UNIVERSITAT DE BARCELONA

Treball final de grau

GRAUS SIMULTANIS EN MATEMÀTIQUES I EN ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

A multimodal deep learning approach for food tray recognition

Autor: Joan Peracaula Prat

Directors:	**Marc Bolaños and Petia Radeva**
Realitzat a:	**Departament de Matemàtiques i Informàtica**

Barcelona, **13 de setembre de 2020**

UNIVERSITAT DE BARCELONA

Abstract

Facultat de Matemàtiques i Informàtica

Departament de Matemàtiques i Informàtica

A multimodal deep learning approach for food tray recognition

by Joan Peracaula Prat

Food recognition, object detection and classification applied to the food domain, is the main topic of this work. We have studied the problem of recognising food instances in tray images of self-service restaurants and have proposed a novel multimodal deep learning approach. From images and daily menus, the model presented uses two state of the art models in object detection and classification and a multimodal neural network to make significantly refined predictions compared to the baseline object detection model, achieving a class weighted average F1-score of 0.862. An ensemble model built from the proposed and the baseline models, also presented in this work, improves the results achieving a class weighted average F1-score of 0.877.

El reconeixement de menjar, detecció i classificació d'objectes aplicat al terreny del menjar, és el tema principal d'aquest treball. Hem estudiat el problema de reconèixer instàncies de menjar en imatges de safates de restaurants d'autoservei i hem proposat una nova aproximació basada en aprenentatge profund amb multiples modalitats. A partir d'imatges i menús diaris, el model presentat utilitza dos models de l'estat de l'art actual en detetecció d'objectes i classificació, i una xarxa neuronal multi-modal per fer prediccions significativament més refinades que el model de detecció d'objectes agafat com a base, aconseguint una mitjana ponderada per classes de F1-scores de 0.862. En aquest treball també presentem un model "conjunt", construït a partir del model proposat i del model base, el qual millora els resultats aconseguint una mitjana ponderada per classes de F1-scores de 0.877.

2010 Mathematics Subject Classification. 62M45, 68T07, 68U10
2020 Mathematics Subject Classification. 62M45, 65D19, 68T07

Acknowledgements

First of all, I would like to acknowledge my two supervisors, Marc Bolaños and Petia Radeva, who have provided me a great guidance through the development of the project. They have clarified to me deep learning concepts during the research process, helped in technical issues during the implementation process and gave valuable advise during the thesis writing process.

Secondly, I would like to thank the constant support received by my family and Anna. All of them have encouraged me and have helped me in whatever they could.

Contents

Chapter 1

Introduction

1.1 Motivation

Eating is one of the few things we all have in common, we must eat to live, but farther from being a mere survival habit, eating has become much more in the current society. Eating can be a way to personal well-being. Eating can be a pleasant activity. And most important eating is one of the principle socialising activities. We eat with our family, with friends, with work mates, at home, at bars, at restaurants, in the street...

Eating is an important daily activity in our society, and restaurants are one of the principle meeting points for this activity. Restaurants can have different objectives apart from serving food, for example giving clients a pleasant time with high quality food, offering a good environment for meetings, serving food fast, etc. Regarding self-service restaurants, what differentiate them from other restaurants is that, in general, they don't have waiters and is the client who serves him/herself.

After that, there's a worker who checks and charges the client for the food taken. Due to human action, the check out process may lead to some problems such as slowness, long queues of clients, errors in the payment process, etc. What is more, in supermarkets and fast food restaurants they already have automatic ordering and payment machines that let them avoid workers for these tasks. So, the question that arises is: is it possible to avoid human labour in the check out process of a self-service restaurant? In that case it would certainly avoid the problems mentioned before as well as it will suppress the contact between worker and clients, which, unfortunately, in 2020 due to the pandemic of COVID-19 is an important thing to consider.

It is clear that the problem is not about the payment process, but the process of checking the food taken. As food dishes are not products with bar codes that can be scanned, the optimal way, the most automatic one, to recognize the food is through visual recognition. Then, the generic objective of this work is to present a system able to recognise food in tray images of self-service restaurants.

More technically we are talking of object detection and recognition/classification. The two of them are (and have been since years ago) important problems that the research area of Computer Vision tries to solve. Artificial Intelligence algorithms, which intuitively are algorithms designed to give certain "knowledge" to a machine in order to execute a concrete task the optimal way, have helped Computer Vision advance greatly in recent time.

The breakthrough of Deep learning and Convolutional Neural Networks in recent years has made an important contribution to the research in those fields. Based on advances and techniques from the state of the art, we aim with this work to present a novel model alternative and competitive to actual models in the concrete domain of food recognition.

More precisely, the main objective of this essay is to provide a model capable of detecting and recognising food dishes on self-service restaurant tray images. Given an image of a tray with multiple food dishes/pieces and the daily menu, we expect the model to detect a bounding box for each piece of food and return a label of the food class it has recognised. Fig. 1.1 is an example of what we expected as an output of the model.

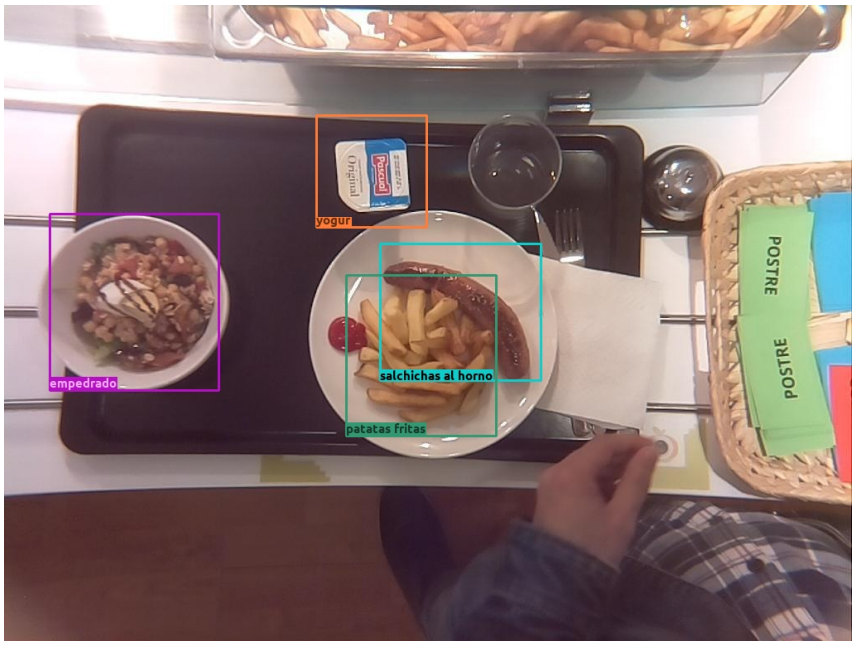


FIGURE 1.1: Expected output of the model.

The model proposed is a multimodal neural network which is based on two existing models that excel in concrete tasks: one in object detection and the other in image classification. Starting with two different modalities inputs, RGB images of food trays and categorical data of the daily menu, we use those models in order to extract valuable data and combine it in a final multimodal neural network.

Apart from the model, in this work we also present a model ensemble of our proposed model and a state of the art object recognition model. Our aim is to explore and improve food recognition in tray images, by proposing a combination of models that potentially improves the baseline food recognition model.

Finally, we would like to say that, although not being an objective for this work, we have developed these models with the intention that they could be applied to real restaurants in the near future.

1.2 Thesis organization

In the following sections of the Introduction we are going to introduce the study fields and domain of this work. In Chapter 2 we will explain the state of the art for each of the study fields and related work. In Chapter 3 we are going to define the terminology used along this work and explain the background and components of neural networks in order to understand our model.

A full explanation of the proposed model as well as the model ensemble mentioned before can be found in Chapter 4. In Chapter 5 we will see the results of the proposed model, results of some model variations tested and results of the model ensemble. Finally, in Chapter 6 we will give conclusions of the research done and in Chapter 7 we will discuss future work based on our contributions.

1.3 Deep Learning and Convolutional Neural Networks

As mentioned in the previous section, in order to detect and recognise food dishes on self-service restaurant tray images we have chosen the deep learning approach. Particularly, the proposed model is a multimodal neural network built upon a classification Convolutional Neural Network (CNN) and an object detection network, giving as a result a Deep Neural Network with multiple inputs of different modality.

In the State of the Art chapter (Chapter 2), we will see that the state of the art of our objectives rely on Deep learning and CNNs. This is due to the fact that in the literature it is proven that this type of models perform much better than the classical computer vision algorithms. For that reason we chose the Deep learning approach in order to solve the problem contemplated.

As said in advance before, our proposed model makes use of an existing powerful CNN model which performs very well in single instances images classification. Discussed and introduced in chapter 2, our chosen CNN for this work is *Inception-ResNetV2* (Szegedy et al., 2016). However, when presenting the model structure in section 4.1, we will see that the task this CNN does in our model is independent of the chosen model, making it model-agnostic. This independence makes the model adaptable to new and better classification CNNs that may appear in future years.

1.4 Multimodal Learning

The majority of the well-known deep learning models have only one input type of data. For example, the most common types are CNNs with images as single input or the Natural Language Processing models with text as single input. In some occasions it is possible to extract from the source of the data multiple features of different modality. The machine learning research area that studies how to integrate these different modality data inputs in order to learn intrinsic characteristic between them is called Multimodal Learning.

In our case, the source of the data let us extract not only the food tray images of the restaurant, but also the daily menu. Therefore, for each tray image, that contains several food dishes/pieces, we know the corresponding menu of that day and, that presents a correlation between them since, in general, the food dishes

from the image must be also in the daily menu. This connection between food in the image and food in the menu, simple for our human logic, may be not that simple for an algorithm. Multimodal Learning helps in that direction and thus the machine will be able to learn this and more complex connections between different modality inputs.

Multimodal Learning is an approach or technique for complex tasks in Computer Vision. It is not linked to a concrete domain, but used in multiple domains and for multiple tasks. Multimodal Learning can be used for combining image and audio inputs (Mroueh, Marcheret, and Goel, 2015, is an example for speech recognition) or other combination of different modalities inputs. For example in Jin et al., 2018, they used time series signals and clinical text to predict in-hospital mortality risk.

1.5 Object Detection and Recognition

It is known as Object Detection the task, related to computer vision and image processing, that consists of identifying and locating instances of a certain object class. Examples of applications are face detection (Sun, Wu, and Hoi, 2018), automatic image annotation (N. Murthy, Maji, and Manmatha, 2015) and pedestrian detection (Gavrila, 2000). In our context, the raw data we have is a single image of a tray with several food dishes or pieces, so the first step would be locating all instances of food in the image.

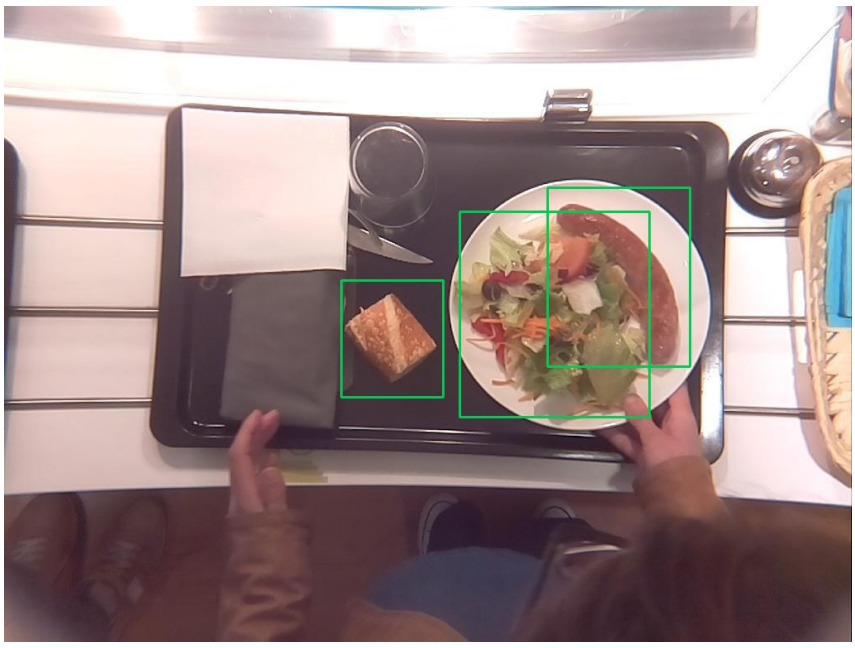


FIGURE 1.2: Instances of food located in the image

What we want is to detect objects of multiple classes. We expect the model to be able to locate the instances and to tell us the class of each instance detected. This combination of object detection plus classification is what is called object recognition, although sometimes in the literature the word detection is used referring to recognition. In the same line, in our context, we do not only want to locate all instances of food in the image, but also recognise the class of each one.

Fortunately, in the literature, there are methods that perform very well in the object recognition task. And so, nowadays, we do not need to detect and classify separately. In this work, we have chosen the You Only Look Once (YOLO) model version 3 (Redmon and Farhadi, 2018). A farther explanation can be found in the State of the Art chapter (Chapter 2) and in the Methodology chapter (Chapter 3).

More precisely, we will use as a baseline model the *YOLOv3* model implemented and trained for food detection by Bora, Bolaños, and Radeva, 2020. In their work, they used *YOLOv3* to detect initial food candidates and then apply filtering and menu based enhancement processes to them. Our proposed model is, so, an alternative to theirs that departs from the their food candidates detection model.

1.6 Food recognition

Food recognition, object recognition applied to food, is in its whole a current line of research due to the multiple applications it can provide and difficulties it presents. From a single food image, extracting information about calories and nutrition or the ingredients and recipe of the dish, are two of the most promising food recognition applications. In our context, the self-service restaurants, a good application of food recognition would be the independence of a human that checks the food chosen for the payment step, i.e. having a self-checkout system.

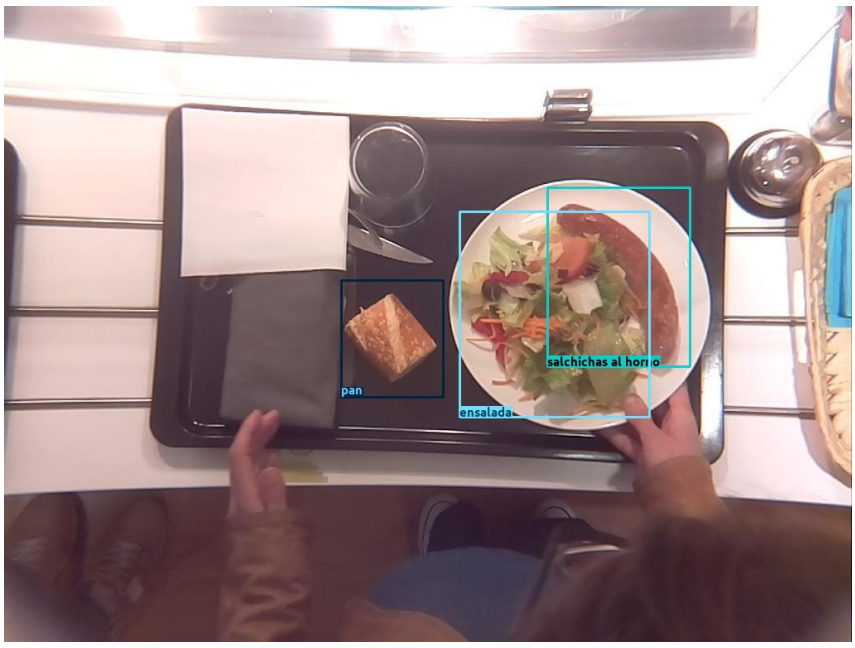


FIGURE 1.3: Instances of food located and recognized in the image

In regard of the difficulties, food is a vast domain of a lot of ingredients and cooking techniques that merged together create, with its varieties between culture, dish presentation and recipe variation, an almost infinity (if not infinity) quantity of different food dishes. That makes this study case a very challenging one, even taking a small portion of the food classes into account.

In Chapter 4, the Results Chapter, we will present the experiments done with our model and a dataset built from a real world self-service restaurant. We will also

see that the food considered in the dataset consists of 95 classes of food from the Spanish cuisine. The dataset used is not currently public and was build specifically for the food tray recognition task by Bora, Bolaños, and Radeva, 2020.

1.7 Contributions

The principle contributions that our project presents are:

- A novel multimodal food recognition model for tray images of self-service restaurants, which takes as an input a global RGB image of the food tray and a description of the daily menu.
- An ensemble of our model and the state of the art object detection YOLO model.
- Experimental results of our food recognition model and the ensemble model with data from a real world self-service restaurant of Spanish cuisine.

Chapter 2

State of the Art

In this chapter we will go through different works in the literature regarding different study fields related to our work. In section 2.1 we are going to discuss several relevant convolutional neural networks. In section 2.2 we will present approaches for multimodal learning. In section 2.3 we will see different models for generic object detection and recognition tasks, and in section 2.4 we are going to focus on the food domain and see related work to ours.

2.1 Deep Learning and Convolutional Neural Networks

Starting with Krizhevsky, Sutskever, and Hinton, 2012 work, Convolutional Neural Networks have provided a powerful tool for image classification problems. Later it was seen that the depth of a network was very important in order to achieve better classification results. *VGGNet* by Simonyan and Zisserman, 2014 and *GoogLeNet* or *Inception-v1* by Szegedy et al., 2014. These two, were the pioneers of the "very deep" networks, which motivated the research in deeper networks.

Residual connections and the *ResNet* architecture presented by He et al., 2015, produced another breakthrough in the field, as it was seen that introducing residual connections to deep networks made a significant improvement in training speed. The *ResNet152* was able to achieve a Top-1 Accuracy of 76.6% and Top-5 Accuracy of 93.1% on *ImageNet* (Deng et al., 2009). Later on, a refined second version of ResNet using Identity Mappings (He et al., 2016) were presented, *ResNet152V2*, and it achieved a Top-1 Accuracy of 78.0% and Top-5 Accuracy of 94.2% on ImageNet.

The GoogLeNet/Inception-v1 network was refined to a new network called *InceptionV2* with the introduction of batch normalization (Ioffe and Szegedy, 2015). The same network was later improved to *Inception-v3* (Szegedy et al., 2015) and to *Inception-v4* and *Inception-ResNet-v2* (Szegedy et al., 2016), the last one achieving a Top-1 Accuracy of 80.3% and Top-5 Accuracy of 95.3% on ImageNet and becoming one of the best in the recent years.%

Finally, a more recent network that has beaten the Inception-ResNet-V2 results is *NASNetLarge* (Zoph et al., 2017) with a Top-1 Accuracy of 82.7% and Top-5 Accuracy of 96.2% on ImageNet.

2.2 Multimodal Learning

Learning from data of different modalities is used in multiple domains and for multiple applications: recognising affect emotion from audio and visual sources Gunes and Piccardi, 2005, emotion recognition from video sources Kahou et al., 2015 or named entity recognition from images and text on social media posts Moon, Neves, and Carvalho, 2018.

There are multiple techniques explored in order to combine different modality data. Early fusions, also known as feature based, and late fusions, also known as decision based, were studied by Snoek, Worring, and Smeulders, 2005, and Gunes and Piccardi, 2005. Hybrid fusions (Atrey et al., 2010) as well as model ensembles (Dietterich, 2000) are other multimodal techniques.

More recently techniques are joint training methods based on deep neural networks. Joint training methods are neural networks which join features from different modalities inside the network and use the combined features to make predictions. The fusion operation can differ, but most common are addition, concatenation and multiplication. Some works that use this technique are Ngiam et al., 2011, and Bolaños et al., 2017.

Finally, another approach based also on deep learning is Liu et al., 2018. They proposed a method which first makes decisions for each of the modalities separately to then combine them in a multiplicative way. Their method is based on the idea that depending on the sample, possibly, not all the modalities are equally important.

2.3 Object Detection and Recognition

Regarding the object detection and recognition tasks, the first framework that provided competitive real-time results, based on Haar-like features, was named *Viola-Jones object detection framework* and was presented by Viola and Jones, 2001. Another early approach was *scale-invariant feature transform (SIFT)* published by Lowe, 2001, which is a feature detection and descriptor algorithm for image used in object recognition and other computer vision applications.

Another important machine learning approach is *Histogram of oriented gradients (HOG)* published by Dalal and Triggs, 2005. HOG is a feature descriptor algorithm based on counting the number of gradient orientations in local areas of an image.

One of the first deep learning approaches for object detection named *R-CNN* was Girshick et al., 2013, which was based on generating region proposals using a selective search algorithm, then applying a CNN for extracting features and finally classifying them with Support Vector Machines (SVM). Later, an improved version of *R-CNN* was published by Girshick, 2015, and called *Fast R-CNN*. As its name, *Fast R-CNN* is a faster version of the *R-CNN* achieved by applying the CNN to the whole image to extract features and then generate region proposals from the feature maps with a selective search method.

A more refined version called *Faster R-CNN* by Ren et al., 2015, improved even more the speed by changing the search method for a neural network to generate proposal regions. More recently appeared a fourth refined version called *Libra*

R-CNN (Pang et al., 2019) that aims to solve the imbalance in sample, feature and objective levels during the training process.

Finally, *You Only Look Once (YOLO)* is the name of a model that supposed a breakthrough in its first publication (Redmon et al., 2015) due to the great improvements in speed without losing accuracy. It was achieved because of a novel algorithm that uses a single convolutional network. More refined versions of the YOLO algorithm appeared one after another (*YOLOv2*, Redmon and Farhadi, 2016 and *YOLOv3*, Redmon and Farhadi, 2018) until the latest official to this date being *YOLOv4* (Bochkovskiy, Wang, and Liao, 2020).

2.4 Food recognition and Related Work

In the particular domain of food, the first works that appeared intended to tackle the binary classification problem of food/no food. Kitamura, Yamasaki, and Aizawa, 2009 was the first approach using machine learning and tested on a small dataset. Nowadays, with deep learning neural networks, it has been possible to get great results in several food datasets regarding the binary classification problem (McAllister et al., 2018).

About the food recognition problem, Bossard, Guillaumin, and Van Gool, 2014, is one of the firsts. They built a new dataset named *Food-101* and using a machine learning approach based on random forests, they got a top-1 accuracy of 56.4%. On the same dataset, a model published by Liu et al., 2016, and named *DeepFood* achieved a top-1 accuracy of 77%. Later on, Martinel, Foresti, and Micheloni, 2016, work, based on a CNN with two branches, one with a residual structure and another with a simple slice convolution layer, concatenated at the end, achieved a top-1 accuracy of 90.27% on *Food-101*, becoming the best performance on a public dataset.

In the context of restaurants, two works that need to be mentioned are Bettadapura et al., 2015, and Beijbom et al., 2015. The first one used extra information of the restaurants, such as the geo-location, in order to improve the recognition task. The second one was based on the idea that the same class food dishes tend to have the same visually aspect within the same restaurant, and so they proposed a restaurant-based CNN.

