COMPUTER VISION PROJECT

Food Recognition and Leftover Estimation

Pietro Girotto 2088245
Niccolò Tesser 2088246
Enrico D'Alberton 2093708

Pd, 20/07/2023

INDEX

The Project

In this project the main objective is to make an estimation of food's leftovers, the test set images are composed by the original tray image (before the meal), and a group of three leftover trays images (after the meal), the masks found and the respective bonding boxes used as ground truth for the metric calculation.

At the beginning of the project we thought about not to use Neural Networks and try with classical Computer Vision techniques because of the time needed to collect a good training set (from the Piovego cafeteria combined with other sources) and for the training phase. However, the final common decision was to use the Network for the classification task for matching the "before" and "after" meal. This was justified by the fact that fine tuning classical CV methods for such a task would be more time consuming than training a NN network.

Strategy used

The project has been split into four subproblems:

- Segmentation of single dishes from tray
- Segmentation of bread and salad from the tray
- Classification of single extracted dishes
- Metrics calculation

The overall workflow for the model is as follows:

1. Detect plates and segment them
2. For each plate segment each food present
3. Find and segment salad and bread (if present) from the original tray image
4. Feed the segmented images to the neural model for classification
5. Repeat for the second image to match foods and estimate leftovers

Assigned tasks

Pietro Girotto: creation and testing of the classification task by use of neural networks. The source files are linked in the report and available in the project folder under src/model.

Nicolò Tesser: segmentation of dishes and respective mask generation using classic OpenCV methods, source files are inside the src/dish_detector folder.

Enrico D'Alberton: segmentation of bread and salad and respective mask generation using classic OpenCV methods, source files created are inside src/bread_detector and src/salad_detector.

The metrics calculation was left as the last step because we focused on the main task of segmentation and classification because we thought that it has been useless to have metrics calculation before having good masks for the food and a working Neural Network.

Segmentation Techniques

The problem of segmentation in Computer Vision is the task of splitting an image in meaningful subregions, in this project the aim was to segment dishes from trays (like fish cutlet, potatoes, bread, salad and so on).

Dish segmentation (Nicolò Tesser)

Introduction to the problem

The dish segmentation task of the project was made by using customized functions exploiting OpenCV's provided functions. The main problem of segmentation is that I have to find a global method, domain dependent, to extract the correct regions containing different foods.

Methods

The task was divided into subtasks, the idea for extracting single dishes from a tray is composed by three main steps:

- extract plates from a tray using HoughCircles function from OpenCV with customized parameters;
- remove the white part of the plate to have an area that contains only food;
- if necessary split dish from dish in the same food area;

Plates extraction

The first step was the simplest one, by exploiting HoughCircles function with custom parameters like min_radius and max_radius, it was pretty easy to find the correct masks for the plates, once done the area around the found plate was cropped for further processing. Some of the results:

The whole generated dishes can be found in single_plates folder.

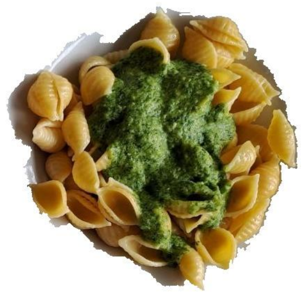
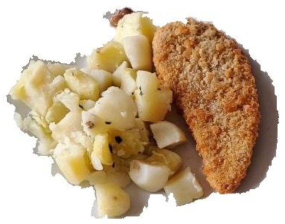
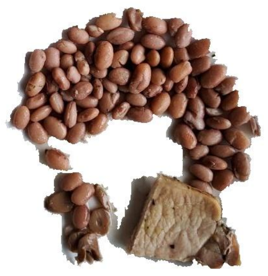
Food area extraction

The second step was particularly tricky and cost a lot of time: a large amount of time was spent in order to find the best segmentation technique.
To extract food area a lot of methods was tested, some methods coming from OpenCV implemented functions and some other custom, the most promising methods tested were:

- Color filters in RGB and HSV color space
- Watershed

- Grab Cut

The first technique tested was GrabCut, a graph based segmentation method for automatic foreground extraction inside a rectangular area, but it was very slow regarding the computational time and the results weren't good enough for justifying the large amount of time spent to extract the food area (foreground).
After hours of tests the choice was to move to Watershed segmentation, the problem was that Watershed needs input markers for the food area and for the background area, these markers can be easily defined by user input but this wasn't the case, so I have to find a method to define these markers.
Firstly I tried to use the area returned from GrabCut as markers for Watershed but the performance was too unbalanced comparing the initial tray with a huge area of food and the leftover tray with a small amount of food area, moreover this test didn't resolve the computational time problem.
So I moved to the faster techniques based on color filters, trying RGB, HLS and HSV color spaces. This test was promising at the beginning but after a lot of tests I noticed that finding a global threshold (better to say a global range) value was very difficult because of the similarity of some foods colors and the background, so this method was left.
Finally I decided to use Canny edge detector to have a meaningful set of markers for food area, the background area markers was defined using a custom circular mask (the area out of the circular mask was set as background markers) and the uncertainty area was create by dilating the seed found with canny, it is in this area that watershed can make its calculation and derive the watershed lines that split the image in different areas.
The results are better in the initial food images than leftover ones but they are better than the other cited methods so this was the final choice. Some of the obtained results:

The whole generated dishes can be found in single_dishes folder.

Dish separation

This was the hardest part of my tasks, the results aren't as I hoped. This task was full of difficulties and to solve the issue I tried to put my whole knowledge, from K-Means to MeanShift filtering, color filters in RGB and HSV color spaces, but I didn't reach an acceptable result. The dish splitting was not necessary in all the images so the network received some "ideal" input (like pasta and other single segmented dishes) and other non-ideals in which there was more than one food in the same area. The idea that I've tried to implement was that by using the pyrMeanShiftFilter from OpenCV or similarly a BilaterlFilter, the images looks like cartoonish and so the colors are more uniform in a given area, then I tried to use a combination of multiple HSV color filter based on reference values to isolate the searched dish but these filters are very difficult to adjust in their threshold and obviously, depending on the application domain, they could overlap in some cases (like the color of potatoes is similar to the color of pasta, so not knowing the true class is difficult to separate based only on this knowledge). I’ve also tried with a huge GaussianBlur and then a prototype of MSER for blob detection but the results weren’t good enough. The last chance I had in mind was template matching, I’ve implemented a method that search for template in the image obtained in the previous step, this method seemed to work and matched region is often in the right place, but sometimes it returns a completely different location, moreover the number of false positive matches was huge and so after different tries I left also this technique. The code is in the dish_detector.cpp file anyway, just to show that an idea has been implemented, also template images could be found in src/dish_detector/template. We decided not to use the function because the false positive leads to complete wrong masks and nullifies the mask previously obtained.

Considerations

I think that the real life application of this project could be done in a more controlled environment so that some of the difficulties found could be better solved, like the position of the camera that acquires the tray’s images could be put in at a fixed distance in a way that all dishes has the same diameter (in pixel), the white color of the plate could be more uniform and other advantages. I think that my approach could better perform in such a scenario, but the aim of the project is to test our ability in solving these problems and provide a robust solution to different situations of pose, occlusion and illumination.

Salad Segmentation (Enrico D'Alberton)

Introduction

The saladDetector function is designed to extract the salad from a tray of food using computer vision techniques and image processing algorithms seen during the lessons. The function takes an input image of the tray, identifies the salad area, and creates a binary mask representing the salad region. Additionally, it computes the bounding box around the detected salad and stores it in a vector along with the confidence value of the detection.

Function Overview

The function can be divided into several main steps:

Dish Detection: the function starts by detecting the dish or the circular tray using the Hough Circle Transform. It marks the circular region found in the tray image and sets the pixels outside it to black. This is important for the breadDetector, since it removes false positives.

Color Filtering: the function then applies a color filter to the tray image to extract the salad region. It uses a set of predefined colors and their respective thresholds to create a binary mask representing the salad area. The colors have been extracted from the images of the salads.

Morphological Operations: to remove noise and refine the salad region, morphological operations such as erosion and dilation are applied to the binary mask.

Watershed Algorithm: the watershed algorithm is used to segment the salad region further. It separates connected regions and ensures that the salad area is distinct from other elements in the tray.

Bounding Box Extraction: the function calculates the bounding box around the detected salad region and stores its coordinates and dimensions in a vector.

