Pizza-detector: detecting ingredients on pizza

Jordy Dal Corso

Introduction

The application of machine learning on detecting features on images has been developed much in the last years with the rise of convolutional neural networks. As many other neural networks they admit an input layer and an output layer and these networks employs the mathematical operation called convolution as part of the hidden layers. Convolution somehow mananges to summarize values of different matrix entrances reducing the dimension of the input but still keeping as much information as possible.

In the case of what we are going to see, a convolutional neural network is implemented as the core of the project. The input layer is characterized by an image, or, more precisely, the image of a pizza, while the output of the network consists in a vector of probabilities which basically tell us if each of the ingredients taken into consideration is on the pizza image.

This project is supposed to be part of something bigger. The idea comes from [1] and a natural development of this is an application that allows customers to create their own pizza while sitting at the table of a restaurant. This is done by [1] using a GAN after having introduced a detector which is similar to the one presented here.

Interactive menu is something McDonald is trying to implement with switches that toggle the presence of ingredients inside hamburgers and something similar can be seen in the company Wet Seal, which created the web app Outfitter to make people create their own outfit [2]. An interactive choice of the product by customers can reasonably lead to a better satisfaction and sense of control as a part of the "visive" culture we're living in the last decades.

User Needs, Success and Results

The main user need we aim to satisfy with this application (as a part of something bigger) can be summarized by the last lines of the introduction. Users waiting in a restaurant would enjoy to create their own product and see it before it's effectively cooked. They could see how ingredients and colours fuse together over their pizza creating good-looking ones.

This application somehow could allow us to collect the features (ingredients) that in future instances could be put over new "blank" pizza images by our customers.

So when is success reached? Clearly the best situation would be the one in which the detector correctly detects every ingredient over the pizza provided.

This is somehow costly in terms of time and resources since the dataset (which will be introduced later) only provide labels for a fixed number of ingredients. On the other hand my preparation over CNNs and intense network training has been roughly obtained over the internet so this led to defining success as a "decent" detection of a bunch (7) of fixed ingredients over pizzas.

The overall results obtained are pretty good in my opinion. This opinion is based on:

- The application correctly detects around 53% of the ingredients. Since the MIT article [1] gets around 77% over the same problem, I'm pretty satisfied since I trained the network for 10 epochs only and didn't optimized parameters layer by layer.

- My previous preparation on the topic of NNs was basically zero. Integrating lessons and internet researches I learnt something about how convolution works, how neural networks work, how to put a project on GitHub, how to use a bunch of interesting libraries (Tensorflow-Keras, Pandas, CV2, Matplotlib, Streamlit) and how to overcome problems while dealing with image formats, network training, standardization of procedures.

Summarizing what I've told so far, results from a pure utilitaristic point of view can really be improved, for example by training the network for more epochs or by choosing a better architecture to fit this particular problem but this somehow went beyond my available time and preparation. Results in terms of a personal point of view are really good since the preparation about the topic and in general programming skills I obtained from this can lead me to a deepening in these fields, maybe toward other courses or a master thesis.

Data Collection and Evaluation

In order to train a NN that can detect ingredients on pizza images, I needed a dataset with labelled pizza images. The dataset is provided by [1] and contains over 9000 labelled pizza images. The images are labelled in the sense that every pizza is associated with a binary vector (inside a .csv file). The length of the vectors is 13 and the i-th entrance of the vectors is 1 if the i-th ingredient is on the pizza, while it is 0 otherwise. The ordered list of 13 ingredient was provided in a different text file.

Since within the 13 ingredients only 7 appeared consistently on the pizza images provided, I reduced the length of the vectors to 7, keeping only the entrances related to the most common ingredients. This was easily done by a script I made myself. This reduced the number of images

too, since with another script I deleted all the images containing only uncommon ingredients.

A preprocess over the images has been done too. In particular using the library cv2 I managed to resize every image to a 299×299 (RGB channel first) because this is the standard input shape for the CNN I used.