Now, in the same field of study as ours, food detection and recognition in food trays of self-service restaurants, one related work is Aguilar et al., 2017. The model they presented applies in parallel a CNN for semantically segmenting food (binary segmentation: food/no food) and on the other side they applied *YOLOv2* in order to detect and recognise food classes. Combining both, the segmentation helping to filter bounding boxes, plus using background subtraction and non-maximum suppression algorithm they get the final output. Tested in *UNIMIB2016* (Ciocca, Napoletano, and Schettini, 2017), a public dataset with 2000 images of canteen food trays with a total of 15 different food classes, they achieved a F1-score of 90%.

Finally, Bora, Bolaños, and Radeva, 2020, is probably the most related work to ours. Their contribution consists of a pipeline based on *YOLOv3* and several menu-enhancement methods built upon, in order to filter and adapt the YOLO results to different restaurant environments. They built a new dataset with food

tray images captured from a real self-service restaurant and achieved 83% precision and 93% recall detecting dishes present on testing tray images.

Chapter 3

Methodology

In this chapter we are going to define the terminology and theoretical background of the work. An expert reader can skip this chapter.

In section 3.1 we will define neural network basic concepts as well as some of the most important CNN layers, activations and optimizers. In section 3.2 we will explain the Non-Maximum Suppression (NMS) algorithm, a data processing algorithm commonly used after object detection models. Finally, in sections 3.3 and 3.4 we are going to explain the characteristics of two state of the art networks our model uses.

3.1 Terminology and theoretical background

The following explanations are basically based in the book *Deep Learning* by Goodfellow, Bengio, and Courville, 2016. We encourage this source of information for farther reading and more detailed theory.

3.1.1 Neural Networks Basics

Neural Networks are one type of machine learning algorithms inspired by the biological neural networks that constitute the human brain. We call machine learning algorithm to a computer algorithm capable of learning from experience. Mitchell, 1997, defines **computer learning** this way: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

In general, the task of a neural network consists of approximating some function f^*. One example of a common task for a neural network is **classification**. We call classifier to a neural network with a classification task. The goal of a classifier is to approximate the function $y = f^*(x)$, which maps an input x to class label y. In order to achieve it, the neural network defines a function $f(x;\theta)$ that takes an input x and a series of parameters θ. Through experience, the neural network updates the parameters θ giving a better approximation of f^*. From now on we will sometimes omit to write the parameters θ aiming for a simpler notation, but we want to remark that they are present.

We should understand **experience** as a series of iterations of a **dataset**, where a dataset is a collection of examples. An **example** is a collection of features quantitatively extracted or measured from an element of data. For instance, one image is an element of data that can be seen as a three-dimensional matrix of RGB pixel

values (collection of features); so the image interpreted as a matrix is an example of the dataset. The examples are inputs x of the function f mentioned before, that usually are vectors $x \in \mathbb{R}^n$ or matrices $x \in \mathbb{R}^m \times \mathbb{R}^n \times \mathbb{R}^3$. Note that each example x has associated a corresponding class label y, that is also included in the dataset.

The term *networks* in the name of this type of algorithm is due to the fact that the defined function f is a composition of several functions: $f(x) = (f^{(d)} \circ ... \circ f^{(2)} \circ f^{(1)})(x)$, where $d \in \mathbb{N}$ is called **depth** of the network. Each of these intermediate functions are called **layers**, so $f^{(1)}$ is the **first layer** or **input layer**, $f^{(2)}$ is the **second layer** and $f^{(d)}$ is the **last layer** or the **output layer**. The layers that are not the first nor the last, are called also **hidden layers** because we do not interact directly with them and the middle-outputs and weights are hidden to us.

Note that we are not defining domains for the layer functions, since it is not necessary that all the layers have the same domain. The only requirement is that the composition of layers is well defined in terms of domain dimensions and correspondence. Actually, in the literature there are multiple predefined layers that have different effects on the network. We will see some of the most important in the next subsection.

More terminology, we call **tensors** to the n-dimensional matrices that are inputs and at the same time outputs of layers. Each of the layers has its own parameters array θ, also known as **weights** and **biases**. A neural network with more than one hidden layer is called **deep neural network**.

The term *neural* stands for the resemblance to the neural structure in biology. We can visualize a hidden layer as an array of **units** that act in parallel and each represents a vector-to-scalar function. That way, each unit receives multiple inputs from previous units and computes a single output value that is, at the same time, an input for the next array of deeper units. This behavior is, in general, the same for all kinds of layers, the difference among them is the intrinsic operation applied.

3.1.2 Layers

In the literature there have been a large number of layers studied. In this subsection, we are going to explain some of the most common that are also important in our model.

Fully Connected

The **Fully Connected Layer** or **Dense** layer is the most straightforward and principle of the layers. Given an input tensor and a number of units, it computes the following operation:

$$O = I \cdot W + B,$$

where O is the output tensor, I is the input tensor, $\cdot$ is the product of matrices, W is the weights matrix of dimensions (input tensor dimension, number of units) and B is the bias tensor of length number of units. Depending on the size of the input, weights and biases can be multi-dimensional matrices. The weights and biases values are initialized as random but are updated at each batch iteration.

Taking the weights matrix as a variable, remember that our goal is to optimize the weights values through iterations in order to get the desired output, we can see that the operation is a linear function, being the bias vector the independent term. Using bias, in general, is optional but it helps to adjust better the output.

Note that in the Fully Connected layer, all the input tensor values are connected to all the units giving a large number of weight values needed and the more number of parameters a network has, the slower it becomes. Reducing the number of weights to train, and so the computing time, is one of the purposes of other layers. One example of layer with a lot of weights is the Convolutional layer, a layer appropriate for image inputs, where the dimension of the input is considerably large.

Convolutional Layer

The **Convolutional Layer** is quite similar to the Fully Connected layer, in the sense that the operation it computes has the same structure:

$$F = I * K + B,$$

where F is the output tensor named **feature map**, I is the input tensor, $*$ is the convolution operator, K is the weights matrix of small dimensions named **kernel** and B is the bias tensor. The weights and biases values are initialized as random, but are updated at each batch iteration.

The principle difference is the operation between the input and the weights matrix or kernel. The **convolution** operation, intuitively and in this discrete context, consists of a local dot product of these two matrices. It can also be seen as a local weighted average of values in the input matrix being weights the values in the kernel matrix. The locality means that the convolution is done element-wise, taking the input and kernel matrices of small dimensions and with the element at its center.

The formal definition of the convolution operation in our context, being a three-dimensional image I our input, is:

$$F(i,j) = (I * K)(i,j) = \sum_{c=1}^{3} \sum_{di=-m}^{m} \sum_{dj=-n}^{n} I(i+di, j+dj, c)\ K(di, dj, c),$$

for each pixel (i,j) in the image. The values m and n are the height and width of the kernel matrix, and c represents the channel of the image and kernel. The kernel matrix K in this case is also three-dimensional.

There are two more important parameters that take part in the convolution layer: **stride** and **padding**. The stride is the convolution frequency between pixels, in another words, the number of pixels - 1 skipped between each convolution. So we expect two integers, one for the height stride and the other for the width stride.

Padding means whether or not to frame with zeros the input image when computing the convolution for pixels in the edges. In case of not using padding, the values in the right or bottom edge, which make the centered kernel not to fit for a proper convolution, are dropped. Note that using stride and padding may result in an output of smaller dimensions.

Embedding

The **Embedding Layer** is, commonly, the first layer of a network with a categorical input and its goal is to map positive integers, indexes of categorical classes, to dense vectors of fixed size in an Euclidean space. This process is called **entity embedding** (Guo and Berkhahn, 2016), and it can be applied to text or indexed categorical data. Once the categorical data is transformed to vector, then it is ready for being input of other layers. The parameters needed for this layer are the input dimension (vocabulary size) and the output dimension (dimension of the dense embedding).

Instead of the embedding one could think that a good option is to convert the categorical data to one-hot encoding vectors, in other words, create a Euclidean space of dimension size of the vocabulary. That would work, but with a big vocabulary, the space would be very large requiring a lot of parameters to learn. Entity embedding not only reduces memory usage and number of parameters to learn, but also, and more importantly, maps similar categorical data closer in the embedding space revealing intrinsic features of the categorical variables. This last property helps the network in speed and performance.

In essence, the embedding layer is a matrix of dimensions: vocabulary size $\times$ embedding size. Then, for each input index, the embedding returns its corresponding row. The vocabulary size is a fixed parameter and the embedding size is a hyperparameter that varies from different models and needs to be adjusted, but must be a value between 1 and the vocabulary size. Finally, the embedding vectors are initialized as random, but are updated during training.

Merge Layer

The **Merge Layer** is an important layer for multimodal learning and, in general, networks with multiple inputs, since it is the key to combine them. It supports various merge operations, such as: summation, multiplication, concatenation, maximum, minimum, average, subtraction or dot product. Depending on the operation, the output will be a tensor with the same dimensions as the input tensors or a single output value.

Time Distributed Layer

The **Time Distributed Layer** is wrapping layer that allows the network to apply any layer to every temporal slice of an input. This layer can only be used with three or greater dimensional inputs, and assumes the temporal slice is at axis 1. The time distributed layer is useful for video inputs or multiple related images inputs.

3.1.3 Activations

Following the biological influence in the design of these networks, it can be seen that each of the units or neurons specializes in concrete features of the inputs. It can be color, edges, corners, forms... the deeper the neuron, the more complex its task is. Although these specializations are hidden or not easily understandable for us humans, exist some software that gives us a little intuition of the neurons behaviour. One example of software for a simple CNN is the work of Wang et al., 2020 and their *CNN Explainer Live Demo*.

The purpose of this introduction is that due to the neurons specialized behaviour, depending of the input, there will be neurons more important than the others, so it is interesting to enhance the output of those and neglect the output of the others. We can achieve it using an activation function after a layer.

The most important activation functions are the following:

The function defined as

$$f(x) = x^+ = max(0, x)$$

is called, in the artificial neural networks context, the **rectified** function and the unit that computes it a **rectified linear unit (ReLU)**. We can see that the rectified function sets negative inputs to zero while leaves positive values untouched.

Another common activation function is the **hyperbolic tangent (tanh)**

$$f(x) = tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

that, unlike ReLu, it has a range of $(-1, 1)$ and the particularity of *pushing* positive values to 1 and negative values to -1.

The **logistic function** or **sigmoid**

$$f(x) = \frac{1}{1 + e^{-x}}$$

has a range of $(0, 1)$ and that makes it useful for binary classification problems, because the output value can be interpreted as a probability.

The **softmax function**

$$f(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \text{ for } i = 1, \ ... \ , \ K \text{ and } x = (x_1, \ ... \ , \ x_K) \in \mathbb{R}^n$$

is a generalization of the sigmoid in a multi-dimensional context. It is commonly used as the last layer in a multi-class classification neural network, since the values of the output vector are values between 0 and 1 that sum together 1. Thus, we get for input x the probability of being an element of classes $1, \ ... \ , \ K$.

3.1.4 Optimizer and Cost Function

An **optimizer** algorithm together with a suitable **cost function** are the elements in charge of the learning and so, the weights update. A cost function measures the performance of the model and the optimizer updates the model parameters in order to minimize the error obtained from the cost function, which essentially measures the error between the model prediction and the expected value (called **ground truth**). The cost function is also called **loss function** or, simply, loss.

Cost functions

There are several loss functions, each of them more appropriate for a certain problem. Next, we will see the three more common loss functions.

The **Mean Squared Error (MSE)**

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2,$$

where n is the number of data, y_i is the i-ground truth value in the data set and $\hat{y}_i$ its corresponding prediction. This is a good loss function for regression problems, as we want an error based on quantitative data.

For classification problems, where the data is qualitative and the predicted outputs are probability values between 0 and 1, the loss function recommended is **Cross-Entropy**, also known as logarithmic loss. This loss function is suitable because the error increases if the predicted probability is close to zero for the ground truth class.

In binary classification, where the number of classes is two (0 or 1),

$$\text{Binary Cross-Entropy} = -(y \log p + (1-y) \log (1-p).$$

In the formula, y denotes the true class label (0 or 1) and p is the probability of prediction being of class 1 given by the model. So, if the ground truth class is 1, the probability of the prediction is p, whereas if the ground truth class is 0, the probability of the prediction is $1 - p$.

For multi-class classification, the cross-entropy loss is calculated by the sum of the separate loss for each class label,

$$\text{Categorical Cross-Entropy} = -\sum_{c=1}^{M} y_c \log p_c,$$

where M is the number of classes, p is the predicted probability of class c and y_c is a binary indicator that is 1 if class label c equals to the ground truth label or is 0 otherwise.

Note that for binary classification we expect a single output probability value, so sigmoid is usually used as the last layer activation, while for multi-class classification, we expect a vector output with the probability distribution for all the classes, thus softmax activation is suitable for the last layer. Finally, say that the formulas mentioned before apply for a single observation. In order to have a loss value for all the model, we need to average the observations losses.

Stochastic Gradient Descent

The basic algorithm and predecessor of newer and improved optimizers is **Stochastic Gradient Descent (SGD)**. SGD is a variation of the classical numerical method Gradient Descent. The difference between them is that while classical Gradient Descent uses all the training dataset to calculate the gradient of the loss function and then steps toward the opposite direction, SGD divides the dataset in small batches of data and for each batch computes the gradient from a random example in the batch and steps toward the opposite direction of the gradient.

We can see that SGD is able to speed up the gradient calculation and make more steps for one step of the classical Gradient Descent. It is true that the Gradient

Descent method is more accurate, but through multiple iterations, also known as **epochs**, SGD converges much faster.

A compromised variation between classical Gradient Descent and SGD is the **Mini-batch Gradient Descent**, which instead of picking a random example from the small batch as SGD does, it computes the average of gradients for all or a subset of examples in the small batch.

The equation that SGD uses for updating the weights of the neural network is:

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta C(\theta_t; x, y),$$

where θ is a parameter of the model (could be weight or bias), t denotes the number of iteration, η is the **learning rate**, a small factor that determines the size of the step and ∇ is the gradient of the cost function $C(\theta_t; x, y)$ at point (x, y) and parameter θ_t. Depending on the cost function, the variables x, y will be ground truth and prediction values (for MSE), ground truth label and prediction probability (for binary cross-entropy) or ground truth and predictions probability distribution vectors (for categorical cross-entropy).

Momentum

Momentum is known as a variation in the weights update equation of SGD that contributes with faster convergence in the algorithm. This variation consists of adding an extra term in the equation formed by the velocity in the previous step multiplied by a constant factor γ, usually set to 0.9.

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta C(\theta_t; x, y) + \gamma \cdot v_t$$

The velocity v_t is the update of the previous iteration.

$$v_t = -\eta \cdot \nabla_\theta C(\theta_{t-1}; x, y) + \gamma \cdot v_{t-1}$$

The intuition behind this variation is that the momentum term increases the step toward the optimal minimum of the cost function if the previous update was significant, speeding the convergence and preventing to get stuck in local minimums. The γ factor helps giving more importance to recent velocity values in time and less to old velocity values. For a detailed explanation of why the Momentum variation works, we recommend to read the Goh, 2017, article.

Adaptive Moment Estimation

The next and last optimizer we are going to explained in this essay is called **Adaptive Moment Estimation (Adam)** (Kingma and Ba, 2014), one of the state of the art optimizers. Based on SGD, Adam also uses Momentum and Adaptive Learning Rate.

A common practice in machine learning problems is to adjust the learning rate during training in order to speed up the convergence. In general, a too small learning rate makes the model learn slowly, while a too large learning rate makes the model diverge. One solution is to use a **learning rate schedule**, a pre-defined schedule of learning rates. The most used schedulers are the time-based decay

(the learning rate decays at each epoch by a decreasing factor), the step-based decay (the learning rate decays by a factor every few epochs), the exponential decay (the learning rate drops at exponential rate each epoch) and the cyclic scheduler (the learning rate rotates every few epochs in a pre-fixed list of values).

The main inconvenience of learning rate schedules is that it needs to be defined manually before the training and, as the learning rate is an hyperparameter, optimal learning rate values depend on the problem and the model implemented. Some optimizers started to introduce ways to adjust automatically the learning rate by some heuristics. This technique is called **Adaptive Learning Rate** and it only requires a initial learning rate value. RMSProp (Dauphin et al., 2015) and AdaGrad (Duchi, Hazan, and Singer, 2011) are two SGD-based optimizers that include adaptive learning rate.

As they say in the Adam paper, Adam optimizer is created from the advantages of RMSProp and AdaGrad:

> "Our method is designed to combine the advantages of two recently popular methods: AdaGrad (Duchi, Hazan, and Singer, 2011), which works well with sparse gradients, and RMSProp (Tieleman and Hinton, 2012), which works well in on-line and non-stationary settings".

Finally, the Adam weight update equation is

$$\theta_{t+1} = \theta_t - \frac{\eta \cdot \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

where η is the learning rate (recommended value in the paper is 0.001), ϵ is a small term preventing division by zero (usually 10^{-7}) and

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \ , \ \hat{v}_t = \frac{v_t}{1-\beta_2^t} \ ,$$

where

$$m_t = (1-\beta_1)g_t + \beta_1 m_{t-1} \quad \text{and} \quad v_t = (1-\beta_2)g_t^2 + \beta_2 v_{t-1} \ .$$

In these last equations, g denotes the gradient $\nabla_\theta C(\theta_t; x, y)$, β_1 denotes the first momentum term (recommended value is 0.9) and β_2 denotes the second momentum term (recommended value is 0.999).

In the original paper (Kingma and Ba, 2014), there are more details regarding the update rule, the upper boundaries of the stepsize and the convergence proof.

Backpropagation

Last but not less important, it remains from this section computing the cost function gradients. Let us assume that in the neural network we have n weights and biases we want to adjust. The gradient of the cost function is

$$\nabla C(w_1, b_1, ..., w_n, b_n) = (\frac{\partial C}{\partial w_1}, \frac{\partial C}{\partial b_1}, ..., \frac{\partial C}{\partial w_n}, \frac{\partial C}{\partial b_n}).$$

Nowadays, most state of the art neural networks are significantly deep, meaning that the number of parameters to train is huge (order of millions). **Backpropagation** is an algorithm that, making use of the derivative chain rule, computes the partial derivatives of the cost function in a simply and fast way: from the end and backwards to the beginning reusing previous calculations.

Given a simple neural network with several Fully Connected hidden layers, being L the total number of layers, the equation in each layer number $l > 1$ is

$$a^{(l)} = \sigma(w \cdot a^{l-1} + b),$$

where $a^{(l)}$ is the output vector of layer l, $a^{(l-1)}$ is the output vector of the previous layer, σ is the activation function for layer l, w is the weights matrix of layer l and b is the biases vector of layer l.

Now, given a weight or bias from the final layer, denoted $w^{(L)}$, the partial derivative of the cost function with respect to $w^{(L)}$ is

$$\frac{\partial C}{\partial w^{(L)}} = \frac{\partial C}{\partial a^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial z^{(L)}}{\partial w^{(L)}},$$

where

$$z^{(l)} = w \cdot a^{l-1} + b.$$

Note that the partial derivatives $\frac{\partial C}{\partial a^{(L)}}$ and $\frac{\partial a^{(L)}}{\partial z^{(L)}}$ are common for all the weights or biases of layer L.

If we assume that the weight or bias w we want to update is from layer $(L-1)$, denoted $w^{(L-1)}$, we have

$$\frac{\partial C}{\partial w^{(L-1)}} = \frac{\partial C}{\partial a^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \cdot \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \cdot \frac{\partial z^{(L-1)}}{\partial w^{(L-1)}}.$$

Note that the partial derivatives $\frac{\partial C}{\partial a^{(L)}}$ and $\frac{\partial a^{(L)}}{\partial z^{(L)}}$ appear also in the previous derivative with respect to $w^{(L)}$, meaning that they are already calculated. Note also that the derivatives $\frac{\partial a^{(L-1)}}{\partial z^{(L-1)}}$ and $\frac{\partial z^{(L-1)}}{\partial w^{(L-1)}}$ are common for all the weights or biases of layer $(L-1)$.

Recursively, and starting from the output layer, the backpropagation algorithm gives us a way to compute all the cost function partial derivatives reusing previous calculations.

3.1.5 Regularization

One of the common problems deep learning models have is **overfitting**. We say that a model is overfitted if it performs very well during training but distinctly not that well in test data. This means that the model has learned too much in detail the features of examples presented in the training process that then it is not capable of extrapolating the learned connections to new images. Overfitting is quite inevitable in deep learning especially when the dataset is small, containing few examples per class. Despite this drawback, there are techniques that help preventing it by making slight modifications to the model in order to help the model generalize better and, thus, improve performance on unseen data. This process is called **Regularization**.

Weight Decay

The **weight decay** technique, also known as L_2 regularization, consists of adding an extra term to the cost function. This term is the L_2-norm of the weights matrix multiplied by a factor λ, an hyperparameter, that determines the strength of regularization applied.

$$C(w, b) + \frac{\lambda}{2}\|w\|^2$$

The intuition behind is that without the regularization term, the model can optimize the cost function without restrictions to weights giving as a result large weights values (positive or negative) that makes the model function excessively complex. By adding the L_2-norm of the weights matrix in the objective function we force the model to not only optimize the weights in order to minimize the cost function, but also to do it with small weights values. Interpreting the measure of complexity of a linear function $f(x) = w^T x$ by some norm of the weights matrix, in this case $\|w\|^2$.

Dropout

The **Dropout** technique (Srivastava et al., 2014) is possibly the most used technique in deep neural networks for regularization. Defined as a layer, and given a dropout rate parameter, it randomly *drops out* inputs of the next layer. Equivalently, sets inputs of the next layer to 0. The dropout rate parameter is the rate of units dropped out randomly at each iteration.

Dropout makes the training process less straightforward and more random or noisy, forcing the nodes of the next layer to give attention to different inputs each iteration. This makes the nodes not to rely too much on few inputs and instead rely on many inputs and making the model more robust.

Batch Normalization

Deep neural networks are in essence the composition of several functions with multiple parameters. During training, each of these parameters is updated by the gradient. Once the gradient is computed all the parameters are updated simultaneously and under the assumption that the others remain constant. In a model with a lot of hidden layers, and so parameters, it can result in unexpected behaviours, such as **vanishing gradients** or **exploding gradients**, gradient decreasing exponentially to 0 or gradient increasing exponentially to infinity, respectively. These problems make the model unstable and unable to learn properly.

