization of C3

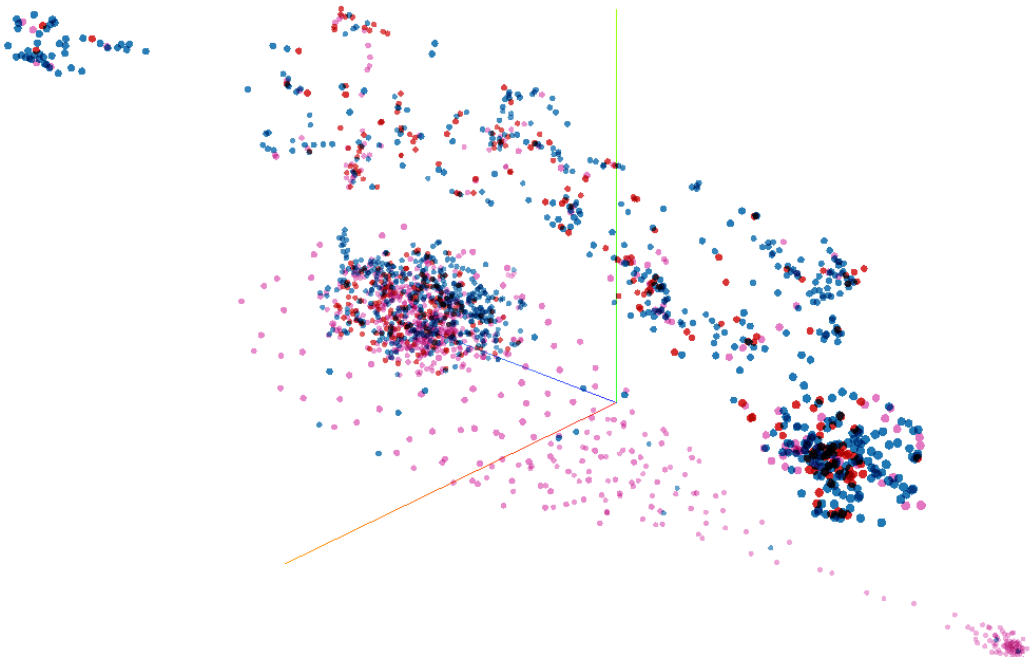

Figure 5.18: T-sne of C3

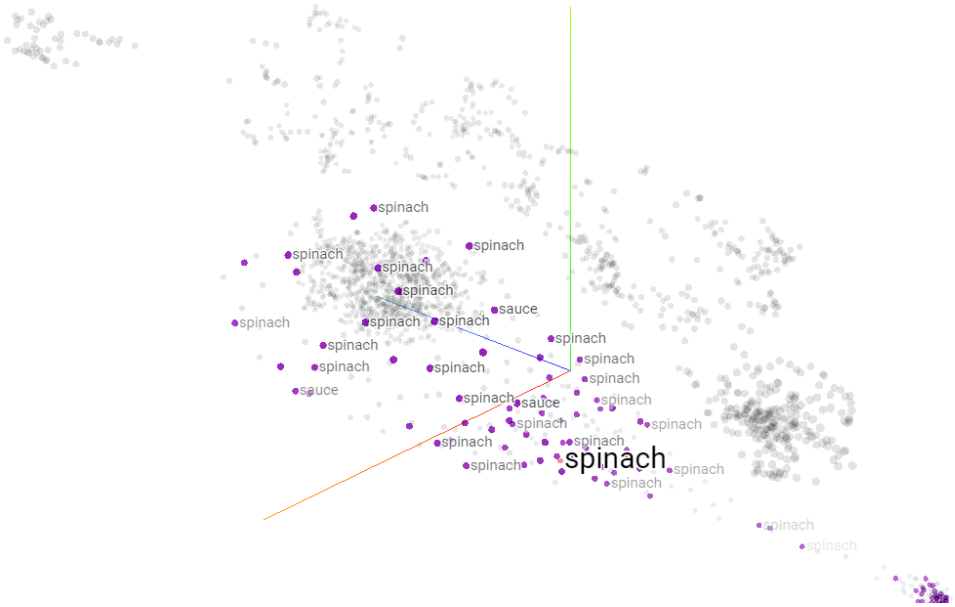

Figure 5.19: T-sne of C3: Spinach

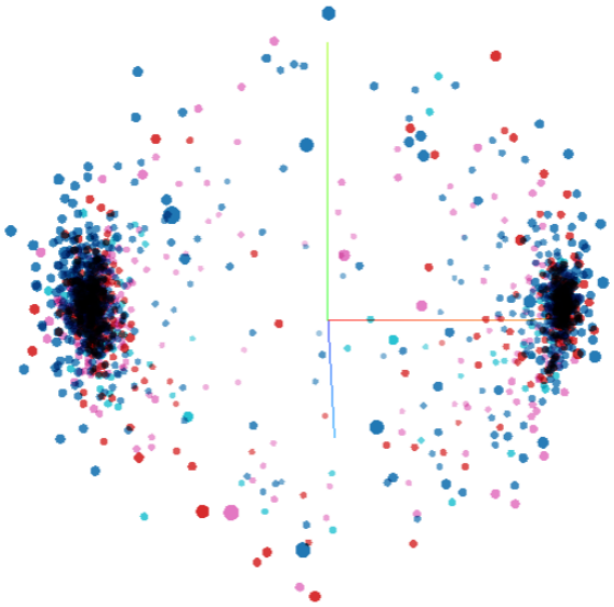

Figure 5.20: PCA visualization of C4

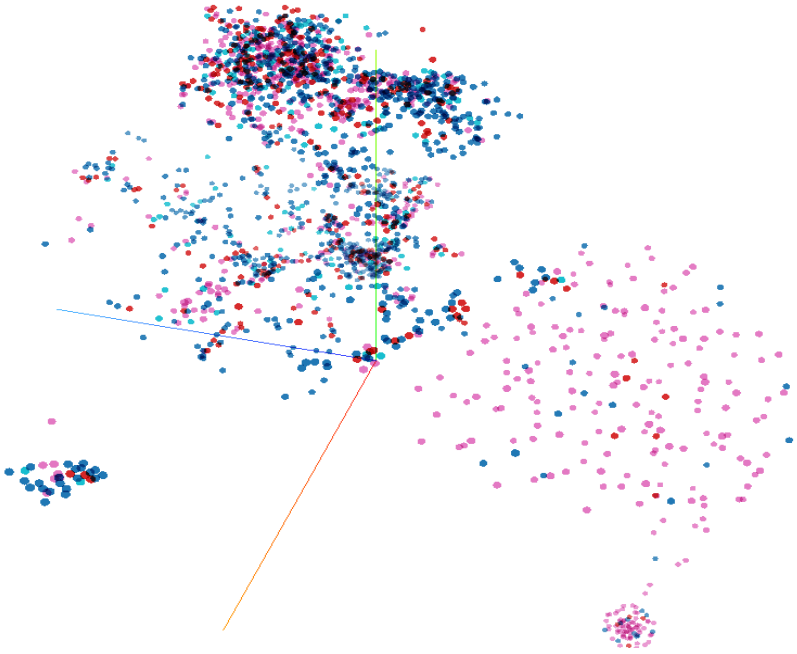

Figure 5.21: T-sne of C4

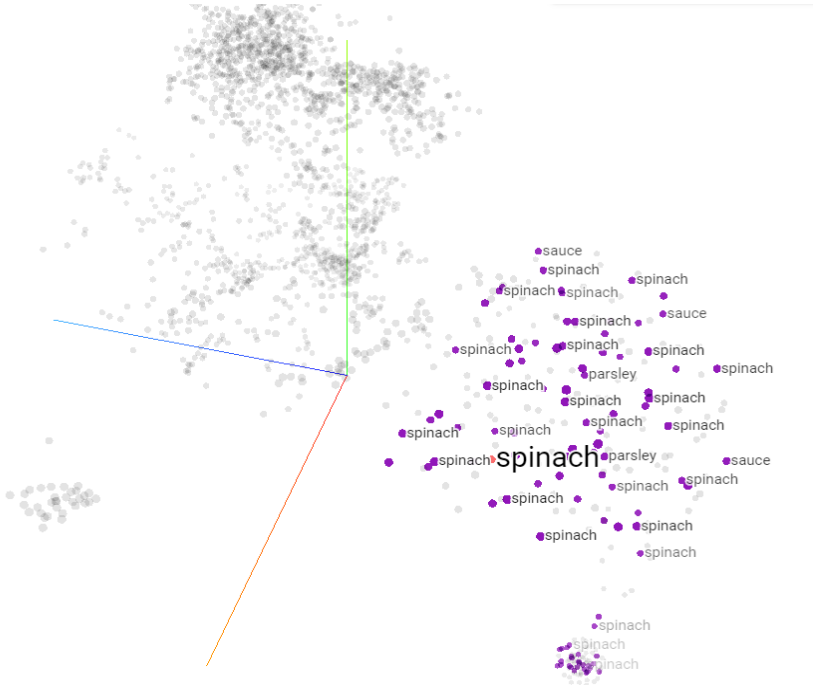

Figure 5.22: T-sne of C4: Spinach

5.4 Train with new data

This experiment aimed to improve our best-performing model (with less over-fitting effect and with the best F1-score). According to the results of Experiment 1 (Figure 5.2), the best performing model has the following hyper-parameters:

- Convolutional base: EfficientNetB5
- Input shape: 224 x 224 x 3
- Batch size: 12
- SpatialDropout2D: 0.5
- Optimizer: Adadelta
- Loss function: binary cross-entropy
- Data Augmentation techniques:
  - Data augmentation using Albumentations library 4.9.
  - Data augmentation using ImageDataGenerator 4.8.