The overall dataset quality is really good since 9000 labelled images in a zip file are not always opensource or easy to obtain. A better dataset would have probably involved more ingredients, for example wurstels, which are really common in Italy.

Mental Models

The AI-ML implementation within this project is somehow for learning purpose if not seen within the bigger project described in [1] and in the introduction. Then if we consider this project as the starting point for an interactive menu application, the usage of an AI would seem quite natural to the users and customers. They wouldn't ask how it works in depth since they would be enlightened by the facility of the usage and the funny and interesting features it brings in their life.

Explainability, Feedback and Errors

Predictions are basically 7 probability values associated to every of the ingredients involved so they are real number between 0 and 1. If a prediction is close to 1 we can say that our model predicted the associated ingredient is on the pizza provided. if a prediction is close to 0 we can say that our model predicted the ingredient is not on the pizza. Then we need to deal with the "intermediate values": for example how to deal with a 0.4?

A first way to deal with these values is to "binarize" the predictions, so turn every prediction ≥ 0.5 to 1 and 0 otherwise. This can also be how the accuracy of the model is calculated. Another way to deal with these values is the one I choose: I let predicitons as real probability numbers and suggested an "open" interpretation to the users then if you see a 0.5 you can think both "Ok, this is somehow far from 0 so the ingredient is probabily there" or "This is between 0 and 1 so the model is not confident". I somehow prefer the first interpretation since from what I have seen so far if a prediction is over 0.3 the ingredient or something similar is probably on the pizza.

If we want to stick to the "binarization", optimizing the thresholds for which a prediction is set to be 0 or 1 could also be a big improvement to this project since some ingredients tend to return lower values than others in case of their presence over the pizza.

Another big improvement would be a system which allows the user to tell the model if it is wrong and give the right answer. If these feedbacks were integrated in the model this would clearly lead to a better precision.

The last big improvement I would cite is the adding of another model which effectively detects if the image provided is a pizza or not. This is the easiest improvement to implement but I didn't managed to carry this out due to a matter of time.

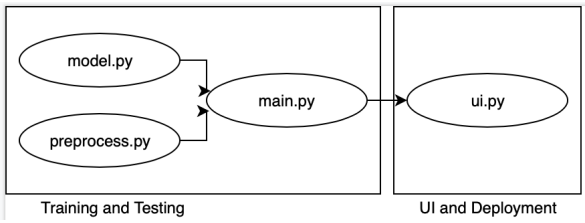
Notice that errors (like providing a non-pizza image or missing predictions) are not "deadly" since as I told before this is something like a learning project and this should be implemented in a more complex interactive menu application only if the accuracy is improved and all the possible problems and exceptions are treated.

Brief code summary

This section is only a big summary of how the code works. I highly recommend to integrate this piece of text with:

- The code documentation. Not only I wrote the docstrings related to classes and function but I also tried to explain every line of the scripts I used.
- The diary, in which I briefly wrote how the idea and the development evolved

The project is written in Python. The basic idea was to preprocess the labelled images and use them to train a CNN which detects ingredients on pizzas. After saving the parameters of the trained CNN, the model imported and used as the core of an user interface which allows users to load pizza images and let the model make predictions about the ingredients over it.



There are 4 important Python scripts which are the core of the project:

- **preprocess.py** is the class which characterizes preprocess objects, which are basically sets of resized images and associated labels. These preprocess objects are the one used to train the CNN. They are used by the function *train()* which is inside the *model* class.
- **model.py** is the class which allows to create and train the CNN model (InceptionV3) used to predict the ingredients. The class *model* provide useful functions to train, test and save a model.
- **main.py** is the script that consists in a sequence of instructions used to preprocess images, train a model and save it in order for it to be reused by the user interface. It mainly uses the classes *preprocess* and *model*.
- **ui.py** consists in the user interface, which is built with Streamlit. The UI is very straightforward since it asks the user for a pizza image and after a brief computation returns the prediction about the 7 ingredients involved by the model.

Bibliography

[1] http://pizzagan.csail.mit.edu
[2] Daft, Organization Theory and Design