Batch Normalization is a solution proposed by Ioffe and Szegedy, 2015. This solution consists of a layer in the network that standardizes its inputs. In this context also known as normalization, standardization means rescale data in order to have a mean of 0 and a standard deviation of 1. Given a d-dimensional input, the transformation that Batch Normalization layer applies is

$$\hat{x_i}^{(k)} = \frac{x_i^{(k)} - \mu_B^{(k)}}{\sqrt{{\sigma_B^{(k)}}^2 + \epsilon}}$$

and

$$y_i^{(k)} = \gamma^{(k)} \hat{x}_i^{(k)} + \beta^{(k)} ,$$

where $k \in [1, d]$, B refers to a concrete mini-batch of data of size m, $i \in [1, m]$, $\mu_B^{(k)}$ and ${\sigma_B^{(k)}}^2$ are the mean and variance respectively of dimension k, ϵ is a small constant, $\gamma^{(k)}$ is a learned scaling factor (optional and initialized as 1) and $\beta^{(k)}$ is a learned bias factor (optional and initialized as 0).

Batch Normalization has become a useful regularization technique to use instead or together with Dropout. The weights standardization that Batch Normalization provides, makes the network more robust to random initialization of weights and higher learning rates, consequently reducing the training time.

3.2 Non-Maximum Suppression

The output of Object Detection algorithms is, in general, a list of proposal bounding boxes with the corresponding list of classes labels and confidences scores for each bounding box. Over the hundreds of candidates bounding boxes, due to how the algorithms generate proposal bounding boxes, there are many repeated that envelop the same object in the image. In order to filter and get a desired output of one bounding box per object, it is commonly applied a **Non-Maximum Suppression (NMS)** algorithm.

Given a list of proposal boxes B, the corresponding list of confidence scores S and an overlap threshold T as inputs, the algorithm returns a filtered list of proposals R. The algorithm is the following:

1. Initialize R as an empty list.
2. Take the bounding box with the highest confidence score from B. Remove it from B and add it to R.
3. For each of the remaining proposals in B, compute the Intersection over Union (IOU) against the selected bounding box. If the IOU is greater than the threshold T, remove that proposal from B.
4. If list B is not empty yet, go back to step number 2.

The **intersection over union (IOU)** metric between two regions A and B in a plane is defined as

$$IOU = \frac{A \cap B}{A \cup B}$$

and measures the overlap between regions A and B.

3.3 You Only Look Once

As already introduced before in this work, You Only Look Once (YOLO) is a neural network for object detection and recognition. Presented originally by Redmon et al., 2015, it supposed a breakthrough in object detection study field because it was able to perform very well using a single neural network, while previous algorithms used a pipeline composed of multiple separate steps. Authors say in the paper: "We reframe the object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilites".

In this section we will explain the intuition of this network, for farther details we strongly recommend to read the original papers.

YOLO divides the input image into a grid of $S \times S$ cells. For each object in the image, it is said that the cell *responsible* for predicting it is the cell that contains the center of the object. Each cell in the grid makes B bounding boxes predictions and C class probabilities for the cell. The bounding box is represented as $(x,\ y,\ w,\ h,\ c)$, where (x, y) are the coordinates normalized to $[0, 1]$ that denote the center of the box, w and h are the weight and height of the bounding box also normalized to $[0, 1]$ and c is what they call the confidence score.

The confidence score is defined as $Pr(Object) \cdot IOU(pred, truth)$, where $P(Object)$ is the probability of the object and $IOU(pred, truth)$ is the intersection over union metric between the prediction and the ground truth.

As said before, each grid cell makes C class predictions that are returned as class probabilities computed with the conditional probability of object being of class i if the cell is responsible for predicting it: $Pr(Class_i \mid Object)$. So the output with the predictions is encoded as a tensor of dimensions $S \times S \times (B \cdot 5 + C)$.

In order to have class-specific confidence scores for each bounding box, they multiply the conditional class probabilities and the confidence of the bounding box:

$$Pr(Class_i \mid Object) \cdot Pr(Object) \cdot IOU(pred, truth) = Pr(Class_i) \cdot IOU(pred, truth).$$

With the maximum of the class confidence scores, they finally obtain a single confidence score and class prediction for each bounding box. After that they apply a Non-Maximum Suppression algorithm to filter the number of final bounding boxes, and obtaining the final detections.

Since the first version published, there have been continuous modifications to different parts of model that have resulted in various versions, each faster and better in performance than the previous one. For farther research, the YOLO versions to this day are: *YOLOv1* (Redmon et al., 2015), *YOLOv2* (Redmon and Farhadi, 2016), *YOLOv3* (Redmon and Farhadi, 2018) and *YOLOv4* (Bochkovskiy, Wang, and Liao, 2020). Being version 4 the latest official version.

3.4 InceptionResNetV2

Inception v1 published by Szegedy et al., 2014, was the first of a series of promising classification convolutional neural networks through the years of research in deep learning. The premise of this network is that, considering that the kernel size of convolutional layers is relevant for the network performance, there are images which have the crucial information for classification distributed globally, so a larger kernel is preferred, and other images with the information distributed more locally, and so a smaller kernel is more suitable.

Moreover, very deep networks are more likely to overfit and are computationally expensive. Taking all into account, the solution presented with the Inception v1 network was to make the network "wider" instead of "deeper". Intuitively, it means that the convolutional layers of the network are substituted with a block containing a series of parallel convolutional layers with different kernel sizes. Those outputs are then concatenated and sent to the next inception block.

After, two posterior versions of this network, *Inception v2* and *Inception v3*, were published by Szegedy et al., 2015. Those versions increased the accuracy and reduced the computational complexity of the original network.

Years later, *Inception v4* and *Inception-ResNet v1* and *v2*, where presented in the same paper by Szegedy et al., 2016. Focusing on the last ones, Inception-ResNet v1 and v2t, are networks inspired by the ResNet and its residual blocks (He et al., 2015).

Residual blocks are called to blocks of layers of the network, that instead of strictly feeding the output of one layer directly to the next one, they additionally feed into a fusion 2 or 3 layers away. Fig. 3.1 depicts it.

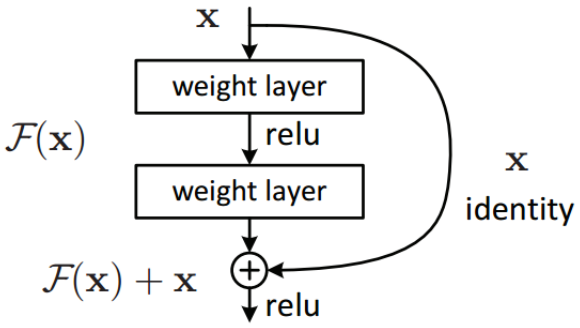


FIGURE 3.1: Example of a residual block.

It is believed that residual blocks produce many benefits in a neural network. Their skip connections let the network the possibility to reduce the computational complexity skipping dynamically certain layers. Residual blocks also help during the backpropagation algorithm to reduce the vanishing gradient problem and let larger gradients to reach to early weights of the network, thanks to the skip connections.

Then, Inception-ResNet v1 and v2 builds the mentioned before inception blocks with skip connections, and so, becoming an hybrid inception-residual block. These versions of the Inception network were able to achieve better accuracy results in lower epochs of training. The difference between versions 1 and 2 is only the previous Inception network they are based on, Inception v3 and v4, respectively.

Chapter 4

Multimodal Food Tray Recognition

In this Chapter, in section 4.1 we are going to explain the architecture of the principle model we present, named *Multimodal Food Tray Recognition* (MFTR). In section 4.2 we will briefly explain how we have trained the network and in section 4.3 we are going to present the second model proposed, which is an ensemble of YOLO and MFTR models.

4.1 Proposed model

In self-service restaurants there are two sources of information we can make use of: food tray images and the available dishes for that day (the daily menu). Our model uses both of them and expects them to be in the format: RGB-image and sentence with food classes as words for the menu. "31 2 68 94 17 55" is an example of a good menu input, where order is not important and each number denotes a class of food in the menu for that day. Although these two are the true inputs of the model, we will see that there is a certain adaptation of the data before the multimodal structure that leads to a third input, the yolo predictions.

Fig. 4.1 is a general representation of our model, where we can see that there are three differentiated parts of our model that summarized do the following:

1. **Food candidates detection**: Receiving a food tray image as input, this process applies a pre-trained Food Detection network (YOLO) to the image and then applies the NMS algorithm (with relaxed thresholds) in order to filter repeated bounding boxes.

2. **Features vectors extraction**: Using a pre-trained classification CNN (InceptionResnetV2), this process generates a feature vector for each of the proposed bounding boxes, which contains visual features extracted from the bounding box.

3. **Multimodal neural network**: This is the main process, which receives as inputs the feature vectors from part 2, the YOLO predictions form part 1 and the menu from the raw inputs. This multimodal neural network combines these inputs in three different fusions obtaining two outputs. Finally, the predictions of output 1 and 2 are merged into a single list of predictions, the predicted background is discarded and a final more restrictive NMS algorithm is applied to obtain the final output.

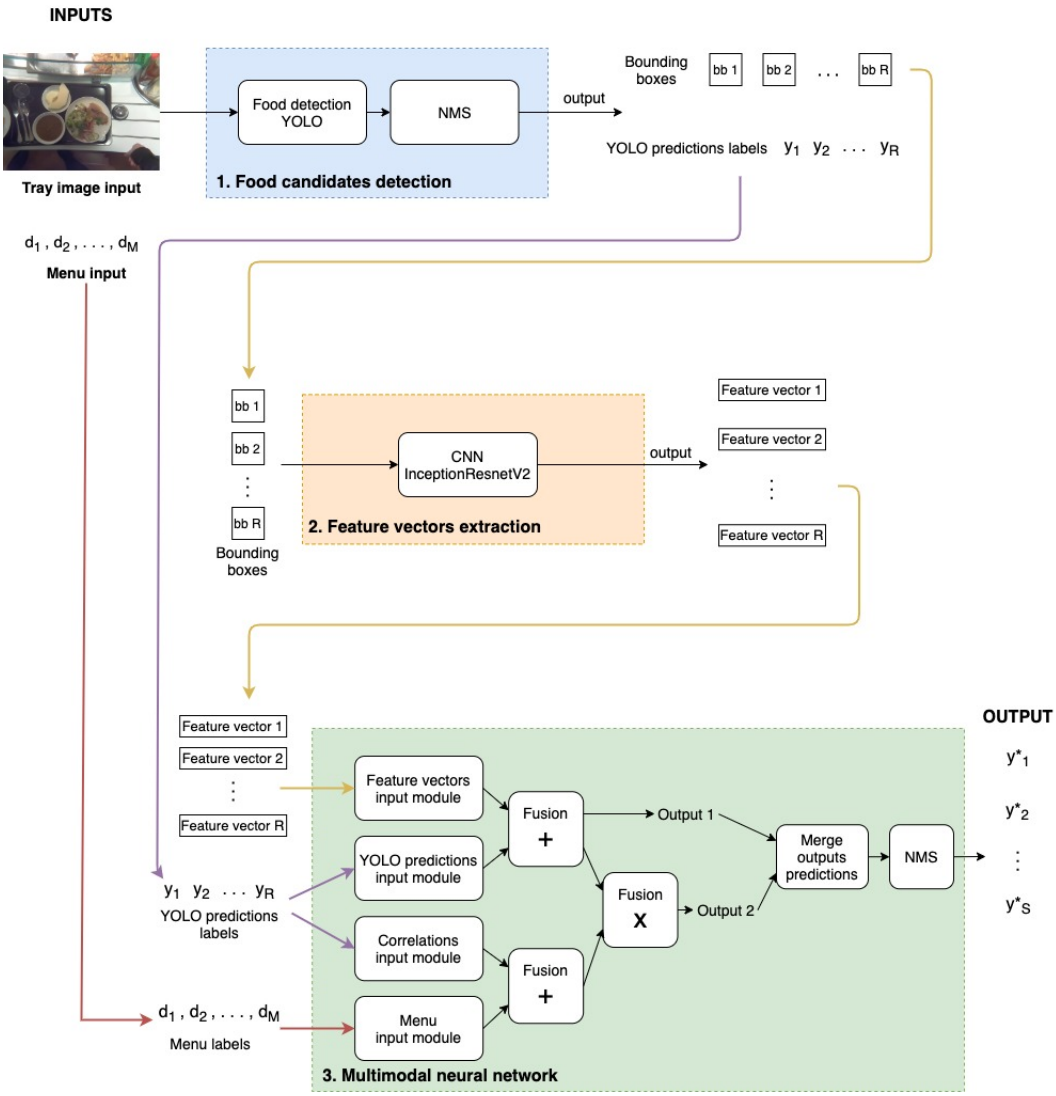


FIGURE 4.1: Multi-modal Food Tray Recognition architecture.

In the next subsections we are going to explain in more detail each of the three parts of the model.

4.1.1 Food candidates detection

Our model aims to improve the performance of food tray detection integrating data from different modalities. We do not present our model as an alternative to the powerful state of the art object detection models in food domain, in fact our models benefits from them and is build upon them. This is the case of YOLO, a powerful state of the art object detection model, that we use in order to extract raw bounding boxes from the input image.

In fact, we use the *YOLOv3* version pre-trained with images and classes of the food domain. Despite being available the forth version of YOLO on the date of publication of this work, when we started the state of the art version was the third, and so it is the one we took.

With a certain threshold, YOLO returns the list of bounding boxes, class labels and confidence scores, which have a greater confidence than the threshold value.

Then, we apply the NMS algorithm twice with different *IOU* threshold values. The first NMS only filters overlapped bounding boxes with the same class label and the second NMS filters overlapped bounding boxes with strictly different class labels. The reason why we apply two versions of NMS is because the *IOU* threshold used is different, the first one inferior than the second, i.e. the first one more restrictive than the second.

Note that, two bounding boxes of the same class need to be less overlapped than two bounding boxes of different classes in order to take both into account. In section 4.2, the section about training our model, we will discuss the threshold values used and the reasons of their choice.

After the NMS, the outputs received are the final list of bounding boxes and the class label that YOLO has given to each of the bounding boxes. From now on, we will refer to these classes labels as **YOLO predictions (YP)**. In this step we also obtain the confidence score from YOLO for each of the predictions. Those are no longer used in the model, but will be useful for the ensemble model proposed on section 4.3.

In case that at the end of this process there are more than 20 bounding boxes / predictions for an image, we keep the top 20 with highest confidence scores and discard the rest.

4.1.2 Feature vectors extraction

This process simply transforms each of the bounding boxes to feature vectors. In order to achieve it, we use a pre-trained CNN, in this case we have used the *InceptionResnetV2*, a state of the art neural network which performs very well in image classification problems. Having dropped the last layer of the CNN, the classification layer, the output that results when predicting is a tensor which contains intrinsic visual features of the image in question. So, repeating this process for each of the bounding boxes, we get a list of feature vectors, which is one of the inputs of the multimodal neural network.

Although YOLO is capable of doing very good predictions, it is not better than a specialized CNN in classification when applied to single-object images. Then, the reason of using a CNN is to get a "second opinion" based on visual properties for each of the candidate bounding boxes.

4.1.3 Multimodal neural network

This is the most important part of the three, where the essence of the multimodal learning is applied. Once in this point, we have three sources of data: the feature vectors extracted from the bounding boxes images (visual modality), the predictions made by YOLO for each of the candidate bounding boxes (categorical data encoded as text: text modality) and the food classes that were in the menu the day of the image (categorical data encoded as text: text modality).

Each of these three inputs is introduced to the model in different ways, through what we have called **input modules**. Those are a series of layers that receive the

input and prepare it for the features fusion. Since we have visual and text modalities, the features module is significantly different from the other ones, that are at the same time slightly different between them. Note that the YOLO predictions input leads to two separate input modules, that is because we want to interpret this input in two different ways, expecting to add more value to the model. In the following paragraphs we are going to look into the four input modules.

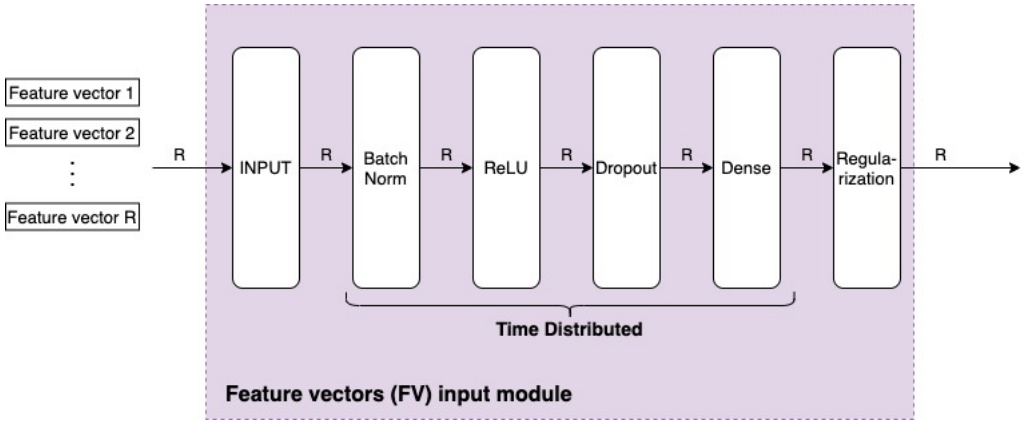


FIGURE 4.2: Feature vectors (FV) input module structure.

Starting with the **feature vectors input module**, Fig. 4.2, the only of visual modality type, we can see that it is a very straightforward series of layers. The feature vectors are introduced through an input layer and led to Batch Normalization, ReLU activation, Dropout, Dense and Regularization layers, wrapped with Time Distributed layers in order to apply these layers through the time dimension, i.e. each feature vector separately. The number of units parameter of the Dense layer is a fixed value that determines the number of features each vector will have before the fusion layer. This value is common with the other modules since we want the inputs of the fusion layers to have the same dimensions.

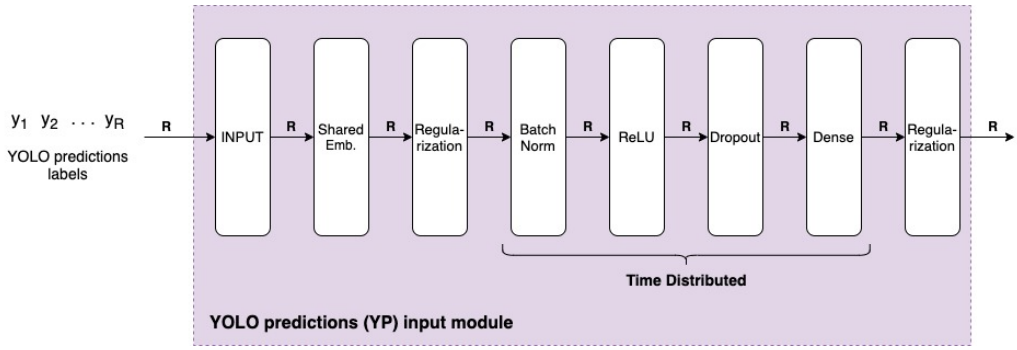


FIGURE 4.3: YOLO predictions (YP) input module structure.

The **YOLO predictions input module**, Fig. 4.3, is the first of the text modality ones. The YOLO predictions input format is a sentence of the class labels, numerical indexes of the classes, separated by blank characters. The number of class labels is the same as the number of feature vectors, since they are one to one related. "6 13 68 94" is an example of YOLO predictions input, where 6 is the YOLO-predicted class of the first feature vector (i.e. bounding box), 13 corresponds to the second and etc.

Through an input layer, the YOLO predictions are inserted to the module and led to an Embedding layer, which encodes the input into a tensor of dimensions $(batch\ size, max\ frame, embedding\ size)$, where *batch size* is the number of examples to train/predict in the batch, *max frame* is the maximum number of bounding boxes and predictions allowed and *embedding size* is an hyperparameter from the Embedding layer which denotes the size of the embedding space. The importance of the embedding layer is that it transforms each of the YOLO predictions into a vector in an Euclidean space, understandable by the rest of the network.

Next, similarly as in the feature vectors module, the embedding output tensor is led to the series of layers Batch Normalization, ReLU activation, Dropout, Dense and Regularization, wrapped with Time Distributed layers. As said before, the number of units of the Dense layer is the same as in the features vectors Dense, so both output tensors are comparable in the posterior fusion layer.

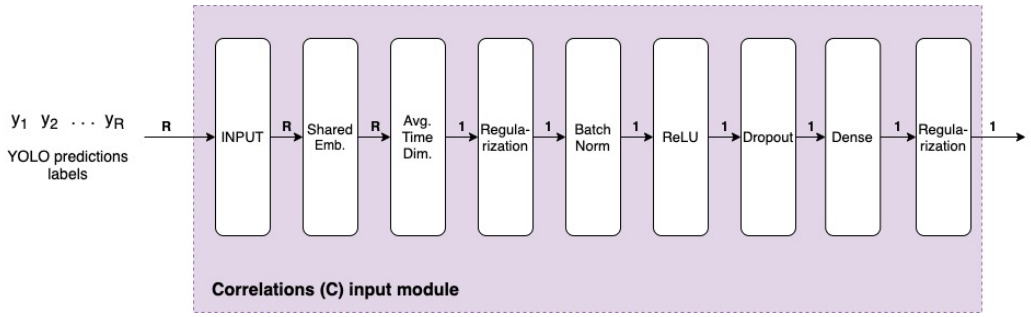


FIGURE 4.4: Correlations (C) input module structure.

We can see that the **correlations input module**, Fig. 4.4, is quite similar to the YOLO predictions one. Firstly, it is important to remark that the Embedding layer of this module (and the same applies to the menu input module) is the same layer being used in the YOLO predictions module. As the text vocabulary for the menu and YOLO predictions are equal, using the same Embedding layer forces the model to share the layer's weights across the three input modules and, so, the mapping from class labels to Euclidean vectors is the same independently of the input source.

In spite of coming from the same input source, the main difference between the YOLO predictions and correlations modules is how is treated the embedding output tensor. In the YOLO predictions module the time slice of the input is maintained while in the correlations module we apply an average through time, resulting in a single vector of length embedding size. This resulting vector is the average of the encoded class labels vectors, aiming to obtain a vector that could represent in a sense the correlations between the food classes predicted, having in mind that the embedding layer maps intrinsically related classes to close vectors in an Euclidean space.

In the **menu input module**, Fig. 4.5, we can see that it has the same structure as the correlations module. What differentiates them is the input. In this case, with the menu input we expect this module to produce a single vector average of the encoded menu's classes vectors and, thus, a vector representing the correlations between the daily menu's food classes. Note that the output has the same size as the correlations module output, so they are comparable in the next fusion layer.