After computing the per-class F1-score of this model, we have noticed that even by adding dropout and augmentation strategies in our model, there were still some low performing classes. Due to that, we decided to study deeply the ten low-performing classes by adding new images into them that have been partially labeled.

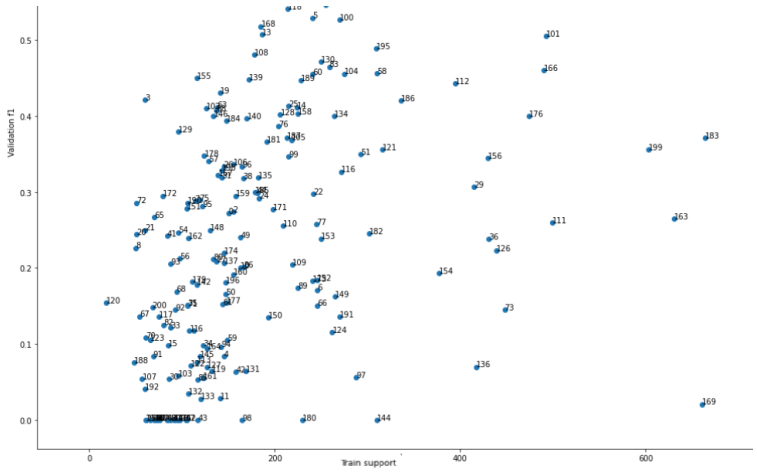

Figure 5.23: Validation F1-score vs Train Support

After analyzing the results, we decided to apply the following techniques to improve our model:

1. Add new partially-labeled images into the ten low-performing classes.
2. Change the loss function according to the paper "Learning a deep convnet for multi-label classification with partial labels" [13] because of our single-labeled dataset.

Our Augmented dataset

Our augmented dataset is composed of images downloaded from Yummly, Meituan, and Flickr. We have labeled them according to the Food-201 annotation files, but we only single-labeled them due to the complexity of annotating all food present in the images.

The selection of these images was based on the result of the best performing model of experiment 1. We have chosen the ten lowest F1-scores classes by using "classification report (from Keras)".

The reason that we decided to add new images into the Food-201 dataset was that we want to see how the model improves by adding more samples of the low performing classes.

We have added 200 images into each selected class. The classes are the following:

* Bun
* Cup cakes
* Gravy
* Hot and sour soup
* Lobster bisque
* Mint
* Mushrooms
* Pizza
* Spinach
* Takoyaki

After getting the result of the model with new images, we have decided to add 200 more samples into the five lowest performance classes (bun, gravy, mint, mushrooms, and spinach) of the ten previous ones. Also, we have added more images into the validation and test dataset of these classes according to the proportion of the split of the original dataset: $split = \{'Train' : 0.8, 'Val' : 0.15, 'Test' : 0.05\}$. You can find the support of our new images in the following table:

Suport												
	Before			After adding 200 images			Adding 200 to 5 low performance class			Adding images to val and test sets		
	Train	Val	Test	Train	Val	Test	Train	Val	Test	Train	val	test
LobsterBisque	311	150	40	511	150	40	511	150	40	511	150	40
Mint	**109**	**44**	**5**	**309**	**44**	**5**	**509**	**44**	**5**	**509**	**95**	**32**
Mushrooms	**127**	**51**	**16**	**327**	**51**	**16**	**527**	**51**	**16**	**527**	**98**	**33**
Spinach	**230**	**95**	**31**	**430**	**95**	**31**	**530**	**95**	**31**	**530**	**100**	**33**
Bun	**71**	**29**	7	**271**	**29**	7	**571**	**29**	7	**571**	**107**	**35**
Takoyaki	312	150	39	512	150	39	512	150	39	512	150	39
Gravy	**142**	**59**	**20**	**342**	**59**	**20**	**542**	**59**	**20**	**542**	**101**	**34**
CupCakes	321	151	37	521	151	37	521	151	37	521	151	37
pizza	338	154	30	538	154	30	538	154	30	538	154	30
HotAndSourSoup	322	150	31	522	150	31	522	150	31	522	150	31

Figure 5.24: Support

Description of the Sub-experiments

To see the effectiveness of adding new single-labeled images into the Food-201 dataset and the performance of the partial binary cross-entropy [13]) we have divided this experiment into some small sub-experiments.