Confidence Evaluation: the function assigns a confidence value of 1.0 to the extracted salad, since the algorithm for sure will extract the salad if there is one.

Salad Mask Creation: finally, the function creates a mask in the cmp_tray_mask image, representing the salad region with a specific label (in this case, 12 is used as the label for the salad category).

Improvements and Considerations

While the function performs reasonably well, there are several improvements and considerations that can be made:

Parameter Tuning: some parameters used in the function, such as the color thresholds and morphological kernel size, are variable. Tuning these parameters for specific scenarios can improve the function's robustness and accuracy.

Generalization: the function's color filter and other parameters are tailored to specific salad colors. To make the function more general, it could be adapted to handle a wider range of salad types with different colors.

Bread Segmentation (Enrico D'Alberton)

Introduction

The breadDetector is divided into two separate functions: breadDetectorFullTray and breadDetectorEmptyTray. The difference is in the discriminant method used. In fact, in the breadDetectorEmptyTray, since multiple bounding boxes are computed, I decided to pick the one that has the best confidence with the class "Bread" when passing the bounding box to the model.

Methodology

Let's break down the main steps and methodologies used in the function:

Color Removal

The function starts by removing specific colors from the input tray image. It creates a vector of BGR colors to avoid, and for each color, it iterates through the entire image and sets the pixels of that color to black (0,0,0). This step helps in reducing unwanted colors that might interfere with subsequent processing (like the post-it to take the dessert).

Color Space Conversion and Enhancement

The function converts the modified tray image to the HSV color space. After the conversion, it enhances the intensity values of all three channels (Hue, Saturation, and Value) by 10%. This step is performed to make the subsequent segmentation more effective and robust.

Mask Generation

The function then uses the inRange function to create a mask based on a specific HSV range. This mask isolates the regions in the tray image that are likely to contain bread. The lower and upper HSV thresholds (cv::Scalar(13, 110, 158) and cv::Scalar(47, 255, 255)) are selected to cover a range of green colors that typically represent the bread.

Noise Removal

The mask obtained in the previous step may contain some noise. To remove the noise, the function performs morphological operations (erosion and dilation) on the mask.

Watershed Segmentation

The watershed algorithm is applied to the tray image using the mask obtained in the previous step. Watershed segmentation helps in separating different connected regions in the image based on the gradient of the intensity values. This step separates the bread region from other regions in the tray.

Contour Analysis

The function finds the contours in the resulting mask obtained from the watershed. Contours are the boundaries of connected components in the image. The function then iterates through the contours and removes the ones that are too small to be considered as bread.

Second Watershed Segmentation

After filtering out small contours, the function applies watershed segmentation again using the filtered mask. This step refines the segmentation of the bread region.

Extract Bread Region

Finally, the function extracts the bread region from the tray using the obtained mask. It applies the mask to the original tray image to get the region of interest (ROI) containing the bread. It also applies additional filtering to the ROI using mean shift filtering and color filtering to further refine the bread extraction.

Output

The function outputs the bounding box of the extracted bread region, as well as a confidence value for the correctness of the bread detection. This value is treated differently from the two functions. If breadDetectorFullTray finds bread, it always considers it with confidence 1 since the function does not use the model to distinguish the bread. Instead, the breadDetectorEmptyTray return the confidence to the "bread" class returned by the model.

Classification Task (Pietro Girotto)

Introduction to the problem

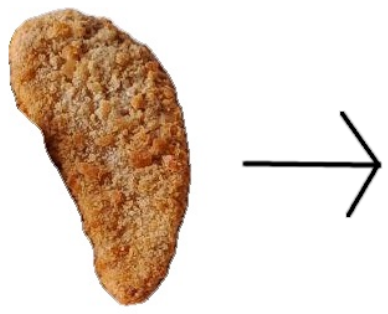
The project as a whole, heavily relies on the successful classification of the segmented foods in order to match before and after masks. To do so multiple ideas were tested against the data provided:

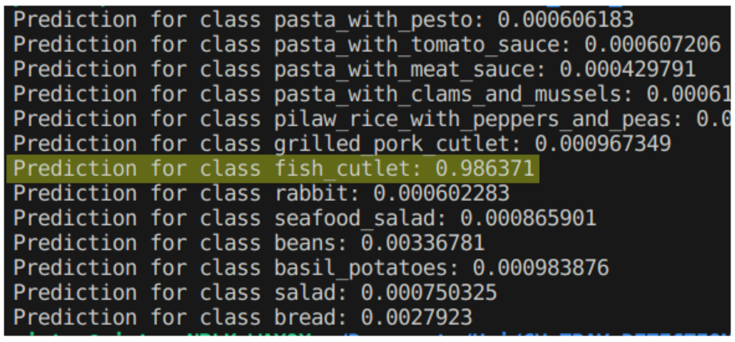
- **Color based matching**: trying to match elements based on color and other useful features of the segmented image. This proved to be heavily unreliable and difficult to implement without explicitly fitting the testing images.
- **CLIP zero shot segmentation** (Radford et al.): using a pre-trained model developed by OpenAI that was capable of classifying images into categories, given a list of text prompts, one per category. However, this model had far from acceptable performances, as in the test set had an accuracy of just 30%
- **Fine tuned ResNet** (He et al.): Fine tuning a ResNet instance on a custom dataset and using it for image classification. This proved to be the final choice, with an accuracy of over 80% on the test set.

As mentioned before, ResNet was eventually the model chosen. This meant that training and validation datasets were to be created.

Dataset Creation

The dataset creation underwent multiple iterations, leading to better and better performances.
The leading idea was to input a segmented image and output an array of probabilities-alike values for each and every food category.





```
Prediction for class pasta_with_pesto: 0.000606183
Prediction for class pasta_with_tomato_sauce: 0.000607206
Prediction for class pasta_with_meat_sauce: 0.000429791
Prediction for class pasta_with_clams_and_mussels: 0.00061
Prediction for class pilaw_rice_with_peppers_and_peas: 0.0
Prediction for class grilled_pork_cutlet: 0.000967349
Prediction for class fish_cutlet: 0.986371
Prediction for class rabbit: 0.000602283
Prediction for class seafood_salad: 0.000865901
Prediction for class beans: 0.00336781
Prediction for class basil_potatoes: 0.000983876
Prediction for class salad: 0.000750325
Prediction for class bread: 0.0027923
```

Above is an example of image prediction.
The issue was indeed to find and collect images that resembled somewhat the test set, still retaining generality and avoiding overfitting.
To do so the final dataset was assembled as follows:

1. Find Google Images query that resembled the categories requested, downloading each and every image in the query (around 700 per class).
2. Filter out unrelated images or deceptive samples (retaining 90-280 images per class).
3. Manually crop each image in such a way that the food is the main object of the photo.

4. Artificially boost the number of samples to 301 images per class by applying random affine transformations and flipping to the original images.

The final dataset can be found here.
Each iteration corresponds to an enhancement in the model's performance (see evaluation for numerical results).

Model Architecture and Training

The final model was based on ResNet34. Even though SotA results are performed usually with ResNet50, such a model is far too complex for a small classification task with just 13 total labels. In order to avoid overfitting it was used a smaller instance coupled with a final fully connected two layer for classification purposes.
Specifically, only the last layer of ResNet34 and the classification layer were actually fine tuned. The probabilities-alike values were computed with softmax on the embeddings of the last classification layer. To further avoid overfitting and making the overall model more robust, dropout normalization was used during training.

Training was done with 8 epochs, using the Cross Entropy Loss and a learning rate of *3e-5*.
Such parameters were found to be optimal by trial and error.

The model was eventually tested on the test set, and finally exported to a suitable format for importing it in C++ through the libtorch library (The Linux Foundation).
Have a look at the python notebook used for training here.