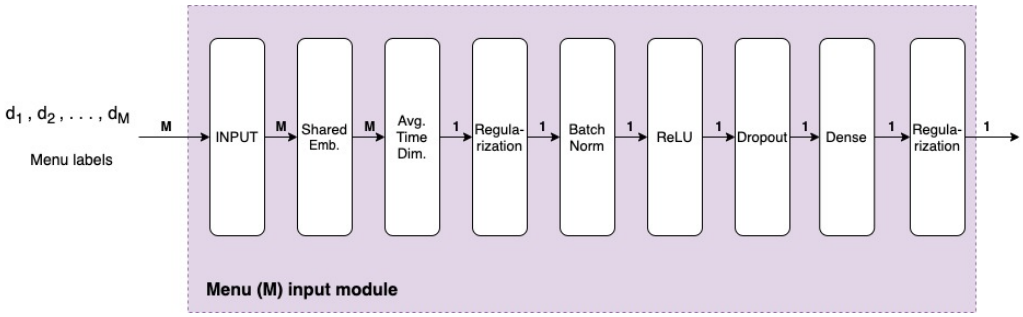


FIGURE 4.5: Menu (M) input module structure.

In Fig. 4.6 we can see that the output tensors from modules feature vectors (FV) and YOLO predictions (YP) on one side, correlations (C) and menu (M) on the other, are merged in pairs in an additive fusion layer. Remember that the output tensor from FV and YP maintain the time slice, and so the one-to-one relation, while C and M do not. Therefore, the output of the first fusion (FV + YP) has for each time slice t a vector that combines the features from FV[t] and YP[t] vectors. This output tensor is then led to a series of Batch Normalization, ReLU, Dropout and Dense layers, wrapped by Time Distributed layers. The number of units of the Dense layer is the same as the total number of classes and the activation is *softmax*, in order to obtain the first output of the model.

Inspired by *GoogLeNet* (Szegedy et al., 2014), also known as *Inception v1*, our model has, as mentioned before, two outputs: one at the end and the other in the middle of the network. It is a feature that worked well for the authors of GoogLeNet and we thought it suitable in our case, because inputs FV and YP are both together sufficient for making good predictions (in section 4.2 we will see quantitative proof of this statement). Then, this output can be combined with a farther output that includes the features extracted from C and M inputs.

Regarding correlations and menu, the output of the second fusion (C + M) is a single vector that combines the features from C and M outputs. The resulting vector is repeated *max frame* times and so converted to a tensor of dimensions $(batch\ size, max\ frame, num\ fusion\ features)$, same size as the output of fusion (FV + YP).

Both fusion outputs are finally combined with a multiplicative fusion layer and led to the second output of the model. Arrived in this point, for a single input image, the outputs of the model are

$$\text{Output 1: } (p_1^1,\ p_2^1,\ \ldots,\ p_R^1)$$

$$\text{Output 2: } (p_1^2,\ p_2^2,\ \ldots,\ p_R^2),$$

where the super index denotes the output, $R = max\ frame$ (maximum number of bounding boxes and predictions allowed for image), and

$$p_i^o = (x_1,\ x_2,\ \ldots,\ x_V),\ o \in \{1,2\},\ i \in [1,R],$$

is a vector containing the probability distribution for bounding box i and output o. V is the total number of classes of the vocabulary.

The merge of outputs 1 and 2 is computed as follows

$$\text{Merged output: } (p_1^m,\ p_2^m,\ ...,\ p_R^m),$$

where

$$p_i^m = (max\{p_i^1[1],\ p_i^2[1]\},\ max\{p_i^1[2],\ p_i^2[2]\},\ ...,\ max\{p_i^1[V],\ p_i^2[V]\}).$$

Then, the class label predictions are

$$(y_1^m,\ y_2^m,\ ...,\ y_R^m) = (argmax(p_1^m),\ argmax(p_2^m),\ ...,\ argmax(p_R^m))$$

where *argmax* is the function that returns the argument/index of the maximum value in the vector.

At this point we have prediction y_1^m for bounding box bb_1, y_2^m for bb_2 and etc. Now the model discards bounding boxes and predictions that are empty (in case there were less than *max frame* bounding boxes) or predicted as *background*. The remaining ones go through a last NMS algorithm with little more restrictive thresholds than the NMS in part 1, in order to definitively remove bounding boxes enveloping the same food instance in the original image.

The final output is, then, a list of bounding boxes $bb_1,\ bb_2,\ ...,\ bb_S$ with their corresponding predictions $y*_1,\ y*_2,\ ...,\ y*_S$, where S is the total number of them, different for each image.

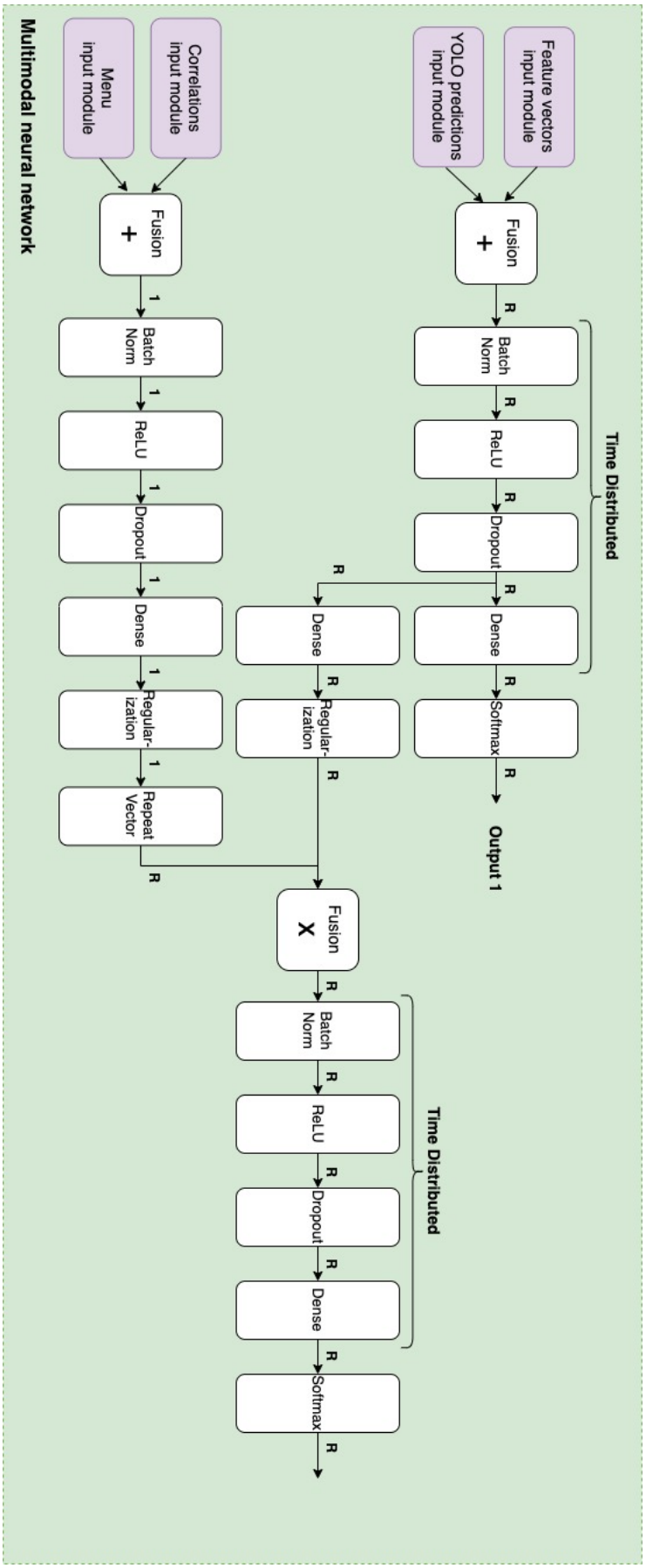


FIGURE 4.6: Fusions structure of the multimodal neural network.

4.2 Training our model

4.2.1 Dataset

The dataset used for training and testing our model was originally created by Bora, Bolaños, and Radeva, 2020, in their work and is property of *LogMeal*, a private API for Artificial Intelligence tasks applied to food. The dataset consists of a collection of food tray images taken on 95 different days at a restaurant in a real environment. Each image has its own annotations that are bounding boxes with class labels. Also, the dataset contains the menu for each of the days of the images.

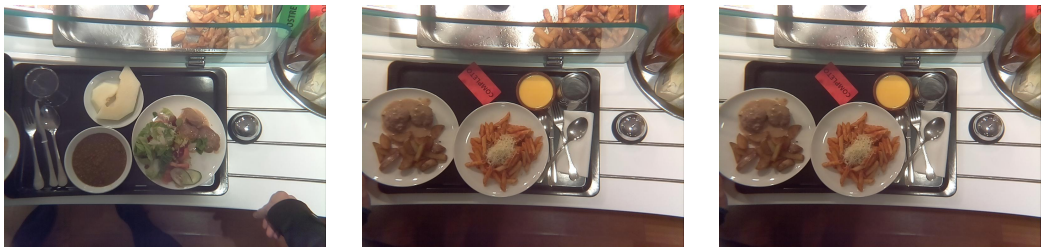


FIGURE 4.7: Example of images from the dataset.

First	paella, sopa de cocido
Second	salchichas al horno, croquetas caseras
Side Dish	ensalada, patatas al horno, patatas hervidas, patatas fritas, tomate, verduras con patatas, judias hervidas, judias blancas
Dessert	manzana, melon, flan, natillas, macedonia, platano, yogur, pastel, naranja, sandia, arroz con leche, uvas, kiwi
Drink	agua
Soft Drink	coca cola
Bread	pan

TABLE 4.1: Example of a daily menu

We have split the dataset in training and testing splits. The partition was done day-wise, the images of a whole day are in train or test, but not divided. Therefore, the training split contains 74 days that are a total of 11648 images that contain food of 95 different classes which are pieces of food, such as *bread* or *apple*, and dishes from the Spanish cuisine, such as *albóndigas en salsa* or *macarrones boloñesa*.

The test split is composed of 21 days that are a total of 4067 images of food from 56 classes (contained also in the training split). Although in the dataset there is a considerable high number of images, the also high number of different classes and the unfortunate class imbalance depicted in Fig. 4.8, it was not able to create a validation split, which would be the ideal for measuring performance during training with a separate collection of data. Therefore, we have used the test set for the validation and test phases.

4.2.2 Food candidates detection

The YOLO model we used was the same one trained by Bora, Bolaños, and Radeva, 2020, with the dataset they built in their work. As stated before, our

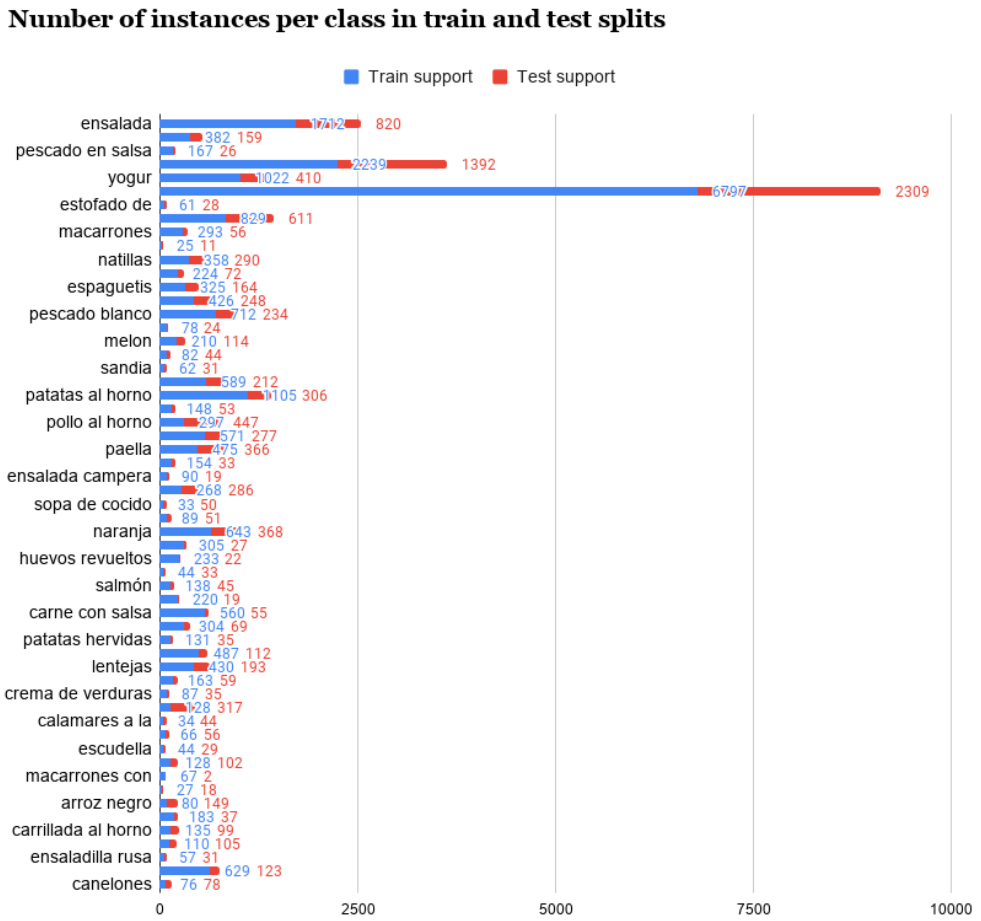


FIGURE 4.8: Number of instances per class in train and test splits.

work intents to be an alternative multimodal model to theirs solution, using the same base model for generating the candidates.

In this part, for extracting the candidates from YOLO we have used a threshold of 0.001, meaning that the returned bounding boxes from YOLO have a confidence of 0.001 or greater. Being 1.0 the maximum value of confidence, a threshold of 0.001 is quite low. It is true, but the purpose of our model is to not leave any food away, in spite of the amount of background gathered with this trade-off, since we expect our model to detect well background and discard it.

For the NMS section, we have used an *IOU* threshold of 0.6 for overlapped bounding boxes of the same class and an *IOU* threshold of 0.95 for overlapped bounding boxes of strictly different class. Meaning that we are more restrictive with the firsts and more permissive with the seconds.

These three threshold values were chosen after a grid search with the criterion of leaving few or none not detected food and having, in average, a workable number of candidate bounding boxes. After that, we decided the total number of bounding boxes allowed per image. We called it *max frame* (for the resemblance of frames in videos) and set it to 20. There were few images with more than 20

bounding boxes, but those discarded had a very low confidence and almost all were false positives bounding boxes of background or repeated bounding boxes, and so not of much interest.

After the NMS process, we did a little process of data preparation which basically consists of cropping the bounding boxes, save them in folders and write on files the paths to: the bounding boxes, the YOLO predictions, the menu and the ground truth derived. About the ground truth, in order to train the classification CNN and the Multimodal neural network, we need to create annotations for each of the bounding boxes based on the annotations of the original image. The way we did that was looping over the predictions sorted by YOLO confidence and compare each of those with the true annotations. In case the there's a match (overlap with more than 0.5 *IOU*), the ground truth for that predicted bounding box is the label of the original annotation. In the negative case, the ground truth is set to 0, the background class.

4.2.3 Feature vectors extraction

The CNN network used to extract the feature vectors was an *InceptionResnetV2*, initialized with ImageNet (Deng et al., 2009) weights, plus a final Fully Connected layer. First, the CNN was fine-tuned with the training bounding boxes generated from the previous step and validated with the bounding boxes from the test split. The fine-tuning process was done in 50 epochs with a batch size of 24, Adam optimizer, initial learning rate of 0.0001 and using step-based learning rate decay for the last epochs.

Taken the model's epoch with the highest accuracy (number of correct predictions over the total number of predictions), in order to generate the feature vectors we load the weights of the best epoch, drop the last Dense layer and make predictions with this model for all the bounding boxes images in the dataset, training and test. It results in vectors of size 1536 that contain intrinsic visual features of the image. Those vectors are saved in a file, and as well a file containing the paths to each of them is created.

The implementation, training and testing of this CNN, the variation for extracting the feature vectors and the Multimodal neural network (explained in the next subsection) was done using the library *Multimodal Keras Wrapper* version 2.1.3, which is a, quoting the creators: "Wrapper for Keras with support to easy multimodal data and models loading and handling." Moreover, the models were executed in a GPU using *Tensorflow* as backend, all isolated inside a *Docker Compose* image.

4.2.4 Multimodal neural network

For training the Multimodal neural network, we used an embedding size of 50 for the text modality inputs. There is no rule for the choice of the embedding size, but taking into account that it denotes the dimension of the output Euclidean space and that the number of classes of our dataset is almost 100, it seemed good to us reducing the dimension of the space to the half. We did some tests and there wasn't a big difference among embedding sizes close the 50.

About the number of features for the middle tensors in the network, after a few tests with different values, a number of 200 features worked the best. The same

applies to the structure and operations of the fusions. After trying fusions with multiple inputs combinations and the element-wise sum, product and concatenation operations, the configuration with the best results is the one chosen in the model and drawn in figures above. The concrete variations tested are explained in section 5.3 of Results chapter.

About the multi-output, we saw that the feature vectors and YOLO predictions inputs, apart from being related one-to-one, are the most relevant inputs as they come from two powerful networks (*YOLO* and *InceptionResnetV2*). A previous tested model with only a fusion of those two inputs gave pretty good results, so we thought it was interesting to preserve this output in the final model and make a combined prediction from two outputs.

The way of combining the predictions of the two outputs is the most straightforward way, giving equal importance to them. That is because we thought that although the second output has more information, in some cases it can be noisy and become in worse predictions.

Finally, the last NMS algorithm uses more restrictive thresholds than the previous NMS algorithm. The *IOU* threshold for overlapped bounding boxes with the same class is 0.4 and for bounding boxes of strictly different class is 0.9. Those hyperparameters were the pair of parameters that gave a higher metric as a result in a grid search with several values.

One last comment regarding the training of the Multimodal neural network is that during training we used Regularization, Batch Normalization and Dropout layers, and weight decay of 0.0001. These techniques were used only in the training process, not during prediction, and helped preventing the model to overfit. The Multimodal neural network was trained during 50 epochs with a batch size of 24, Adam optimizer, initial learning rate of 0.0001 and using a step-based learning rate decay.

4.3 Constructing the MFTR and YOLO models ensemble

Our second contribution in this work is the definition and construction of an ensemble of models Multimodal Food Tray Recognition (MFTR) and YOLO. The model structure is the same as MFTR, explained before. The ensemble is produced during prediction in the merge outputs step, which now interferes a third agent: the predictions from YOLO. Fig. 4.9 is a representation of the model.

For this ensemble, before the merging step, we need to have the classes labels and confidence scores that YOLO gives for each bounding box, the ones that in MFTR are discarded in the second NMS process of Food candidates detection part. Remember that in Food candidates detection part we apply NMS twice: first discarding overlapped bounding boxes with the same class and then discarding overlapped bounding boxes with strictly different classes. Thus, it is needed to save the bounding boxes with class labels and confidences that we get at the middle of the two NMS algorithm.

Then, in the Merge Outputs step of part 3, we read those previously saved YOLO predictions and adapt the YOLO data in order to match the outputs format and order of bounding boxes. After that, we have in hand the following elements:

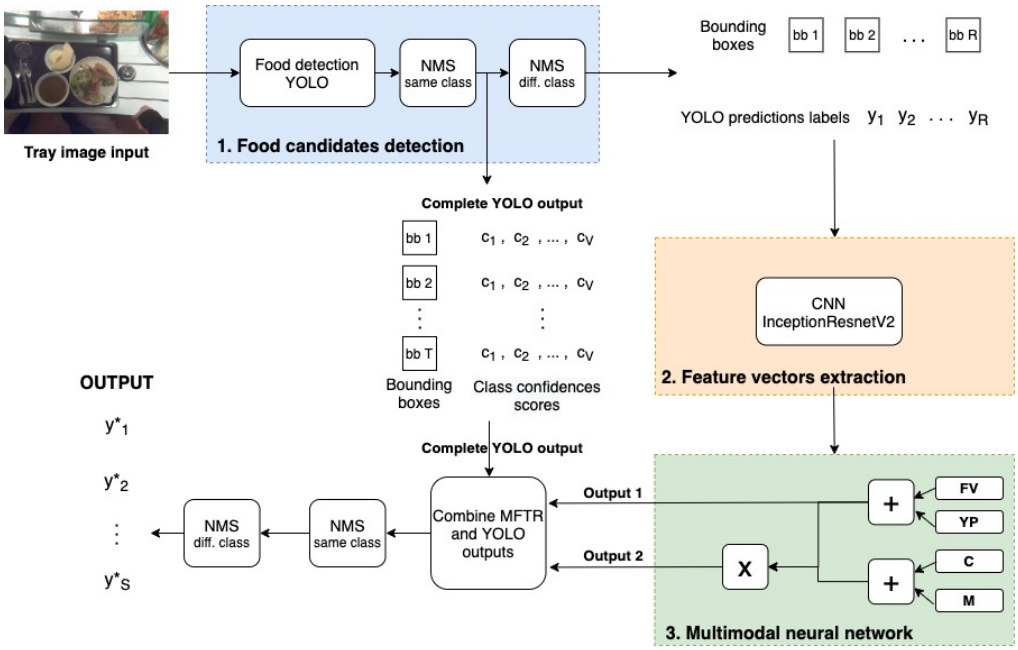


FIGURE 4.9: MFTR and YOLO models ensemble structure.

1. **Output 1** predicted labels and probabilities distributions:

$$(y_1^1, y_2^1, \dots, y_R^1) \text{ and } (p_1^1, p_2^1, \dots, p_R^1).$$

2. **Output 2** predicted labels and probabilities distributions:

$$(y_1^2, y_2^2, \dots, y_R^2) \text{ and } (p_1^2, p_2^2, \dots, p_R^2).$$

3. **YOLO** predicted labels and confidence scores:

$$(y_1^Y, y_2^Y, \dots, y_R^Y) \text{ and } (p_1^Y, p_2^Y, \dots, p_R^Y).$$

Remember that $y_i^o = argmax(p_i^o)$ where $o \in \{1, 2, Y\}$ denotes the number of output or YOLO for Y and $i \in [1, R]$ denotes the index of the bounding box.

Now, looping over i, i.e. over bounding boxes, we have two cases:

1. Case 1: At least 2 of the predictions y_i^1, y_i^2 and y_i^Y are equal.
   In that case, we return the prediction label of the majority and the highest probability value between the majority ones.

2. Case 2: All three prediction labels are different.
   We do the following:

   First, consider the vector of maximum probabilities between output 1 and 2 for prediction i

$$p_i^m = (max\{p_i^1[1], p_i^2[1]\}, max\{p_i^1[2], p_i^2[2]\}, \dots, max\{p_i^1[V], p_i^2[V]\}).$$

Name y_i^m the label of the maximum probability from the model:

$$y_i^m = argmax(p_i^m).$$

And name $prob_i^m$ the maximum probability value of the model:

$$prob_i^m = max(p_i^m).$$

Note that $prob_i^m$ is the probability of the model's prediction y_i^m.