	Training set	Validation set	Test set	Loss Function
Sub-experiment1	Food-201	Food-201	Food-201	Binary Cross-entropy
Sub-experiment2	Food-201	Food-201	Food-201	Partial Binary Cross-entropy
Sub-experiment3	Food-201+New Images	Food-201	Food-201	Binary Cross-entropy
Sub-experiment4	Food-201+New Images	Food-201	Food-201	Partial Binary Cross-entropy
Sub-experiment5	Food-201+New Images	Food-201+New Images	Food-201+New Images	Binary Cross-entropy
Sub-experiment6	Food-201+New Images	Food-201+New Images	Food-201+New Images	Partial Binary Cross-entropy

Table 5.10: Description of the sub-experiments

5.4.1 Results on the Augmented Dataset with Partially Labeled Images

The goal of this experiment was to improve the best model that we had trained in section 5.2.1. We used the best performing model architecture to train with new data that have false negatives. As explained in section 5.4, we have divided into

several sub-experiments. First of all, we will show the global performance of each sub-experiment (F1):

	SubExp1			SubExp3			SubExp5		
	Train	Val	Test	Train	Val	Test	Train	Val	Test
Global Performance	0.878	0.5	0.48	0.78	0.65	0.60	0.90	0.64	0.62

Table 5.11: Global F1-score of the sub-experiments

	SubExp2			SubExp4			SubExp6		
	Train	Val	Test	Train	Val	Test	Train	Val	Test
Global Performance	0.84	0.67	0.64	0.80	0.65	0.62	0.92	0.66	0.63

Table 5.12: Global F1-score of the sub-experiments

Two clarifications of the previous two tables:

- The first table contains sub-experiments with binary cross-entropy as the loss function. And the second table is with the new loss function.
- The difference between the sub-experiments of the same figure is the dataset used.

We can observe that the validation and test F1-scores of the second table 5.14 are better than the F1-scores of the first table 5.11. And also, the F1-scores of the same table have increased from left to right.

Now, we will show the per-class F1-score of each sub-experiment.

	SubExp1			SubExp3			**SubExp5**		
	Train	Val	Test	Train	Val	Test	Train	Val	Test
Mint	0.78	0.14	0.11	0.88	0.26	0.00	0.94	0.54	**0.63**
Mushrooms	0.73	0.18	0.30	0.76	0.14	0.22	0.87	0.47	**0.53**
Spinach	0.74	0.16	0.22	0.79	0.15	0.19	0.79	0.19	**0.51**
Bun	0.52	0.13	0.00	0.81	0.16	0.00	0.89	0.65	**0.64**
Gravy	0.65	0.19	0.32	0.79	0.21	0.22	0.80	0.38	**0.32**

Table 5.13: Per-class F1-score of the sub-experiments

	SubExp2			SubExp4			**SubExp6**		
	Train	Val	Test	Train	Val	Test	Train	Val	Test
Mint	0.72	0.16	0.00	0.82	0.29	0.15	0.96	0.54	**0.69**
Mushrooms	0.56	0.17	0.20	0.77	0.18	0.24	0.92	0.49	**0.56**
Spinach	0.58	0.18	0.24	0.77	0.18	0.16	0.91	0.20	**0.52**
Bun	0.13	0.00	0.00	0.86	0.07	0.11	0.93	0.68	**0.63**
Gravy	0.61	0.24	0.28	0.74	0.21	0.24	0.88	0.34	**0.37**

Table 5.14: Per-class F1-score of the sub-experiments

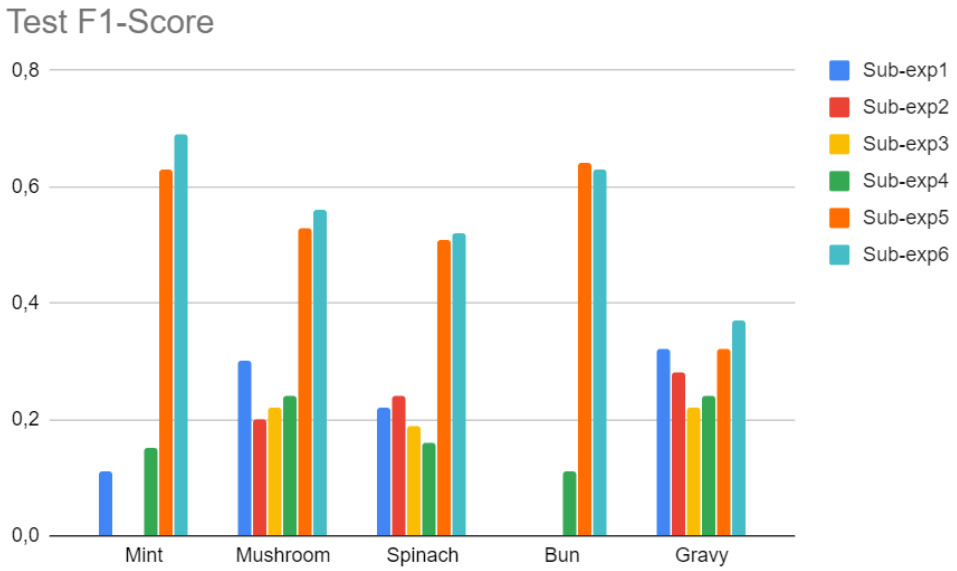

Figure 5.25: Test F1-score of the sub-experiments

According to the results, we can easily extract the following approaches:

- If we take a look at the table 5.14 shown previously, we can observe that the results are better with adding more images into our training set. And what is more, adding more image samples into our validation and test sets also help the models to get a higher F1-score.
- Because of the single-labeled image samples that we have added into our dataset, there are some partially labeled images. So, with the new loss function, the models can predict better than with the binary cross-entropy as the loss function.