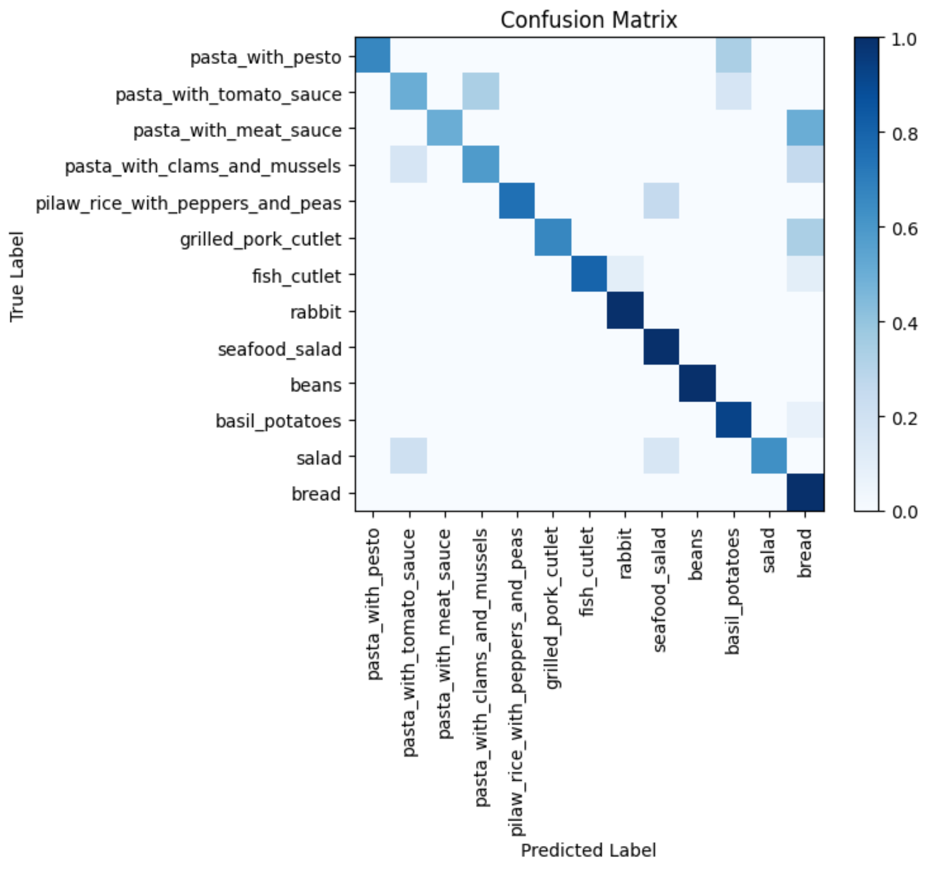
NN Evaluation and Results

To better understand the model's performances and decide whether to update the current version or not three main metrics were mainly used:

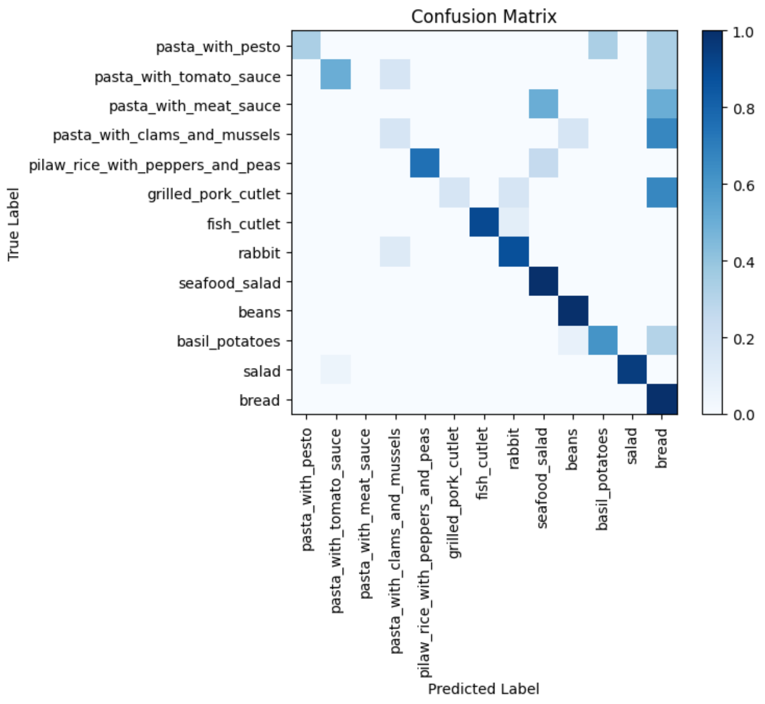
- **Accuracy**: The accuracy computed as $\frac{\#\ true\ label\ is\ the\ most\ probable}{\#\ samples\ in\ the\ testing\ \ set}$ was the main metric. Such a score is important to correctly classify foods.
- **Average Position**: Introducing the average position per label as the average rank of the true prediction, the average position is the average position across all labels. The lower it is the less uncertain the model is towards the true class.
- **Confusion matrix**: The confusion matrix was a clear representation of the model difficulties and strengths.