Doing the same for YOLO predictions we have:

$$y_i^Y = argmax(p_i^Y) \text{ and } prob_i^Y = max(p_i^Y).$$

Then, we retrieve the probability that MFTR gives to the YOLO prediction. Say $j = y_i^Y$,

$$prob_i^{YM} = p_i^m[j].$$

Summarizing, we have:

(a) MFTR's best prediction y_i^m with probability $prob_i^m$.

(b) YOLO's best prediction y_i^Y with confidence score $prob_i^Y$.

(c) MFTR's probability of YOLO's best prediction: $prob_i^{YM}$.

Now, if

$$prob_i^m < prob_i^Y + prob_i^{YM} \tag{4.1}$$

we return the YOLO's best prediction y_i^Y with probability maximum between $prob_i^Y$ and $prob_i^{YM}$. Otherwise, we return the model's best prediction y_i^m with probability $prob_i^m$.

The intuition behind case 2, is that if the YOLO prediction is strong enough, not only having a high confidence but also requiring to be sufficiently present in the model's probabilities, then we should rely on this prediction instead of the model's prediction.

The *strength* of the YOLO prediction measured as said before is compared with the model's best probability. Despite the fact that the right term can be greater than 1, and, so, not being a fair comparison of probabilities, we found plausible to use it. Because we do not expect it to be a comparison of probabilities, but instead a certain comparison of confidence in the predictions: *"Is the YOLO prediction strong enough in two models compared to the probability of the MFTR's best prediction?"*.

Chapter 5

Results

In this chapter we are going to present some of the results obtained from our models. First, in section 5.1 we are going to define the metrics used to evaluate numerically our models. In section 5.2 we will introduce the Baseline model we used to compare our models with. In section 5.3 we will explain different model variations considered during the development process. Sections 5.4 and 5.5 contain, respectively, quantitatively and qualitative results. Finally, in section 5.6 we are going to discuss the limitations of our models presented in the previous sections.

5.1 Metrics

Before going deep into the results, we are going to define the metrics used to measure the performance of models in our context. Those metrics are basically *Average F1-score* and *Weighted Average F1-score*. In order to define these two we need to previously define other terms and metrics that lead to the mentioned metrics.

- **True positive**: We say that a prediction is a true positive (TP) if the predicted value is equal to the ground truth.
- **True positive**: We say that an element is a true negative (TN) if the model did not make a prediction for this element nor was it a ground truth.
- **False positive**: We say that a prediction is a false positive (FP) if the model has made a prediction of an element that doesn't appear in the ground truth annotations or has a different value (in multi-class classification problems).
- **False negative**: Elements annotated in the ground truth that not predicted or predicted wrong by the model are called false negatives (FN).
- **Accuracy**: Accuracy is the rate of number of correct predictions over the total number of predictions. Equivalently,

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Precision**: Precision is the number of correct predictions labeled as the positive class divided by the total number of predictions labeled as the positive class.

$$Precision = \frac{TP}{TP + FP}$$

- **Recall**: Recall is the number of correct predictions labeled as the positive class divided by the actual number of elements that belong to the positive class.

$$Recall = \frac{TP}{TP + FN}$$

- **F1-score**: The F1-score is a metric that combines the precision and recall metrics by an harmonic mean.

$$F1\text{-}score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

Particularly inn our context problem, multi-class classification, we are interested in computing the metrics class-wise, obtaining Precision and Recall metrics for each of the classes. With them calculated, we have considered:

- **Average Precision (Recall)**: which is computed as the mean of all classes Precision (Recall) values.

$$Average\ Precision = \frac{\sum_{i=1}^{N} P_i}{N}$$

$$Average\ Recall = \frac{\sum_{i=1}^{N} R_i}{N}$$

  where N is the total number of classes tested, P_i is the Precision of class $i \in \{1, \ldots, N\}$ and R_i is the Recall of class i.

- **Support**: The support is the number of elements for each class that are annotated in the ground truth. S_i is the support of class i.

- **Weighted Average Precision (Recall)**: It is the weighted mean of all classes Precision (Recall) values. Weights are the support for each class.

$$Weighted\ Average\ Precision = \frac{\sum_{i=1}^{N} S_i \cdot P_i}{\sum_{i=1}^{N} S_i}$$

$$Weighted\ Average\ Recall = \frac{\sum_{i=1}^{N} S_i \cdot R_i}{\sum_{i=1}^{N} S_i}$$

- **Average F1-score**: It is the F1-score of Average Precision and Average Recall.

- **Weighted Average F1-score**: It is the F1-score of Weighted Average Precision and Weighted Average Recall.

In the following section we will see that the support among classes is uneven, making it clear the presence of class imbalanced. Due to that problem, the *Weighted Average F1-score* metrics, which accounts for class imbalance, is considered for us a more suitable metric to measure and more fair in order to compare between models. In the next comparisons we have calculated both metrics, but we have given more relevance to *Weighted Average F1-score*.

5.2 Baseline

As explained in section 4.1, our model is based on the predictions that YOLO does of our images. If we consider the whole part 1 of our model MFTR, which consists of the YOLO and NMS filtering algorithms, it is able of making very good predictions (remember that YOLO is a state of the art object detection network). Thus, we could say that YOLO + NMS filtering or MFTR part 1 is our baseline model and the one which we are going to compare with.

In section 4.2.2 we also explained that we have used very relaxed threshold values in order to discard the less relevant data, despite having to deal with more amount of background data. On the other hand, we have searched for the best configuration of parameters for this model and computed a measure of the performance with those parameters.

Therefore, using the testing data split, we did grid search of parameters:

- Minimum YOLO confidence, which we named simply *thresh*. Possible values: 0.01, 0.1, 0.2, 0.3, 0.4, 0.5
- *IOU* threshold for overlapped bounding boxes of the same class, which we named *NMS*. Possible values: 0.2, 0.3, 0.4, 0.5, 0.6
- *IOU* threshold for overlapped bounding boxes of strictly different classes, which we named *CNMS*. Possible values: 0.8, 0.85, 0.9, 0.95, 0.99

After the grid search, the best configuration of parameters was $thresh = 0.2$, $NMS = 0.4$ and $CNMS = 0.95$, which gave an *Average F1-score* $= 0.7587$ and a *Weighted Average F1-score* $= 0.8114$. Those two are our baseline metrics and those we will compare our model with. From now on, we will refer to YOLO plus NMS filtering algorithms with paramaters $thresh = 0.2$, $NMS = 0.4$ and $CNMS = 0.95$ as **Baseline model**, because we consider it the baseline model we compare with. However, remember that for training our model, MFTR, we used lower parameters for generating the food candidates (section 4.2).

In figure 5.1 can be found the food classes that appear in the test split together with their support and F1-score. First, note that there's a certain class imbalance: on one hand, there are classes like *pan* or *patatas fritas* with more than a thousand instances, and on the other hand there are classes such as *macarrones con queso* or *agua* with less than twenty instances in the test set. A comparison of number of food instances in train and test splits can be found in Fig. 4.8. The class imbalance is a consequence of the nature of the dataset: extracted from real self-service restaurant.

Secondly, the baseline model performs quite well in the majority of the classes, we can see it in the table but also with the *Average F1-score* of 0.7587. It is not rare since the baseline model chosen not only is a state of the art model for object detection, but also because we consider it with a post-filtering process. However, note that there are certain classes with poor performance. For example *salmón*, *patatas hervidas* and *kiwi* are three classes with a F1-score of 0.

Regarding this classes with a F1-score of 0, note that the three have a very low support (less than 50 instances) and, thus, have little affect to the *Weighted Average F1-score*, the metric we think is more relevant because it accounts for class imbalance.

Food class	Support	F1-score	Food class	Support	F1-score
ensalada	820	0.89	platano	51	0.71
flan	159	0.76	naranja	368	0.89
pescado en salsa	26	0.71	lomo a la brasa	27	0.82
patatas fritas	1392	0.86	huevos revueltos	22	0.70
yogur	410	0.80	ensalada de quinoa	33	0.74
pan	2309	0.72	salmón	45	0.00
estofado de legumbre	28	0.40	fideos a la cazuela	19	0.66
salchichas al horno	611	0.81	carne con salsa	55	0.84
macarrones boloñesa	56	0.97	pastel	69	0.79
agua	11	0.71	patatas hervidas	35	0.00
natillas	290	0.77	albóndigas en salsa	112	0.96
manzana	72	0.78	lentejas	193	0.97
espaguetis	164	0.92	pechuga de pollo a la plancha	59	0.72
librito de york y queso	248	0.90	crema de verduras	35	0.64
pescado blanco	234	0.77	hamburguesa	317	0.74
coliflor y patata al vapor	24	0.86	calamares a la romana	44	0.66
melon	114	0.83	ensalada con queso	56	0.85
coca cola	44	0.72	escudella	29	0.89
sandia	31	0.59	empedrado	102	0.72
arroz con leche	212	0.81	macarrones con queso	2	1.00
patatas al horno	306	0.81	kiwi	18	0.00
macedonia	53	0.68	arroz negro	149	0.90
pollo al horno	447	0.92	guisantes con patatas	37	0.89
verduras con patatas	277	0.93	carrillada al horno	99	0.89
paella	366	0.96	empanadillas	105	0.56
tomate	33	0.62	ensaladilla rusa	31	0.78
ensalada campera	19	0.91	fideuá	123	0.94
croquetas caseras	286	0.88	canelones	78	0.93
sopa de cocido	50	0.81			

FIGURE 5.1: Support and F1-score per class of baseline model.

5.3 Model variations tested

Regarding the part 3 of our model, before choosing the final structure of input fusions, we implemented and tested different variations of the final MFTR model. Following the same notation used previously, we are going to use these abbreviations: FV for the Feature vectors input module, YP for the YOLO predictions input module, C for the correlations input module and M for the menu input module. About the fusion operations, we will use + for the additive type fusion, $\cdot$ for the multiplicative type fusion and concat for the concatenate type fusion. With this notation, the part 3 of our MFTR model can be expressed as $(FV + YP) \cdot (C + M)$.

The variations implemented with single output are:

- FV model
- $FV + C$ model
- $FV + YP$ model
- $FV + M$ model
- $FV + YP + C$ model
- $FV + C + M$ model
- $FV + YP + C + M$ model

The variations implemented with two outputs, the first output always after the first fusion and the second output at the final fusion, are:

- $(FV + YP + C) + M$ model
- $(FV + YP) + (C + M)$ model
- $(FV + YP) + (C \cdot M)$ model
- $(FV + YP)$ *concat* $(C + M)$ model
- $(FV + YP) \cdot (C \cdot M)$ model
- $(FV + YP + C) + (C + M)$ model
- $(FV + YP + C) + (C \cdot M)$ model
- $(FV + YP + C) \cdot (C \cdot M)$ model

Regarding the models ensemble, we would like to add that we have tested different comparison statements such as, following the same notation explained in section 4.3:

- Fixed thresholds: $prob_i^m < threshold$.
- Comparison between MFTR's probability and YOLO's confidence score: $prob_i^m < prob_i^Y$.
- Maximum of YOLO's confidence score and MFTR's probability of YOLO's best prediction: $prob_i^m < max\{prob_i^Y,\ prob_i^{YM}\}$.

After testing these and other statements, the one with greater results was the statement 4.1 of the ensemble model.

5.4 Quantitative results

The first comparison of results is among the model variations. For each of the variation we have trained the model during 50 epochs and evaluated the model with the test split 5 times. The metrics computed correspond to the *Average F1-score* and the *Weighted Average F1-score*. After the 5 executions, we calculated the mean and median for each of the metrics mentioned.

We will consider the **median**, and specially the median of the Weighted Average F1-score, as the most representative metric since, unlike the mean, the median corresponds to the measure of performance of an existing execution. And so, we will consider the mean as an extra informative metric.

In table 5.1 we can see the evaluation results for the single output variations of the model. Starting with the FV model, a model with only the feature vectors as input, it achieves a Weighted Average F1-score of 0.8445, surpassing its baseline equivalent metric. Note that regarding the double input models, the secondary input which contributes more is, as expected, the YOLO predictions input making the model surpass the 0.85 threshold.

On the other hand the correlations and menu inputs alone do not provide significant information, specially the menu, being $FV + M$ the model variation with the worst results. Note, that $FV + YP$ variation is even better than the three or four-input variations. Our deduction is that, due to the one-to-one relation

	FV model		FV + C model		FV + YP model		FV + M model		FV + YP + C model		FV + C + M model		FV + YP + C + M model	
	AVG F1	**W-AVG F1**	**AVG F1**	**W-AVG F1**	**AVG F1**	**W-AVG F1**	**AVG F1**	**W-AVG F1**	**AVG F1**	**W-AVG F1**	**AVG F1**	**W-AVG F1**	**AVG F1**	**W-AVG F1**
Mean	**0.7236**	**0.8443**	**0.7231**	**0.8445**	**0.7396**	**0.8525**	**0.6958**	**0.8243**	**0.7380**	**0.8510**	**0.7146**	**0.8344**	**0.7259**	**0.8405**
Exec. 1	0.7269	**0.8445**	0.7279	0.8460	0.7415	0.8531	0.6941	**0.8236**	0.7371	**0.8509**	0.7044	0.8286	0.7257	**0.8432**
Exec. 2	**0.7232**	0.8428	**0.7226**	**0.8449**	0.7354	0.8520	0.6835	0.8190	0.7360	0.8510	0.7207	0.8392	0.7338	0.8465
Exec. 3	0.7230	0.8451	0.7226	0.8453	**0.7401**	0.8516	**0.6946**	0.8260	0.7409	0.8522	**0.7165**	0.8370	0.7141	0.8267
Exec. 4	0.7220	0.8429	0.7190	0.8418	0.7431	0.8530	0.6993	0.8209	0.7382	0.8505	0.7179	**0.8348**	0.7291	0.8428
Exec. 5	0.7232	0.8462	0.7233	0.8446	0.7379	**0.8528**	0.7074	0.8318	**0.7375**	0.8503	0.7133	0.8324	**0.7269**	0.8434

TABLE 5.1: Table of results for single output model variations. Median of metrics are marked in bold.

between the future vectors and YOLO predictions, the network is able to learn stronger correlations than the models that doesn't have the pair FV + YP. Regarding $FV + YP + C$ and $FV + YP + C + M$ variations, we think that the correlations and menu inputs may be producing some kind of noise that doesn't let the network to learn better, nor equally, than $FV + YP$.

After noticing these behaviours of the variations, and particularly the good results of $FV + YP$, we thought it was interesting to have a middle-output from the $FV + YP$ fusion in next model variations. These results also gave us a first intuition that probably the best structure would be with a $FV + YP$ fusion on one hand and correlations and menu fusioned together on the other hand. Despite having this intuition, we decided to also test other fusion structures for the multi-output variations.

In table 5.2 we can see the evaluation results for the multi output variations of the model. Remember that the first output is after the first fusion expressed with parenthesis. The first thing to notice from the three figures is that all of the multi output models have better results than the single output models. Consequently we could say that the double output structure and the output merging step is working well.

Looking to the median weighted average F1-score of these models, we can see that in general the structure of modules FV and YP fusioned together on one side and modules C and M fusioned together on the other side, independently of the fusion operations, perform better than other fusion structures. Between them, the top two variations are $(FV + YP) + (C + M)$ and $(FV + YP) \cdot (C + M)$, MFTR in the table, which are considerably better than the others, but with slight differences in results between the two. However, MFTR has better results in all four metrics considered: mean and median of average F1-score and weighted average F1-score.

Moreover, notice that MFTR, $(FV + YP) \cdot (C + M)$ variation, has the best results in the four metrics among all the variations. That is the reason why this is the structure of part 3 of our model MFTR.

Now, with the best variation taken, we have evaluated the MFTR and YOLO models ensemble explained in detail in section 4.3. The evaluation process done for this model is the same done to the variations. The results are presented in table 5.3.

Table 5.3 is the important comparison. First, taking the median metrics, note that MFTR model is able to surpass both of the baseline metrics. Weighted average F1-score is surpassed by more than 0.05 units, which is a great difference in deep learning context, but average F1-score is slightly surpassed.

Regarding the MFTR+YOLO ensemble, note that, respect the baseline metrics, both median metrics of the ensemble are significantly better: average F1-score is more than 0.03 units greater and weighted average F1-score is more than 0.065 units greater. Comparing with MFTR, the difference between weighted average F1-scores is of more than 0.01 and between average F1-scores is of more than 0.035, both also relevant differences.

A table of per class F1-scores for baseline YOLO model, MFTR model and MFTR + YOLO models ensemble is presented in table 5.4. In that table, the best F1-score per class is marked in bold and the second best is marked in italics. Note: the

	(FV + YP + C) + M		(FV + YP) + (C + M)		(FV + YP) + (C · M)		(FV + YP) concat (C + M)		MFTR		(FV + YP) · (C · M)		(FV + YP + C) + (C + M)		(FV + YP + C) + (C · M)		(FV + YP + C) · (C · M)	
	AVG F1	**W-AVG F1**	**AVG F1**	**W-AVG F1**	**AVG F1**	**W-AVG F1**	**AVG F1**	**W-AVG F1**	**AVG F1**	**W-AVG F1**	**AVG F1**	**W-AVG F1**	**AVG F1**	**W-AVG F1**	**AVG F1**	**W-AVG F1**	**AVG F1**	**W-AVG F1**
Mean	**0.7571**	**0.8589**	**0.7596**	**0.8604**	**0.7536**	**0.8592**	**0.7594**	**0.8598**	**0.7613**	**0.8624**	**0.7569**	**0.8602**	**0.7565**	**0.8590**	**0.7568**	**0.8602**	**0.7521**	**0.8596**
Exec. 1	0.7612	0.8608	0.7563	0.8581	0.7504	0.8570	0.7638	0.8615	0.7621	0.8630	0.7614	0.8619	0.7548	0.8590	0.7660	0.8619	0.7505	0.8579
Exec. 2	0.7542	0.8573	0.7582	0.8620	**0.7555**	0.8614	0.7570	0.8595	0.7580	0.8617	**0.7586**	**0.8603**	0.7574	0.8575	0.7539	0.8584	**0.7509**	0.8583
Exec. 3	**0.7543**	**0.8595**	**0.7633**	**0.8617**	0.7572	0.8607	0.7524	0.8573	0.7679	0.8641	0.7550	0.8581	0.7530	**0.8589**	0.7589	0.8611	0.7540	0.8617
Exec. 4	0.7633	0.8596	0.7661	0.8638	0.7568	**0.8591**	0.7625	0.8608	**0.7610**	**0.8620**	0.7606	0.8609	**0.7558**	0.8593	0.7483	**0.8598**	0.7458	**0.8591**
Exec. 5	0.7523	0.8574	0.7543	0.8564	0.7482	0.8576	**0.7616**	**0.8601**	0.7575	0.8610	0.7491	0.8600	0.7615	0.8604	**0.7568**	0.8597	0.7596	0.8613

TABLE 5.2: Table of results for multi-output model variations. Median of metrics are marked in bold.

Baseline model			MFTR model		MFTR + YOLO ensemble	
AVG F1	**W-AVG F1**		**AVG F1**	**W-AVG F1**	**AVG F1**	**W-AVG F1**
0.7587	**0.8114**	**Mean**	**0.7613**	**0.8624**	**0.7970**	**0.8770**
		Exec. 1	0.7621	0.8630	0.7934	0.8750
		Exec. 2	0.7580	0.8617	0.7924	0.8767
		Exec. 3	0.7679	0.8641	0.8043	0.8777
		Exec. 4	**0.7610**	**0.8620**	0.7986	0.8780
		Exec. 5	0.7575	0.8610	**0.7962**	**0.8774**

TABLE 5.3: Table of evaluation results comparison between the baseline YOLO model, the MFTR model and the MFTR + YOLO ensemble. The median of the average and weighted average F1-scores are marked in bold.

F1-scores of the table are those corresponding to the execution of the model with the weighted average F1-score equal to the median of the executions.

The first analysis we can make of table 5.4 is that our model, MFTR, performs better than the baseline model in 34 classes out of 56, which is more than 60% of the classes.

Secondly, the MFTR+YOLO ensemble model is better than the baseline model in 38 classes (67.86%) and also is better than both, the baseline model and MFTR, in 30 classes (53.57%).

Thirdly, the baseline model is the best model of the three in 18 classes (32.14%). This means that, although the MFTR model and the MFTR+YOLO ensemble model exceed the baseline model in average F1-score and weighted average F1-score, in classes terms there is yet an almost one third of possible improvement.

Finally, we would like to remark some of the metrics values per classes in different senses.

- *agua*, *lomo a la brasa* and *sopa de cocido* are examples of classes with high F1-scores in the baseline model but very low scores in the other models.
- *salmón* and *patatas hervidas* are examples of classes that have very low F1-scores in the three models. None of our both models was able to improve the F1-score of 0 of the baseline model.
- *kiwi*, *empanadillas* and *calamares a la romana* are three examples of classes in which our both models we able to improve significantly the F1-score of the baseline model. In particular, the class *kiwi* improved from 0 to 0.560 and 0.480.

	Baseline model	**MFTR model**	**MFTR+YOLO ensemble**
	F1-score per class	**F1-score per class**	**F1-score per class**
ensalada	0.889	*0.900*	**0.917**
flan	0.757	**0.826**	*0.806*
pescado en salsa	**0.708**	0.607	*0.630*
patatas fritas	0.861	*0.913*	**0.923**
yogur	0.798	**0.868**	*0.866*
pan	0.717	**0.887**	*0.882*
estofado de legumbre	0.400	*0.514*	**0.594**
salchichas al horno	0.810	*0.914*	**0.925**
macarrones boloñesa	0.973	**0.982**	*0.974*
agua	**0.706**	0.000	*0.167*
natillas	0.773	**0.805**	**0.808**
manzana	0.784	**0.817**	**0.817**
espaguetis	*0.919*	0.878	**0.924**
librito de york y queso	0.898	*0.910*	**0.924**
pescado blanco	**0.768**	0.753	*0.765*
coliflor y patata al vapor	0.857	*0.947*	**0.982**
melon	0.828	*0.963*	**0.968**
coca cola	0.716	*0.857*	**0.911**
sandia	0.585	*0.885*	**0.909**
arroz con leche	**0.812**	0.800	*0.801*
patatas al horno	**0.807**	*0.787*	0.781
macedonia	0.683	**0.868**	*0.857*
pollo al horno	0.920	*0.939*	**0.960**
verduras con patatas	*0.929*	0.898	**0.952**
paella	0.957	*0.962*	**0.979**
tomate	0.615	*0.783*	**0.800**
ensalada campera	*0.914*	0.909	**0.941**
croquetas caseras	0.877	*0.912*	**0.928**
sopa de cocido	**0.809**	0.387	*0.485*
platano	0.708	**0.914**	**0.914**
naranja	0.888	*0.955*	**0.964**
lomo a la brasa	**0.818**	0.323	*0.397*
huevos revueltos	0.700	*0.789*	**0.850**
ensalada de quinoa	*0.741*	*0.741*	**0.847**
salmón	0.000	**0.178**	*0.136*
fideos a la cazuela	**0.655**	0.508	*0.610*
carne con salsa	**0.839**	0.760	*0.779*
pastel	0.790	*0.885*	**0.917**
patatas hervidas	0.000	*0.054*	**0.056**
albóndigas en salsa	**0.959**	0.820	*0.904*
lentejas	**0.968**	*0.962*	0.957
pechuga de pollo a la plancha	**0.723**	0.533	*0.587*
crema de verduras	0.641	*0.691*	**0.700**
hamburguesa	**0.737**	0.638	*0.729*
calamares a la romana	0.655	*0.949*	**0.961**
ensalada con queso	**0.845**	0.625	*0.730*
escudella	**0.893**	0.489	*0.815*
empedrado	*0.722*	0.717	**0.733**
macarrones con queso	**1.000**	**1.000**	0.800
kiwi	0.000	**0.560**	*0.480*
arroz negro	0.902	*0.938*	**0.941**
guisantes con patatas	*0.889*	0.848	**0.903**
carrillada al horno	0.888	**0.912**	*0.902*
empanadillas	0.562	**0.896**	**0.896**
ensaladilla rusa	**0.784**	0.488	*0.545*
fideuá	**0.939**	0.923	*0.935*
canelones	**0.928**	0.806	*0.889*

TABLE 5.4: Table of classes F1-score for each model.