Chapter 6

Conclusions and Future Lines

In this project, we dove in the field of food image analysis, more specifically to the multi-label food recognition. This work had as primary objective to analyze the multi-label food classification using a convolutional neural network. We explored in the detail the multi-label CNNs and their application to the multi-label food recognition. We proved the importance of having a well-annotated dataset, and the potential to ease the process by the partially labeled dataset. To this purpose, we applied a recently introduced loss function especially suitable for partially labeled datasets. We have trained several deep models by using pretrained convnets 4.2, and we have evaluated them with the Food-201 dataset [31].

When we started the experimental part of this project another objective soon appeared that referred to the problem of overfitting that the first trained model implies. Due to that problem, we have studied and applied different techniques to cover the problem. According to the results of each experiment, the insights are:

- The importance of adding dropout and applying data augmentation techniques in the model architecture to reduce the overfitting effect 5.3.1.
- The performance of the convnets increases by adding more image samples into the dataset 5.4.1.
- With a partially labeled dataset, the performance increases by changing the binary cross-entropy loss function into partial BCE [13].

The work done in this project could be extended in several directions. Firstly, the exploration of new loss functions or techniques to handle partial labels is a promising way to easy the annotation process and improve the models. Then, the usage of new different networks such as InceptionV3 [45] to classify multi-labeled images could also be explored. Last but not least, an open question is how to create a protocol to easy the annotation process of multi-label food dataset.

Food-201

Most classification datasets do not have an exactly equal number of instances in each class, but a small difference often does not matter. As you might observe in the following figures, the support of the classes is not equal.

An imbalanced data typically refers to a problem with classification problems where the classes are not represented equally. We can say that Food-201 is an imbalanced dataset.

- Training set

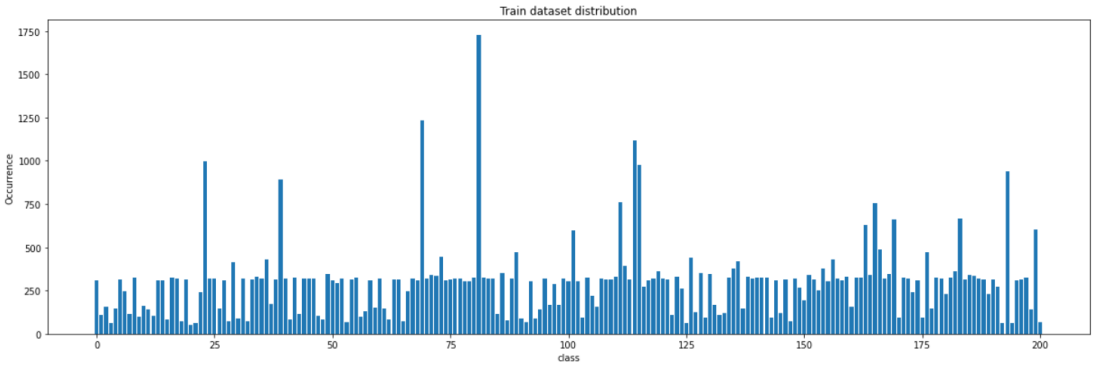

Figure 1: Class Support of training set

- Validation set

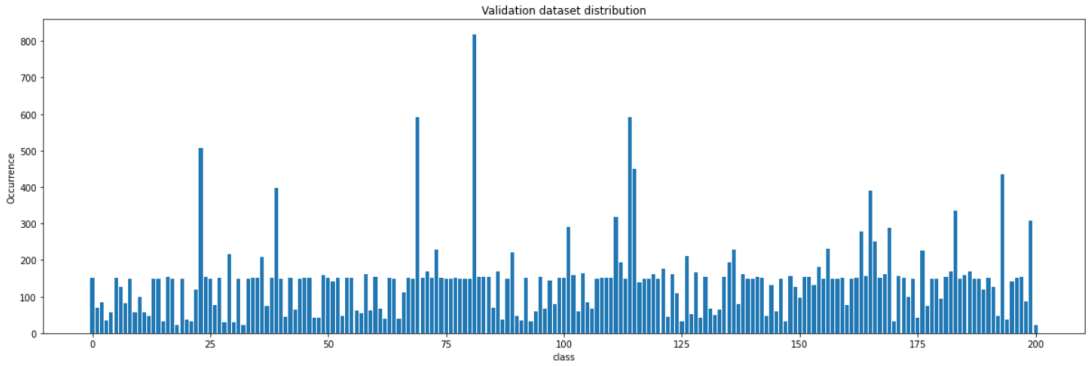

Figure 2: Class Support of validation set

- Test set

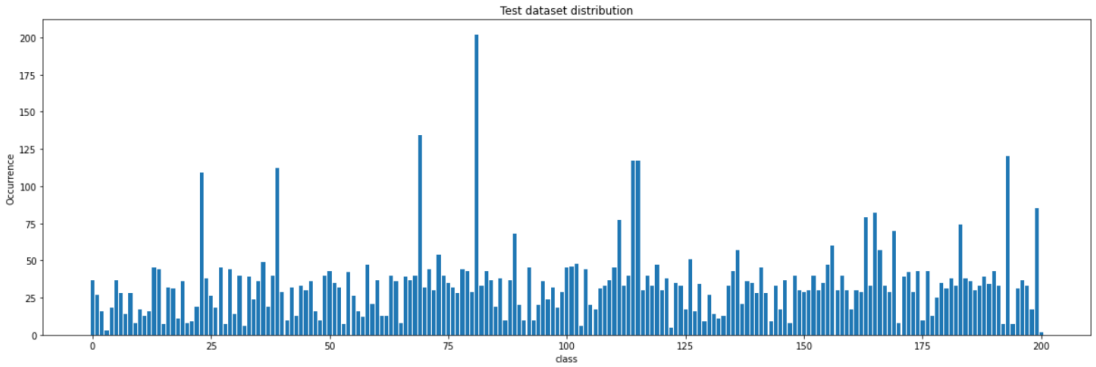

Figure 3: Class Support of test set

Partial Binary Cross-entropy

```python
alpha= -4.45
beta=5.45
gamma=3

def g(py):
  return alpha * (py**gamma) + beta
def binary_crossentropy_new(y_true, y_pred, from_logits=False, label_smoothing=0):
  """Computes the binary crossentropy loss.
  Standalone usage:
  >>> y_true = [[0, 1], [0, 0]]
  >>> y_pred = [[0.6, 0.4], [0.4, 0.6]]
  >>> loss = tf.keras.losses.binary_crossentropy(y_true, y_pred)
  >>> assert loss.shape == (2,)
  >>> loss.numpy()
  array([0.916 , 0.714], dtype=float32)
  Args:
    y_true: Ground truth values. shape = `[batch_size, d0, .. dN]`.
    y_pred: The predicted values. shape = `[batch_size, d0, .. dN]`.
    from_logits: Whether `y_pred` is expected to be a logits tensor. By default,
      we assume that `y_pred` encodes a probability distribution.
    label_smoothing: Float in [0, 1]. If > `0` then smooth the labels.
  Returns:
    Binary crossentropy loss value. shape = `[batch_size, d0, .. dN-1]`.
  """
  y_pred = ops.convert_to_tensor_v2(y_pred)
  y_true = math_ops.cast(y_true, y_pred.dtype)
  label_smoothing = ops.convert_to_tensor_v2(label_smoothing, dtype=K.floatx())

  def _smooth_labels():
    return y_true * (1.0 - label_smoothing) + 0.5 * label_smoothing

  y_true = smart_cond.smart_cond(label_smoothing,  _smooth_labels, lambda: y_true)

  return g(0.9)*K.mean( K.binary_crossentropy(y_true, y_pred, from_logits=from_logits), axis=-1)
```

Figure 4: New Loss Function [13]

Bibliography

[1] Eduardo Aguilar, Beatriz Remeseiro, Marc Bolaños, and Petia Radeva, *Grab, pay and eat: Semantic food detection for smart restaurants*, 2017.

[2] S. Albawi, T. A. Mohammed, and S. Al-Zawi, *Understanding of a convolutional neural network*, 2017 International Conference on Engineering and Technology (ICET), 2017, pp. 1–6.

[3] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool, *Surf: Speeded up robust features*, Computer Vision – ECCV 2006 (Berlin, Heidelberg) (Aleš Leonardis, Horst Bischof, and Axel Pinz, eds.), Springer Berlin Heidelberg, 2006, pp. 404–417.

[4] V. Bettadapura, E. Thomaz, A. Parnami, G. D. Abowd, and I. Essa, *Leveraging context to support automated food recognition in restaurants*, 2015 IEEE Winter Conference on Applications of Computer Vision, 2015, pp. 580–587.

[5] Marc Bolaños, Aina Ferrà, and Petia Radeva, *Food ingredients recognition through multi-label learning*, New Trends in Image Analysis and Processing – ICIAP 2017 (Cham) (Sebastiano Battiato, Giovanni Maria Farinella, Marco Leo, and Giovanni Gallo, eds.), Springer International Publishing, 2017, pp. 394–402.

[6] S.S. Bucak, Rong Jin, and Anil K. Jain, *Multi-label learning with incomplete class assignments*, Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2011), 2011.