5.5 Qualitative results

Once seen the qualitative results, in this section are presented several predictions that are result of our models. In Fig. 5.2 we can see three images that are examples of good detection and recognition of food done by our models, MFTR and MFTR+YOLO ensemble.

There are some cases that the baseline model predicts as food an unusual object of the background, such as a purse or a mobile phone. In order to prevent this, as explained in Chapter 4, our model was trained with background and as a result both of our models are often able to discard these candidates. Fig. 5.3 is an example.

In other cases the baseline model misses predictions due to its restrictive parameters and its bad performance in some classes, as seen in the previous section. Our model, which gets the candidates predictions from a baseline model with relaxed parameters, is able to detect food candidates that are missed by the baseline model we compare with. In Fig. 5.4 we present an example of this case. Baseline model misses *arroz con leche* and *empanadillas*, but MFTR and ensemble models detect correctly them.

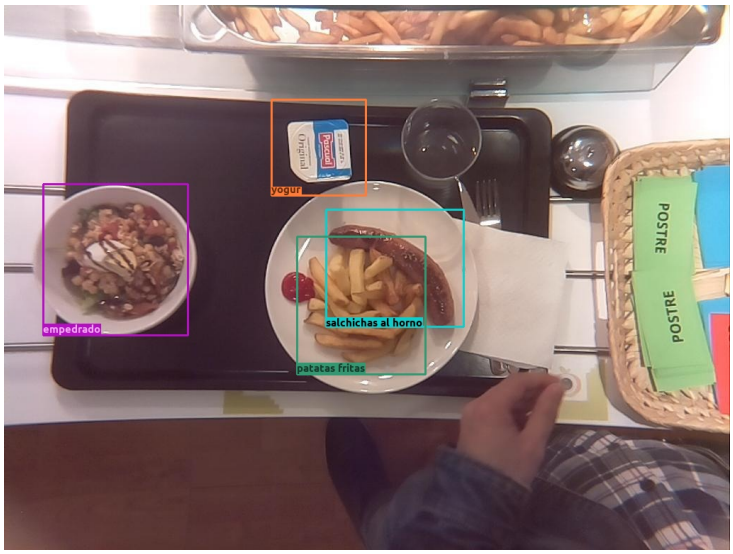
Although our model has a menu input, it has learned not to depend exclusively on it. In Fig. 5.5 we can see that all three models detect *librito de york y queso*, a food dish that in that day it wasn't in the menu of the restaurant and, even though, it was correctly predicted.

In some cases, due to the differences in performance of our models among classes, the baseline model recognises correctly a bounding box and MFTR recognises it as a wrong class, or the other way around. Fig 5.6 shows an example of the effectiveness of the ensemble model regarding these cases, which based on the decision process explained in 4.3 is able to change the wrong prediction of MFTR to the correct one made by the baseline model.

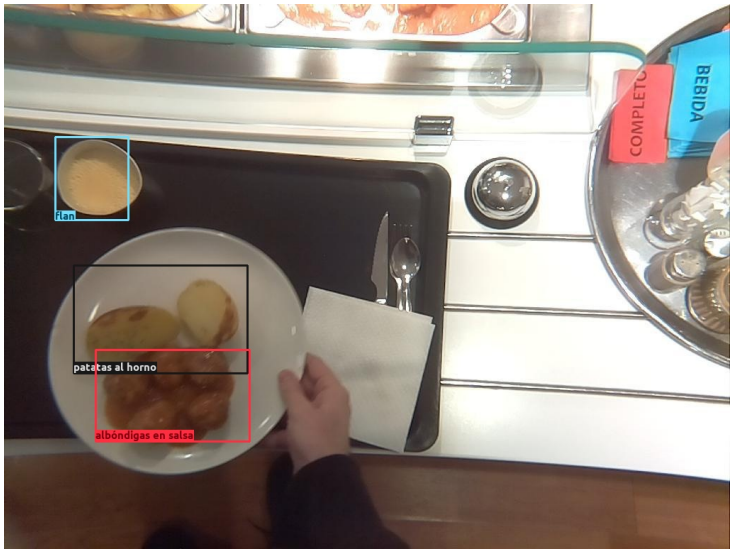
On the other hand, the quantitative results show that the results of our models are not perfect. We have also seen that there are some classes in which the baseline performs better than our proposed models. We have seen, in particular, that *sopa de cocido* is an example of these classes. Fig 5.7 is a visual example of that. The baseline models predicts it but our models do not. Most likely, the bounding box had been predicted as background and, so, discarded. In the previous Fig. 4.8 we can see that there are few instances of this class in the dataset, what makes it difficult for the model to generalize well.

Similarly, in Fig. 5.8 we can see that a background object, a pack of tissues, is predicted as *yogur* in MFTR and ensemble models. We deduce that in this case, the fact of having low parameters did create a bounding box of it as a candidate. However, compared to mobile phones and wallets that are common non-food objects, a pack of tissues is more rare and possibly the model during training did see none or few examples of it, making it difficult to learn to associate it as background.

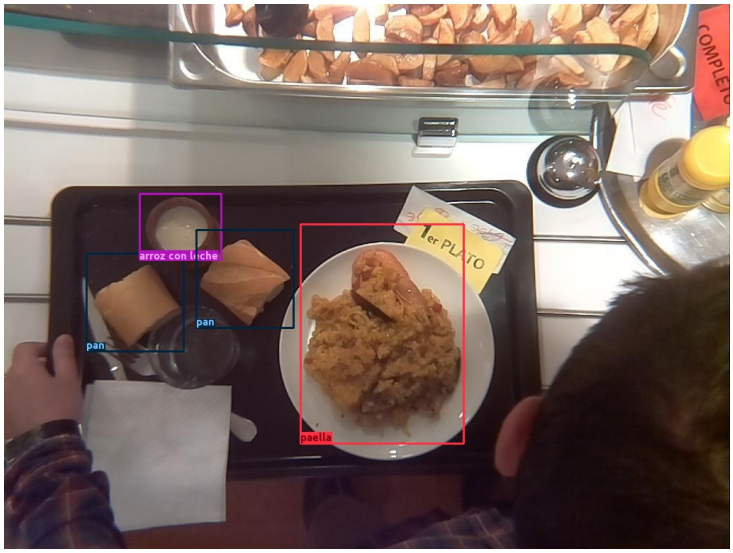
Finally, more visual results can be found in the Appendix A.



(A) MFTR and MFTR+YOLO ensemble

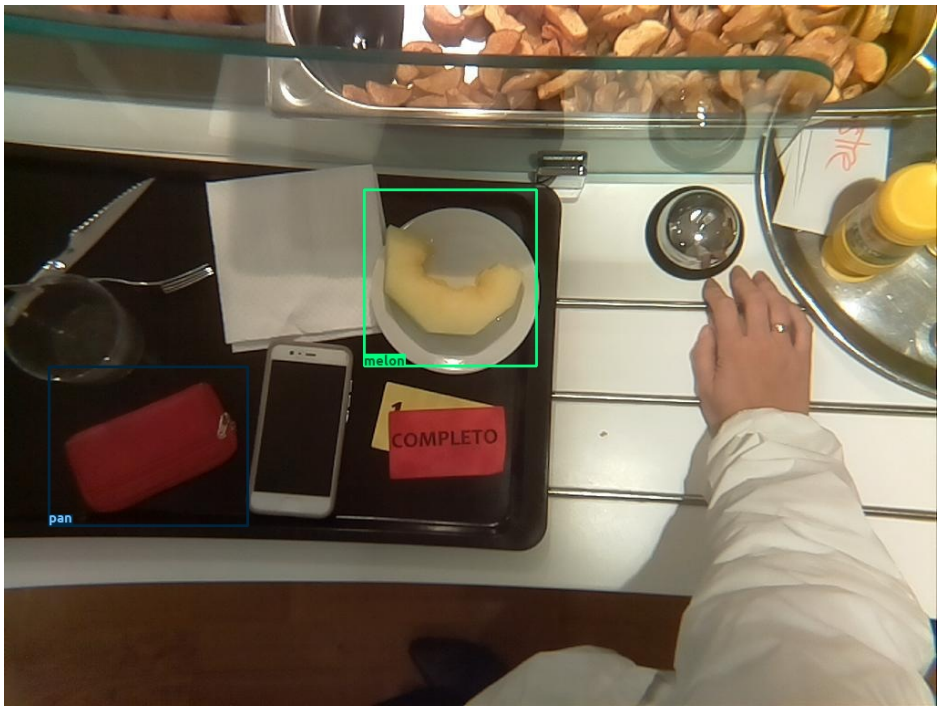


(B) MFTR and MFTR+YOLO ensemble

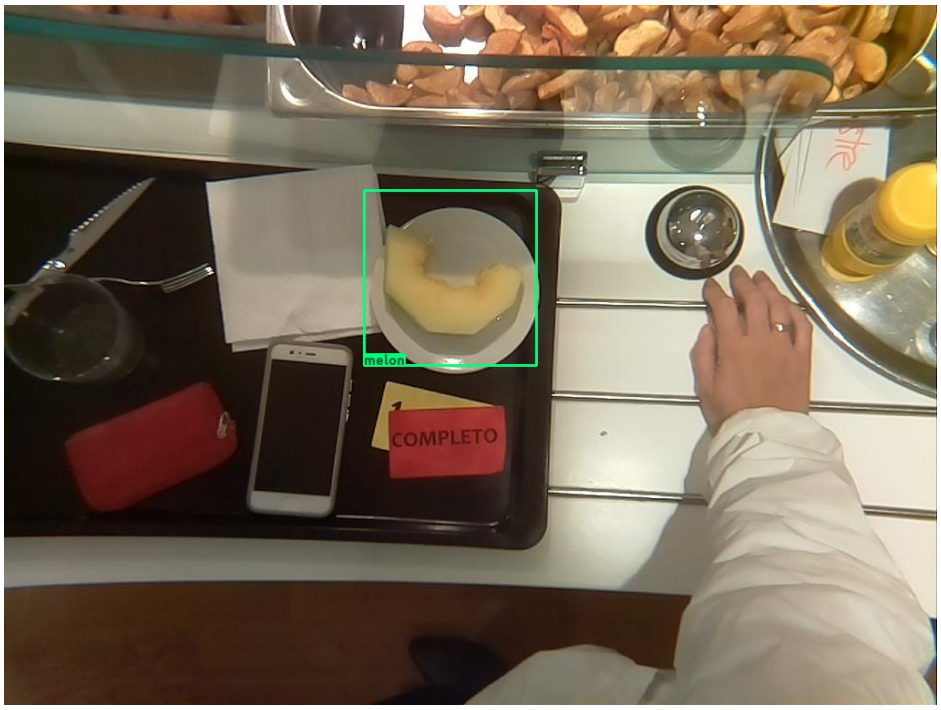


(C) MFTR and MFTR+YOLO ensemble

FIGURE 5.2: Three examples of correct predictions results.



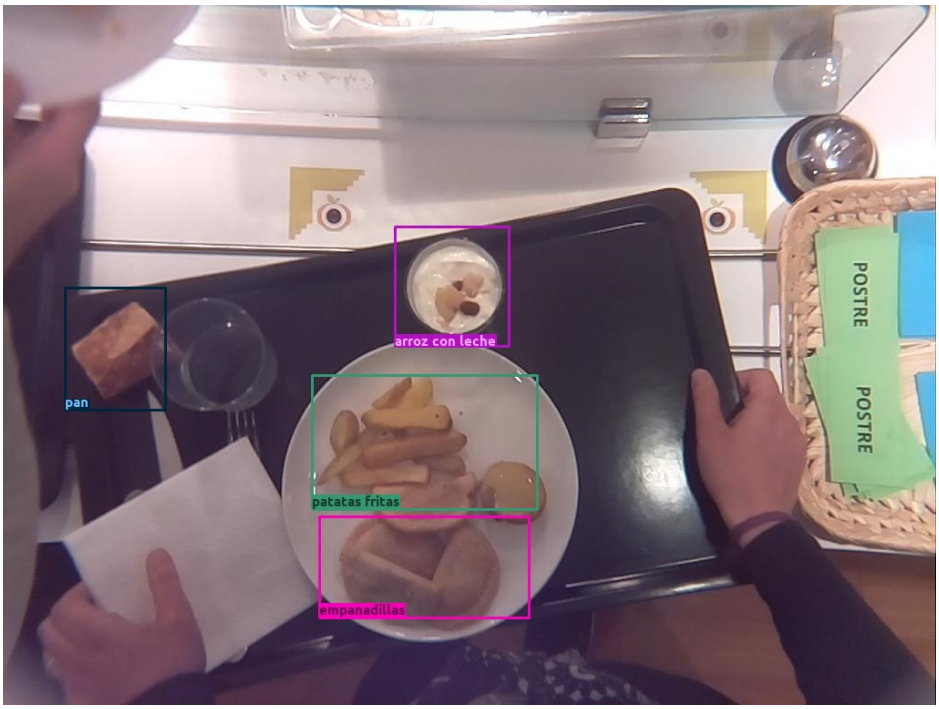
(A) Baseline model



(B) MFTR model and MFTR+YOLO ensemble

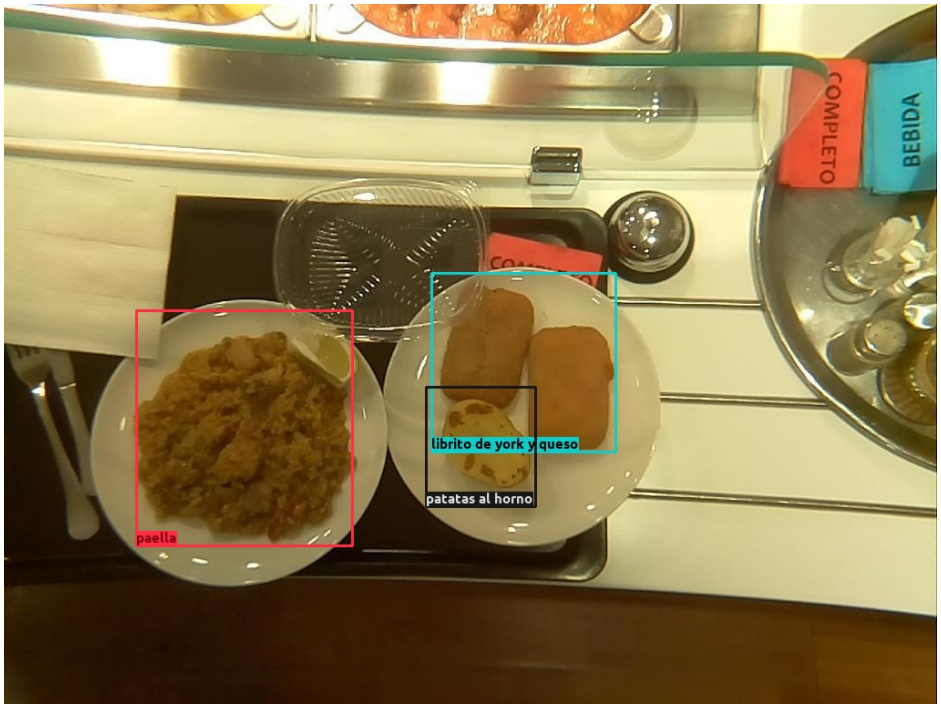
FIGURE 5.3: Example of background correction.

(A) Baseline model



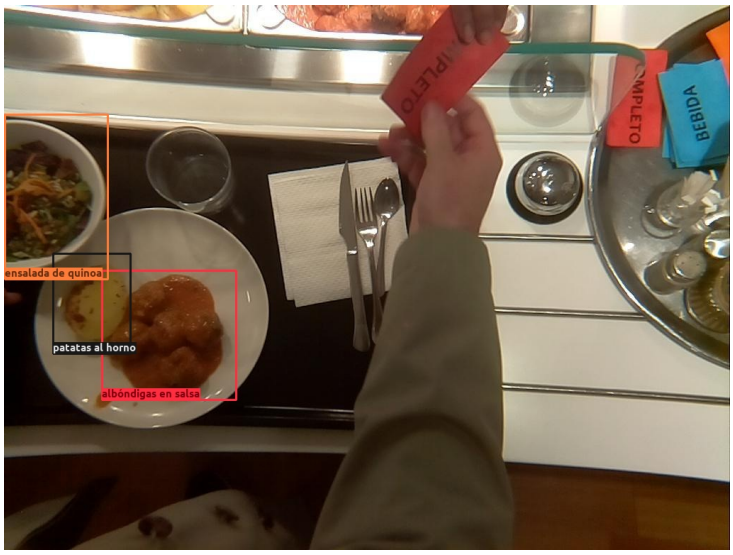
(B) MFTR and MFTR+YOLO ensemble

FIGURE 5.4: Example of food missed by baseline, but predicted by the MFTR and ensemble models. *Arroz con leche* and *empanadillas* in this example.



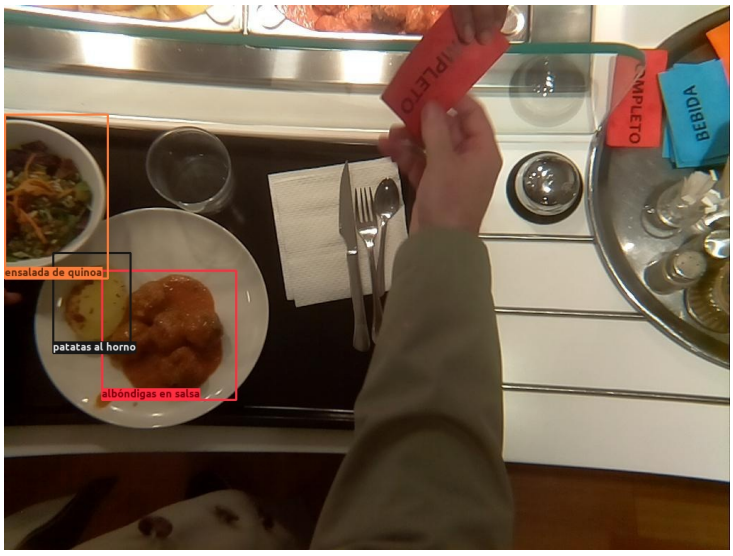
(A) Baseline, MFTR and ensemble models

FIGURE 5.5: Example of food that, even though not being in the menu, has been predicted correctly. *Librito de york y queso* in this example.



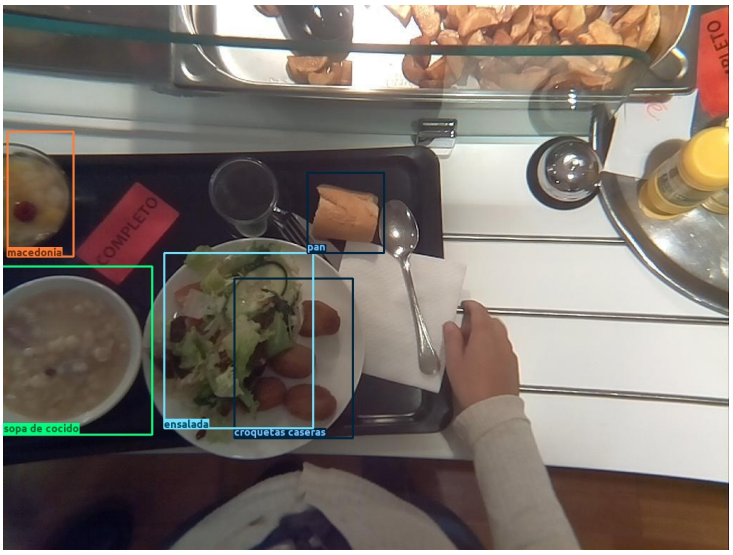
(A) Baseline model

(B) MFTR model



(C) MFTR+YOLO ensemble

FIGURE 5.6: Example of wrong prediction by MFTR but corrected by the ensemble

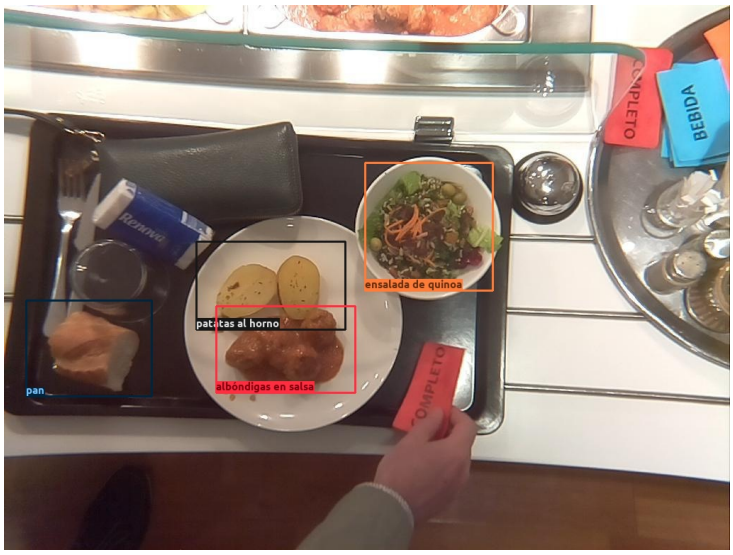


(A) Baseline model

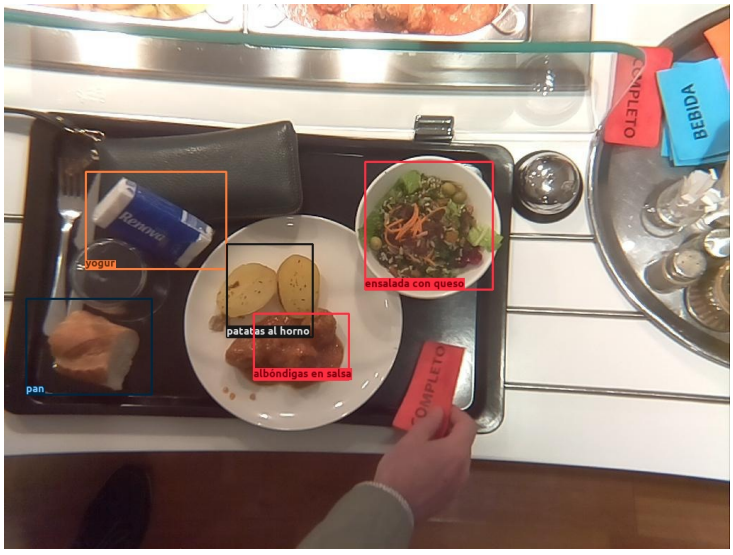
(B) MFTR model

(C) MFTR+YOLO ensemble

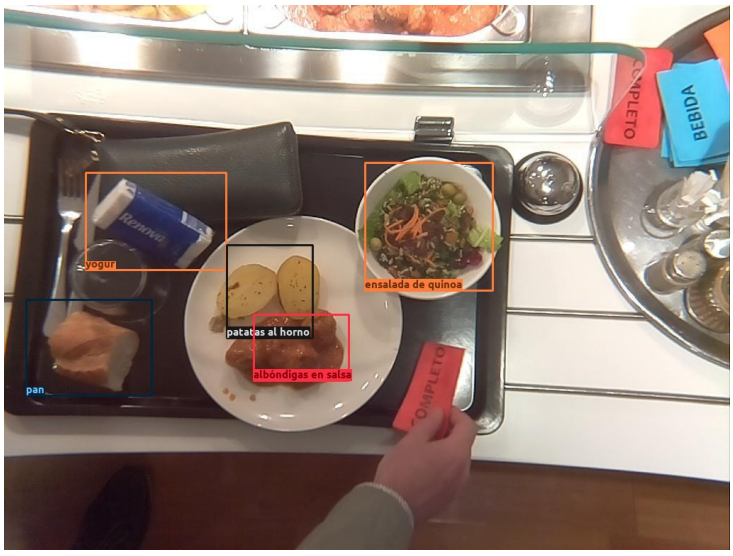
FIGURE 5.7: Example of food predicted by Baseline, but missed by MFTR and ensemble models. *Sopa de cocido* in this example.