[7] H. M. Bui, M. Lech, E. Cheng, K. Neville, and I. S. Burnett, *Object recognition using deep convolutional features transformed by a recursive network structure*, IEEE Access **4** (2016), 10059–10066.

[8] Alexander Buslaev, Vladimir I. Iglovikov, Eugene Khvedchenya, Alex Parinov, Mikhail Druzhinin, and Alexandr A. Kalinin, *Albumentations: Fast and flexible image augmentations*, Information **11** (2020), no. 2, 125.

[9] J. Canny, *A computational approach to edge detection*, IEEE Transactions on Pattern Analysis and Machine Intelligence **PAMI-8** (1986), no. 6, 679–698.

[10] M. Chen, K. Dhingra, W. Wu, L. Yang, R. Sukthankar, and J. Yang, *Pfid: Pittsburgh fast-food image dataset*, 2009 16th IEEE International Conference on Image Processing (ICIP), 2009, pp. 289–292.

[11] Gianluigi Ciocca, Paolo Napoletano, and Raimondo Schettini, *Food recognition and leftover estimation for daily diet monitoring*, vol. 9281, 09 2015, pp. 334–341.

[12] N. Dalal and B. Triggs, *Histograms of oriented gradients for human detection*, 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), vol. 1, 2005, pp. 886–893 vol. 1.

[13] Thibaut Durand and Nazanin Mehrasa Greg Mori, *Learning a deep convnet for multi-label classification with partial labels*, 2019.

[14] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, *Object detection with discriminatively trained part-based models*, IEEE Transactions on Pattern Analysis and Machine Intelligence **32** (2010), no. 9, 1627–1645.

[15] Soren Goyal and Paul Benjamin, *Object recognition using deep neural networks: A survey*, (2014).

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, *Deep residual learning for image recognition*, 2015.

[17] H. Hoashi, T. Joutou, and K. Yanai, *Image recognition of 85 food categories by feature fusion*, 2010 IEEE International Symposium on Multimedia, 2010, pp. 296–301.

[18] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger, *Densely connected convolutional networks*, 2016.

[19] Dat Huynh and Ehsan Elhamifar, *Fine-grained generalized zero-shot learning via dense attribute-based attention*, Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), June 2020.

[20] Sergey Ioffe and Christian Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, 2015.

[21] M. T. Islam, B. M. N. Karim Siddique, S. Rahman, and T. Jabid, *Image recognition with deep learning*, 2018 International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS), vol. 3, 2018, pp. 106–110.

[22] Hokuto Kagaya, Kiyoharu Aizawa, and Makoto Ogawa, *Food detection and recognition using convolutional neural network*, 11 2014.

[23] Y. Kawano and K. Yanai, *Automatic expansion of a food image dataset leveraging existing categories with domain adaptation*, Proc. of ECCV Workshop on Transferring and Adapting Source Knowledge in Computer Vision (TASK-CV), 2014.

[24] Yoshiyuki Kawano and Keiji Yanai., *Food image recognition with deep convolutional features.*, 2014, pp. 589–593.

[25] Diederik P. Kingma and Jimmy Ba, *Adam: A method for stochastic optimization*, 2014.

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton., *Imagenet classification with deep convolutional neural networks.*, 2012, pp. 1097–1105.

[27] N. Martinel, G. Foresti, and C. Micheloni, *Wide-slice residual networks for food recognition*, 2018 IEEE Winter Conference on Applications of Computer Vision (WACV) (Los Alamitos, CA, USA), IEEE Computer Society, mar 2018, pp. 567–576.

[28] Y. Matsuda, H. Hoashi, and K. Yanai, *Recognition of multiple-food images by detecting candidate regions*, 2012 IEEE International Conference on Multimedia and Expo, 2012, pp. 25–30.

[29] Y. Matsuda and K. Yanai, *Multiple-food recognition considering co-occurrence employing manifold ranking*, Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012), 2012, pp. 2017–2020.

[30] Yuji Matsuda, Hajime Hoashi, , and Keiji Yanai., *Recognition of multiple-food images by detecting candidate regions.*, IEEE, 08 2012, pp. 25–30.

[31] Austin Meyers, Nick Johnston, Vivek Rathod, Anoop Korattikara, Alex Gorban, Nathan Silberman, Sergio Guadarrama, George Papandreou, Jonathan Huang, and Kevin P Murphy., *Im2calories: towards an automated mobile vision food diary.*, 2015, pp. 1233–1241.

[32] Lili Pan, Samira Pouyanfar, Hao Chen, Jiaohua Qin, and Shu Ching Chen, *Deepfood: Automatic multi-class lassification of food materials using deep learning.*, 2017, pp. 181–189.

[33] Jie Peng, Shuai Kang, Zhengyuan Ning, Hangxia Deng, Jingxian Shen, Yikai Xu, Jing Zhang, Wei Zhao, Xinling Li, Wuxing Gong, Jinhua Huang, and

Li Liu, *Residual convolutional neural network for predicting response of transarterial chemoembolization in hepatocellular carcinoma from ct imaging*, European Radiology **30** (2019), 1–12.