(A) Baseline model



(B) MFTR model



(C) MFTR+YOLO ensemble

FIGURE 5.8: Example of wrong prediction by MFTR and ensemble models.

5.6 Limitations

The limitations of our models, showed in Fig. 5.7 and Fig. 5.8, are the principal ones due to the dataset. It contains a large number of different food classes but at the same time lacks in amount of data per class. Deep learning models, in order to generalize well, they need to be provided with multiple diverse examples for each class during the training process. This dataset contains classes with not enough examples and most cases repeated, due to the nature of the dataset. The dataset was created from a real restaurant with images captured automatically (Bora, Bolaños, and Radeva, 2020). Those dataset problems, despite making it more close to a real scenario, present some limitations to models that may be considered.

Another problem we found in the dataset is that it contains wrong annotations, making it harder to the model to learn during training and obtain worse measures when evaluating. The causes of the wrong annotations are the human interference and the close similarity between classes, which at the same time opens a new problem: class merging. In Fig. 5.8, apart from the wrong detection of a pack of tissues, MFTR models predicts *ensalada con queso* instead of *ensalada de quinoa*, which are considered as different classes despite of the visual similarity.

Class merging is not an easy problem. In some cases merging some classes can make the model to perform better because of the visual similarity and the incremented amount of data for the class as a result. However, in other occasions the little differences between merged classes can make the model to generalize too much, giving as a result wrong predictions to completely different food with little resemblance. In particular, in the food domain the class merging problem is even more delicate due to different presentations and recipes of a same dish, such as salads or meat with sauces.

Chapter 6

Conclusions

The hypothesis of this work was that, in the context of self-service restaurants, a multimodal model which combines an image of the whole food tray and a list of dishes on the daily menu is able to achieve better results in food dishes detection and recognition than a state of the art object detection model. Consequently, our objective was to develop such a model and test it in a real scenario.

The model we propose in this work, which we have named *Multi-modal Food Tray Recognition* (MFTR) is based on the state of the art object detection model *YOLOv3*. Starting with *YOLO* and Non-Maximum Suppression filtering algorithm, our model gets from the food tray image a set of proposal food bounding boxes and a set of corresponding early class label predictions. After that, a state of the art classification CNN, *InceptionResNetV2*, is applied to the bounding boxes in order to extract visual features of those. Finally, a novel multimodal deep neural network combines the visual features, the early predictions of the object detection model and the daily menu, giving a final set of food bounding boxes and the corresponding class label predictions.

In this work we have also presented an ensemble model of *YOLO* and *MFTR*. The ensemble is decision-based and aims to combine *YOLO* and *MFTR* predictions in order to make better ones from the best of each.

We have trained and tested our models with a real scenario dataset. The data, taken from a real self-service restaurant, was structured in days containing food tray images and the daily menus. Using this dataset, we have compared our models against a pre-trained *YOLO* model plus NMS filtering, with the best configuration of parameters found, we called it our *Baseline model*. We have measured different metrics, but the more relevant one for us is the weighted average of classes F1-scores. The Baseline model got a weighted average score of 0.8114.

Our models, *MFTR* and *MFTR* + *YOLO* ensemble, were trained and evaluated 5 times each, achieving a median of weighted average F1-score of 0.862 and 0.8774, respectively. We can see *MFTR* model is able to improve significantly the predictions compared with the baseline model and the ensemble model performs slightly better as it was expected.

In the qualitative results section we have seen that our model is good correcting the background predictions, and so discarding them, that the Baseline model wrongly generates. Also, our model is able to detect food that was missed by the baseline model and doesn't depend too much on the menu, meaning that it is able to detected food that was accidentally missed in the daily menu. Regarding

the ensemble model, we have seen that its decision-based predictions are able to correct wrong predictions made by MFTR model.

On the other hand, we have also seen examples in which our models predictions were erroneous, compared to the baseline model. We have discussed the possible causes of them, which are mostly based on the data, and the challenge it supposes to overcome those limitations in the food domain. Apart from the dataset, we consider that there are changes to the model and different approaches to explore that could lead to a better performance of our model. In the next section we will briefly describe lines of research that take our contributions as a starting point.

Chapter 7

Future Work

Once finished, with results in hand, we did an analysis of the work done and thought of possible changes in our model that could lead to better results. The first line of future work we consider is a direct change to our model that we strongly believe it would make it better. It consists of integrating part 2 of our model structure, the feature vectors extraction with a state of the art CNN, to the part 3 of our model, the multimodal neural network. That way, we could train the whole model end-to-end forcing the parameters of the CNN and the parameters of the multimodal neural network to be updated together using a linked structure and the same cost function. We believe that with an end-to-end training, the visual features extracted will be less independent and more related to the whole image and the features of the other inputs.

We have to say that, actually, integrating the CNN in the multimodal neural network was our initial idea. However, due to technical and memory usage issues, regarding the libraries used to develop our models, it hasn't been possible to implement it. In spite of the difficulty, we think that the future results may be worth the effort.

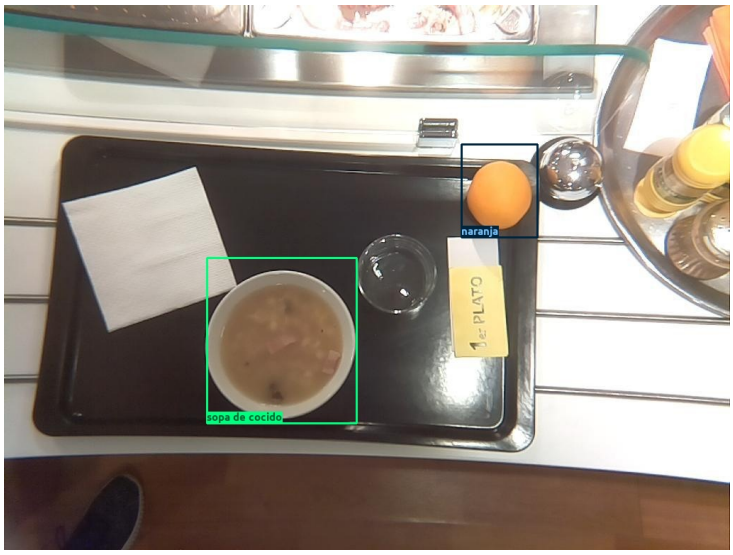
During the development process we considered and tested different variations of the model's fusions and decision statements for the ensemble. However, multimodal learning and models ensembles are two well researched fields with multiple approaches. We chose two approaches that seemed to be suitable for our problem and worked to optimize them in our context. In regard of the multimodal learning, a future work is to explore different multimodal approaches that could improve the model compared to our feature fusion-based chosen approach.

As far as the models ensemble is concerned, we have seen that even though making the per class F1-score metrics of MFTR to improve, there are certain classes in which the baseline model performs better than both, MFTR and the ensemble models. Therefore, as there is room for improvement, we consider interesting to search for better decision statements and different models ensemble approaches as future work.

Appendix A

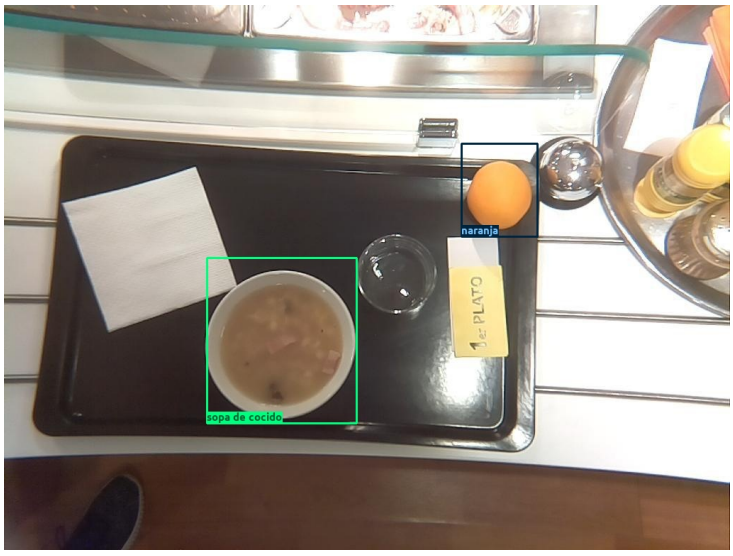
More visual results

In this appendix, we have inserted more visual examples of the results obtained with our models. The images are extra examples of particularly interesting cases discussed in 5.5.



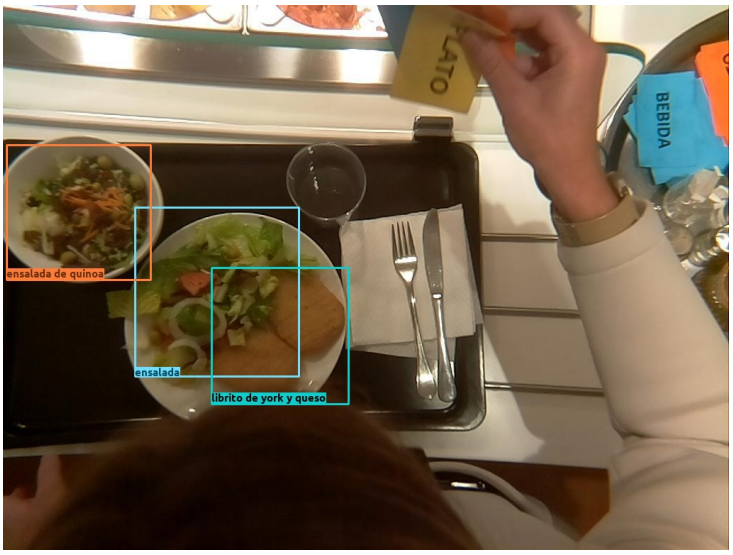
(A) Baseline model

(B) MFTR model

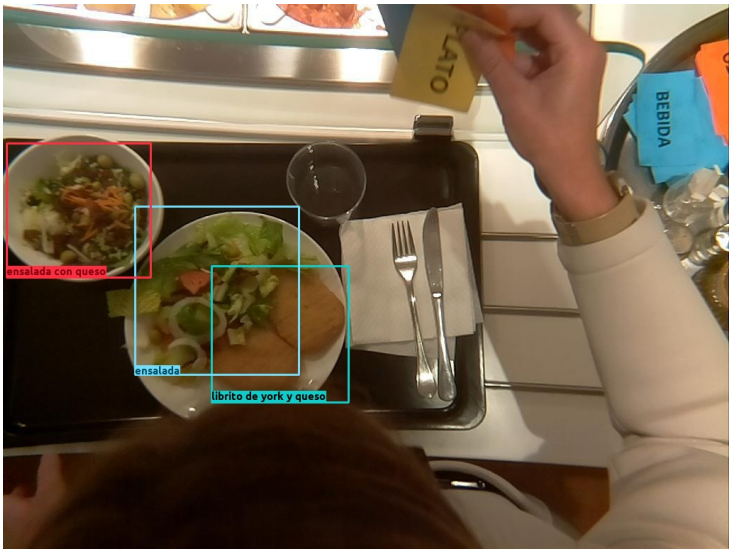


(C) MFTR+YOLO ensemble

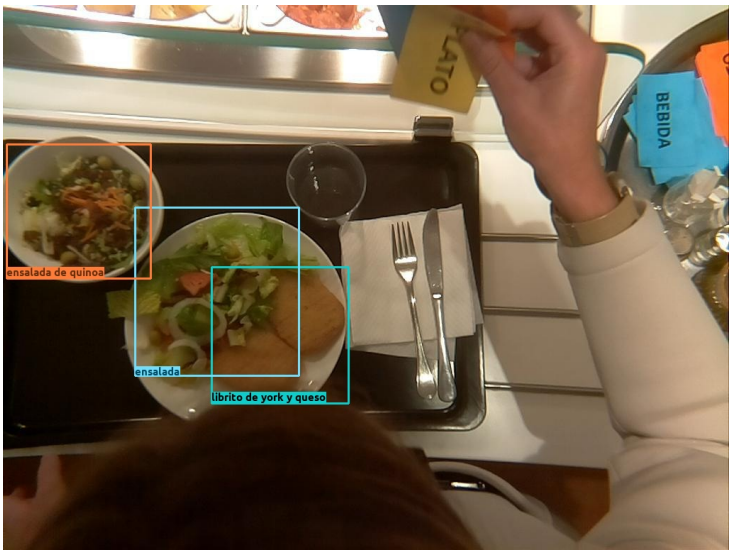
FIGURE A.1: Example of wrong prediction by MFTR but corrected by the ensemble.



(A) Baseline model

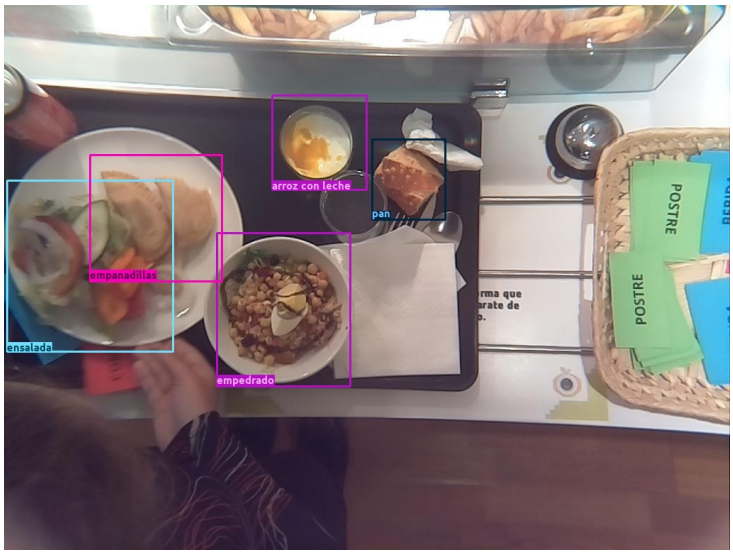


(B) MFTR model

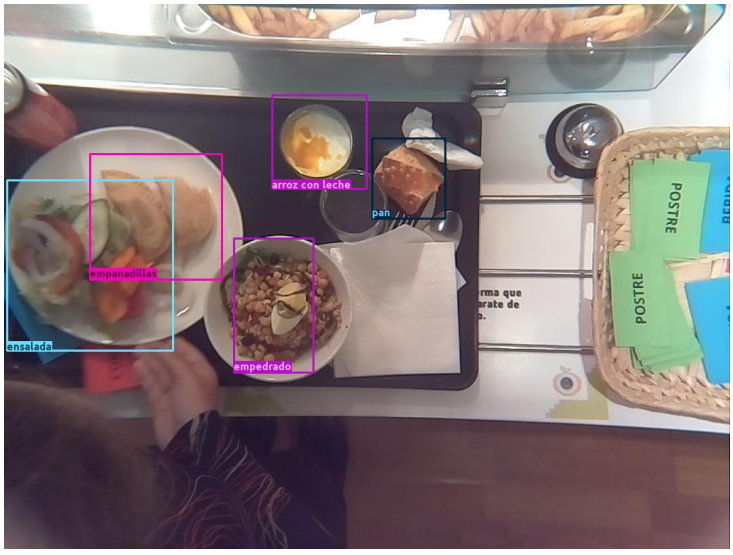


(C) MFTR+YOLO ensemble

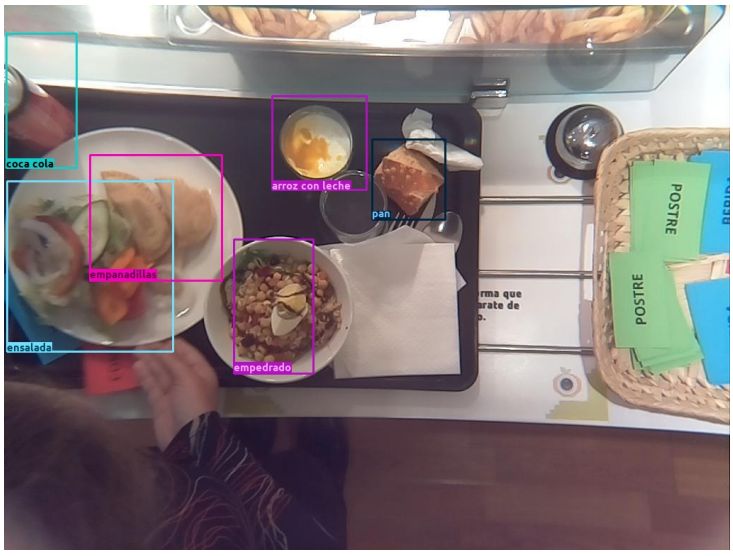
FIGURE A.2: Example of wrong prediction by MFTR but corrected by the ensemble.



(A) Baseline model

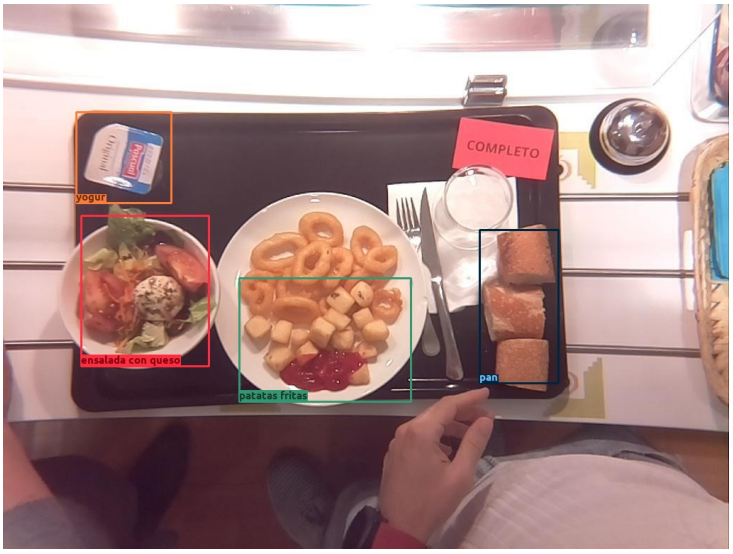


(B) MFTR model

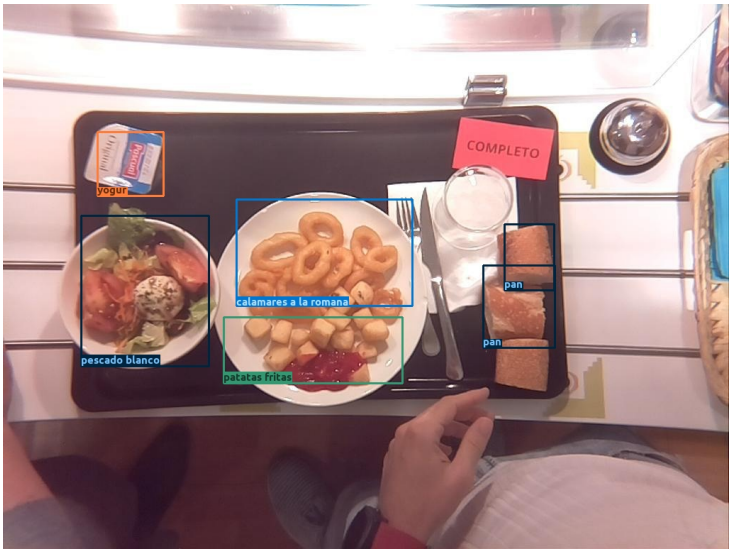


(C) MFTR+YOLO ensemble

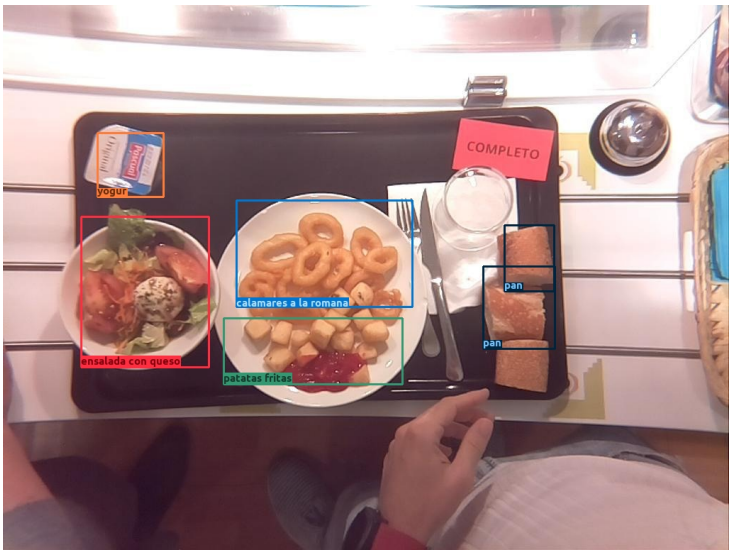
FIGURE A.3: Example of missed prediction by Baseline and MFTR but predicted correctly by the ensemble.