[34] Sirawan Phiphitphatphaisit and Olarik Surinta, *Food image classification with improved mobilenet architecture and data augmentation*, 04 2020.

[35] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi., *You only look once: Unified, real-time object detection.*, 2016, pp. 779–788.

[36] Ramprasaath Rs, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra, *Grad-cam: Visual explanations from deep networks via gradient-based localization*, 10 2017, pp. 618–626.

[37] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei, *Imagenet large scale visual recognition challenge*, 2014.

[38] Jorge Sanchez, Florent Perronnin, Thomas Mensink, and Jakob Verbeek., *Image classification with the fish vector: Theory and practice.*, 2013, pp. 222–245.

[39] Karen Simonyan and Andrew Zisserman, *Very deep convolutional networks for large-scale image recognition*, 2014.

[40] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, *Dropout: A simple way to prevent neural networks from overfitting*, Journal of Machine Learning Research **15** (2014), no. 56, 1929–1958.

[41] M. A. Subhi, S. H. Ali, and M. A. Mohammed, *Vision-based approaches for automatic food recognition and dietary assessment: A survey*, IEEE Access **7** (2019), 35370–35381.

[42] Yu-Yin Sun, Yin Zhang, and Zhi-Hua Zhou, *Multi-label learning with weak label*, Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI'10, AAAI Press, 2010, p. 593–598.

[43] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi, *Inception-v4, inception-resnet and the impact of residual connections on learning*, 2016.

[44] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich, *Going deeper with convolutions*, 2014.

[45] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna, *Rethinking the inception architecture for computer vision*, CoRR **abs/1512.00567** (2015).

[46] Taichi Joutou and Keiji Yanai, *A food image recognition system with multiple kernel learning*, 2009 16th IEEE International Conference on Image Processing (ICIP), 2009, pp. 285–288.

[47] Mingxing Tan and Quoc V. Le, *Efficientnet: Rethinking model scaling for convolutional neural networks*, 2019.

[48] IDF team, *Diabetes facts figures*, `https://www.idf.org/aboutdiabetes/what-is-diabetes/facts-figures.html`, Accessed 2020-09-02.

[49] Keras team, *Keras applications*, `https://keras.io/ap`, Accessed 2020-08-31.

[50] Who team, *Coronavirus disease (covid-19) pandemic*, `https://www.who.int/emergencies/diseases/novel-coronavirus-2019`, Accessed 2020-09-01.

[51] Who team et al., *Obesity and overweight*, `https://www.who.int/news-room/fact-sheets/detail/obesity-and-overweight`, Accessed 2020-09-03.

[52] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christopher Bregler, *Efficient object localization using convolutional networks*, 2014.

[53] Laurens van der Maaten and Geoffrey Hinton, *Viualizing data using t-sne*, Journal of Machine Learning Research **9** (2008), 2579–2605.

[54] Hui Wu, Michele Merler, Rosario Uceda-Sosa, and John Smith, *Learning to make better mistakes: Semantics-aware visual food recognition*, 10 2016, pp. 172–176.

[55] Guangming Xiong, Xin Li, Junqiang Xi, Spencer G. Fowers, and Huiyan Chen, *Object distance estimation based on stereo vision and color segmentation with region matching*, Advances in Visual Computing (Berlin, Heidelberg) (George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Ronald Chung, Riad Hammound, Muhammad Hussain, Tan Kar-Han, Roger Crawfis, Daniel Thalmann, David Kao, and Lisa Avila, eds.), Springer Berlin Heidelberg, 2010, pp. 368–376.

[56] K. Yanai and Y. Kawano, *Food image recognition using deep convolutional network with pre-training and fine-tuning*, 2015 IEEE International Conference on Multimedia Expo Workshops (ICMEW), 2015, pp. 1–6.

[57] S. Yang, M. Chen, D. Pomerleau, and R. Sukthankar, *Food recognition using statistics of pairwise local features*, 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2010, pp. 2249–2256.

[58] Sergey Zagoruyko and Nikos Komodakis, *Wide residual networks*, Proceedings of the British Machine Vision Conference (BMVC) (Edwin R. Hancock Richard C. Wilson and William A. P. Smith, eds.), BMVA Press, September 2016, pp. 87.1–87.12.

[59] Matthew D. Zeiler, *Adadelta: An adaptive learning rate method*, 2012.

[60] Weiyu Zhang, Qian Yu, Behjat Siddiquie, Ajay Divakaran, and Harpreet Sawhney, *"snap-n-eat": Food recognition and nutrition estimation on a smartphone*, Journal of Diabetes Science and Technology **9** (2015), no. 3, 525–533, PMID: 25901024.

[61] F. Zhou and Y. Lin, *Fine-grained image classification by exploring bipartite-graph labels*, 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 1124–1133.