(A) Baseline model

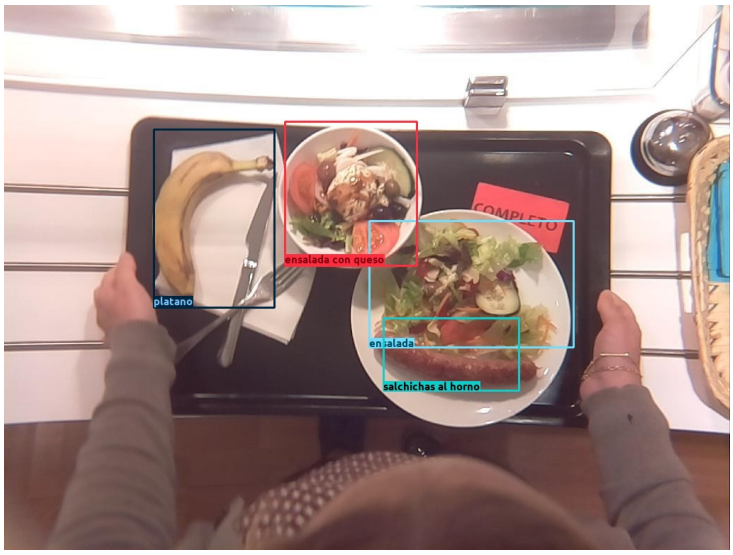


(B) MFTR model

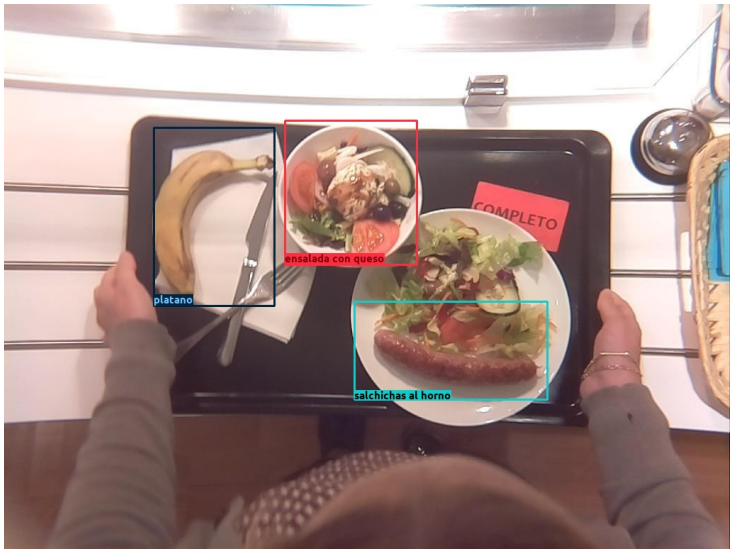


(C) MFTR+YOLO ensemble

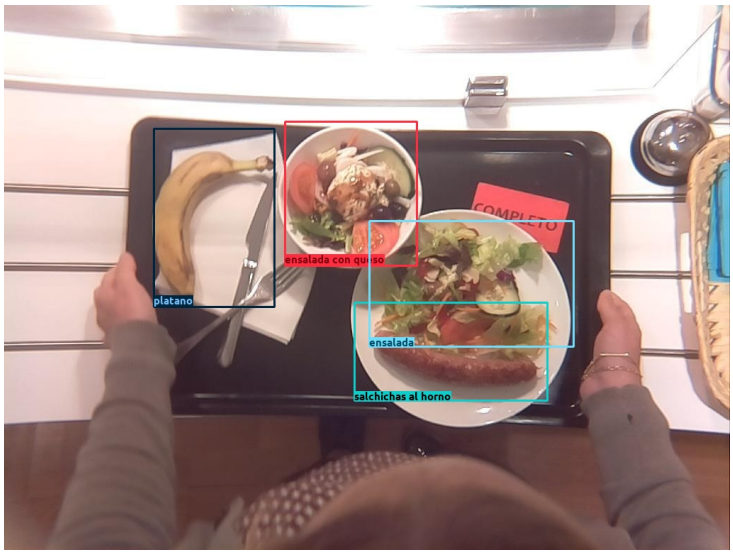
FIGURE A.4: Example of food, even though not being in the menu, has been predicted correctly. *Calamares a la romana* in this example.



(A) Baseline model

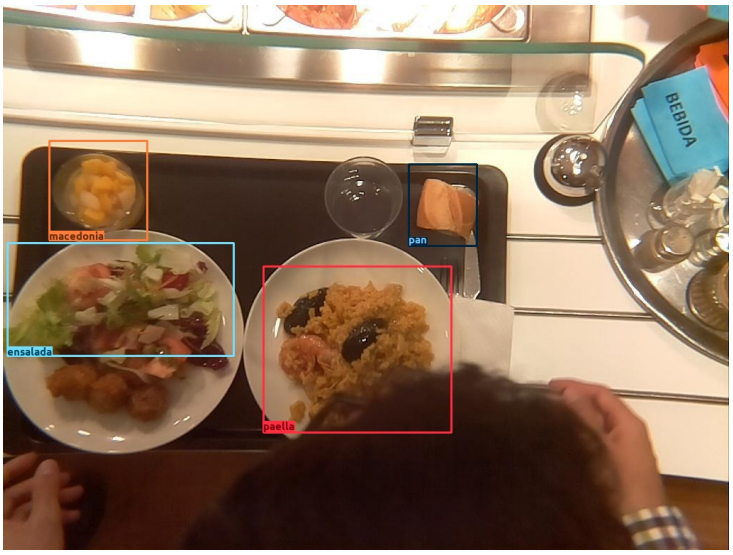


(B) MFTR model

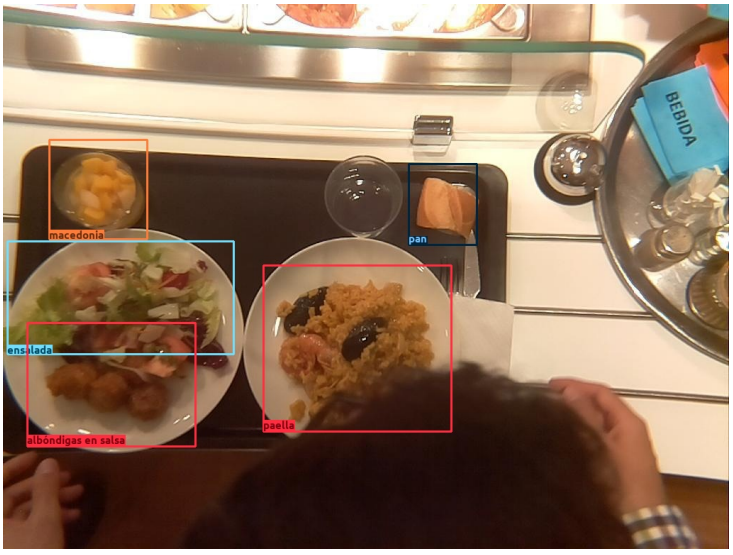


(C) MFTR+YOLO ensemble

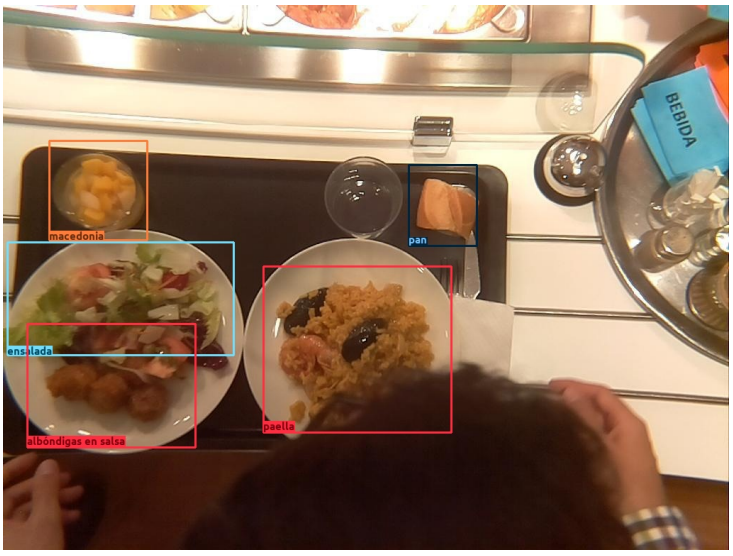
FIGURE A.5: Example of food, even though not being in the menu, has been predicted correctly. *Platano* in this example.



(A) Baseline model

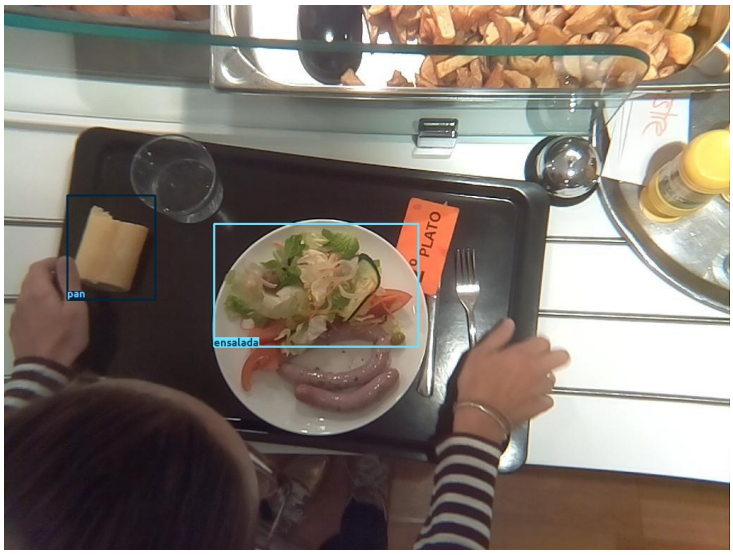


(B) MFTR model

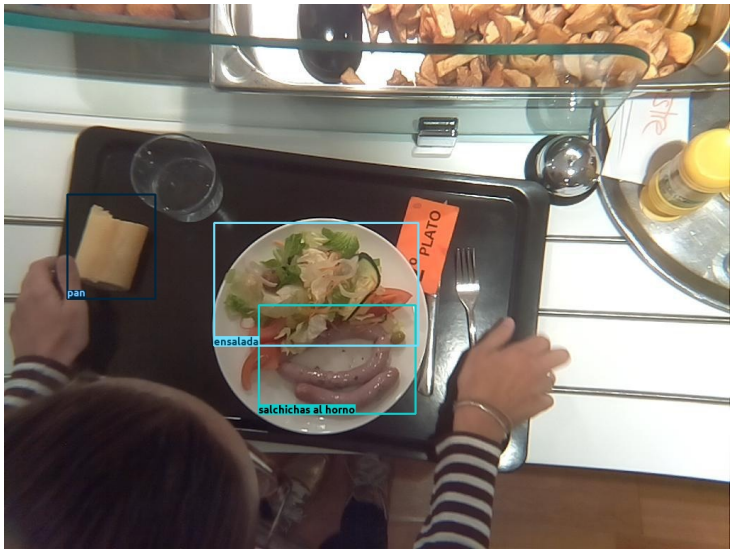


(C) MFTR+YOLO ensemble

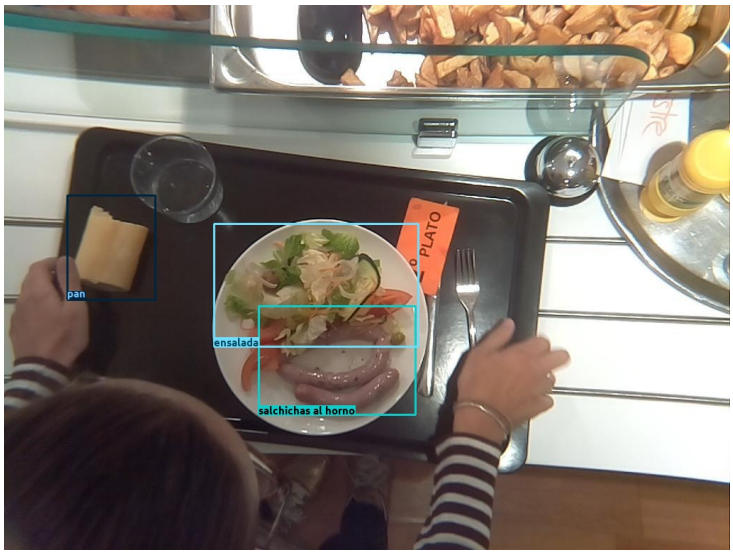
FIGURE A.6: Example of food missed by Baseline, but predicted by the MFTR and ensemble models. *Albóndigas en salsa* in this example.



(A) Baseline model



(B) MFTR model



(C) MFTR+YOLO ensemble

FIGURE A.7: Example of food missed by Baseline, but predicted by the MFTR and ensemble models. *Salchichas al horno* in this example.

Bibliography

Aguilar, Eduardo et al. (2017). *Grab, Pay and Eat: Semantic Food Detection for Smart Restaurants*. arXiv: 1711.05128 [cs.CV].

Atrey, Pradeep et al. (Nov. 2010). "Multimodal fusion for multimedia analysis: A survey". In: *Multimedia Syst.* 16, pp. 345–379. DOI: 10.1007/s00530-010-0182-0.

Beijbom, O. et al. (2015). "Menu-Match: Restaurant-Specific Food Logging from Images". In: *2015 IEEE Winter Conference on Applications of Computer Vision*, pp. 844–851.

Bettadapura, Vinay et al. (2015). "Leveraging Context to Support Automated Food Recognition in Restaurants". In: *2015 IEEE Winter Conference on Applications of Computer Vision*. DOI: 10.1109/wacv.2015.83. URL: http://dx.doi.org/10.1109/WACV.2015.83.

Bochkovskiy, Alexey, Chien-Yao Wang, and Hong-Yuan Mark Liao (2020). *YOLOv4: Optimal Speed and Accuracy of Object Detection*. arXiv: 2004.10934 [cs.CV].

Bolaños, Marc et al. (Apr. 2017). "Egocentric Video Description based on Temporally-Linked Sequences". In: *Journal of Visual Communication and Image Representation* 50. DOI: 10.1016/j.jvcir.2017.11.022.

Bora, Pritomrit Alex, Marc Bolaños, and Petia Radeva (Feb. 2020). "Smart Tray : A Deep learning application for self-checkout system". In:

Bossard, Lukas, Matthieu Guillaumin, and Luc Van Gool (2014). "Food-101 – Mining Discriminative Components with Random Forests". In: *European Conference on Computer Vision*.

Ciocca, Gianluigi, Paolo Napoletano, and Raimondo Schettini (2017). "Food recognition: a new dataset, experiments and results". In: *IEEE Journal of Biomedical and Health Informatics* 21.3, pp. 588–598. DOI: 10.1109/JBHI.2016.2636441.

Dalal, Navneet and Bill Triggs (June 2005). "Histograms of Oriented Gradients for Human Detection". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2005)* 2.

Dauphin, Yann et al. (Feb. 2015). "RMSProp and equilibrated adaptive learning rates for non-convex optimization". In: *arXiv* 35.

Deng, Jia et al. (June 2009). "ImageNet: a Large-Scale Hierarchical Image Database". In: pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

Dietterich, Thomas G. (2000). "Ensemble Methods in Machine Learning". In: *Multiple Classifier Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 1–15. ISBN: 978-3-540-45014-6.

Docker Compose. https://docs.docker.com/.

Duchi, John, Elad Hazan, and Yoram Singer (July 2011). "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *J. Mach. Learn. Res.* 12.null, 2121–2159. ISSN: 1532-4435.

Gavrila, D. M. (2000). "Pedestrian Detection from a Moving Vehicle". In: *Computer Vision — ECCV 2000*. Ed. by David Vernon. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 37–49.

Girshick, Ross (2015). *Fast R-CNN*. arXiv: 1504.08083 [cs.CV].

Girshick, Ross et al. (2013). *Rich feature hierarchies for accurate object detection and semantic segmentation*. arXiv: 1311.2524 [cs.CV].

Goh, Gabriel (2017). "Why Momentum Really Works". In: *Distill*. DOI: 10.23915/distill.00006. URL: http://distill.pub/2017/momentum.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

Gunes, H. and M. Piccardi (2005). "Affect recognition from face and body: early fusion vs. late fusion". In: *2005 IEEE International Conference on Systems, Man and Cybernetics* 4, 3437–3443 Vol. 4.

Guo, Cheng and Felix Berkhahn (2016). "Entity Embeddings of Categorical Variables". In:

He, Kaiming et al. (2015). *Deep Residual Learning for Image Recognition*. arXiv: 1512.03385 [cs.CV].

– (2016). *Identity Mappings in Deep Residual Networks*. arXiv: 1603.05027 [cs.CV].

Ioffe, Sergey and Christian Szegedy (2015). *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. arXiv: 1502.03167 [cs.LG].

Jin, Mengqi et al. (2018). *Improving Hospital Mortality Prediction with Medical Named Entities and Multimodal Learning*. arXiv: 1811.12276 [cs.CL].

Kahou, Samira Ebrahimi et al. (2015). *EmoNets: Multimodal deep learning approaches for emotion recognition in video*. arXiv: 1503.01800 [cs.LG].

Kingma, Diederik and Jimmy Ba (Dec. 2014). "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations*.

Kitamura, Keigo, Toshihiko Yamasaki, and Kiyoharu Aizawa (2009). "FoodLog: Capture, Analysis and Retrieval of Personal Food Images via Web". In: *Proceedings of the ACM Multimedia 2009 Workshop on Multimedia for Cooking and Eating Activities*. CEA '09. Beijing, China: Association for Computing Machinery, 23–30. ISBN: 9781605587639. DOI: 10.1145/1630995.1631001. URL: https://doi.org/10.1145/1630995.1631001.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey Hinton (Jan. 2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: *Neural Information Processing Systems* 25. DOI: 10.1145/3065386.

Liu, Chang et al. (2016). *DeepFood: Deep Learning-Based Food Image Recognition for Computer-Aided Dietary Assessment*. arXiv: 1606.05675 [cs.CV].

Liu, Kuan et al. (May 2018). *Learn to Combine Modalities in Multimodal Deep Learning*.

LogMeal. https://www.logmeal.es/.

Lowe, David (Jan. 2001). "Object Recognition from Local Scale-Invariant Features". In: *Proceedings of the IEEE International Conference on Computer Vision* 2.

Martinel, Niki, G.L. Foresti, and Christian Micheloni (Dec. 2016). "Wide-Slice Residual Networks for Food Recognition". In:

McAllister, Patrick et al. (2018). "Combining deep residual neural network features with supervised machine learning algorithms to classify diverse food image datasets". In: *Computers in Biology and Medicine* 95, pp. 217 –233. ISSN: 0010-4825. DOI: https://doi.org/10.1016/j.compbiomed.2018.02.008. URL: http://www.sciencedirect.com/science/article/pii/S0010482518300386.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

Moon, Seungwhan, Leonardo Neves, and Vitor Carvalho (Jan. 2018). "Multimodal Named Entity Recognition for Short Social Media Posts". In: pp. 852–860. DOI: 10.18653/v1/N18-1078.

Mroueh, Y., E. Marcheret, and V. Goel (2015). "Deep multimodal learning for Audio-Visual Speech Recognition". In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2130–2134.

Multimodal Keras Wrapper. https://github.com/MarcBS/multimodal_keras_wrapper.

N. Murthy, Venkatesh, Subhransu Maji, and R. Manmatha (June 2015). "Automatic Image Annotation using Deep Learning Representations". In: pp. 603–606. DOI: 10.1145/2671188.2749391.

Ngiam, Jiquan et al. (Jan. 2011). "Multimodal Deep Learning". In: pp. 689–696.

Pang, Jiangmiao et al. (2019). *Libra R-CNN: Towards Balanced Learning for Object Detection*. arXiv: 1904.02701 [cs.CV].

Redmon, Joseph and Ali Farhadi (2016). *YOLO9000: Better, Faster, Stronger*. arXiv: 1612.08242 [cs.CV].

– (2018). *YOLOv3: An Incremental Improvement*. arXiv: 1804.02767 [cs.CV].

Redmon, Joseph et al. (2015). *You Only Look Once: Unified, Real-Time Object Detection*. arXiv: 1506.02640 [cs.CV].

Ren, Shaoqing et al. (2015). *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. arXiv: 1506.01497 [cs.CV].

Simonyan, Karen and Andrew Zisserman (2014). *Very Deep Convolutional Networks for Large-Scale Image Recognition*. arXiv: 1409.1556 [cs.CV].

Snoek, Cees, Marcel Worring, and Arnold Smeulders (Jan. 2005). "Early versus late fusion in semantic video analysis". In: pp. 399–402. DOI: 10.1145/1101149.1101236.

Srivastava, Nitish et al. (2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56, pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

Sun, Xudong, Pengcheng Wu, and Steven C.H. Hoi (2018). "Face detection using deep learning: An improved faster RCNN approach". In: *Neurocomputing* 299, pp. 42 –50. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2018.03.030. URL: http://www.sciencedirect.com/science/article/pii/S0925231218303229.

Szegedy, Christian et al. (2014). *Going Deeper with Convolutions*. arXiv: 1409.4842 [cs.CV].

Szegedy, Christian et al. (2015). *Rethinking the Inception Architecture for Computer Vision*. arXiv: 1512.00567 [cs.CV].

Szegedy, Christian et al. (2016). *Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning*. arXiv: 1602.07261 [cs.CV].

Tensorflow. https://www.tensorflow.org/.

Tieleman, T. and Hinton (2012). "G. Lecture 6.5 - RMSProp, COURSERA: Neural Networks for Machine Learning". In:

Viola, Paul and Michael Jones (2001). "Robust Real-time Object Detection". In: *International Journal of Computer Vision*.

Wang, Zijie J. et al. *CNN Explainer Live Demo*. URL: \url{http://poloclub.github.io/cnn-explainer/}.

– (2020). "CNN Explainer: Learning Convolutional Neural Networks with Interactive Visualization". In: *IEEE Transactions on Visualization and Computer Graphics (TVCG)*.

Zoph, Barret et al. (2017). *Learning Transferable Architectures for Scalable Image Recognition*. arXiv: 1707.07012 [cs.CV].