DATASET	ACCURACY	AVG POS
Raw	0.679	1.91
Filtered	0.723	2.04
Cropped & Augmented	0.795	1.55

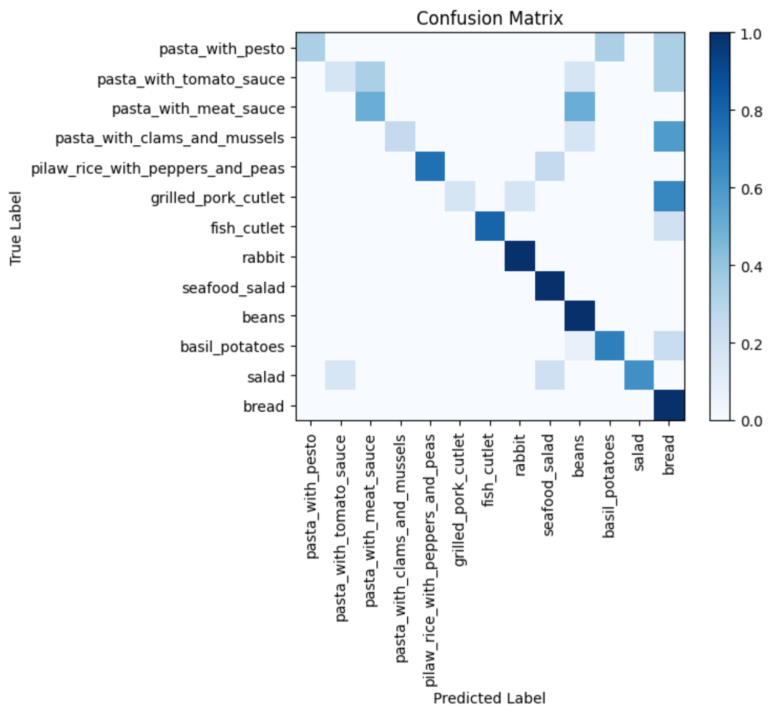
Cropped and Augmented:



Raw:



Filtered:



The model clearly well understood the task and how to solve it. Even though an accuracy of 0.8 is not comparable with SotA results, still given the fact that the training set was sampled from the internet the results were far more than what we expected.

The source code for those passages can be found in the project folder under the folder “python_code”.

Results

Defined Metrics

Three metrics has been used:

- Mean Intersection over Union (mIoU): this metric takes into account the comparison between ground truth bounding boxes and computed ones. It is the average of the IoU computed for each food item;
- Mean Average Precision (MAP): this metric takes into account the Average Precision for each food category and it computes the mean of them among all the categories.
- Leftover estimation: this metric is defined as the ratio Ri (in terms of pixels) between the segmentation mask in the “after” image and the segmentation mask in the “before” image

Numerical results

MAP: 0.39

Images Results

Analysis

Challenges and problems

The task was not so difficult as a concept however, the amount of fine tuning and thoughtfully decisions that had to be made in order to accomplish measurable results added up pretty quickly.

- **For plates segmentation:** the major difficulty found in the dish segmentation was, as previously said, the dish splitting into single food, a lot of hours and energy was spent but the best obtained result was not enough, also eliminating the white part of plates was very tricky because of the strong similarity between different foods (mainly in color) and some occlusion on the plates.
- **For classification:** The main difficulty was to be found in the dataset selection. It was both time consuming and challenging. Fine tuning hyperparameters and analyzing model performances was the second most challenging task.
- **For bread and salad segmentation:** for the salad detection I encountered many problems since the plate containing it was transparent. This means that the background was not uniform as the one for the dishes.
  For the bread segmentation I encountered some problems since the color of the tray was very similar to the bread and often I ended up with random bounding boxes.

Conclusions and future work

This project presented some issues like the suboptimality of both the generated masks and the classification, for various reasons, given these points we can think of a way to improve the performances like using a neural approach for segmentation or creating a model more robust to noise and with even better performances. This could be realistically possible by sampling a large amount of real world photos from the Piovego Canteen in order to be more accurate in the representation of the domain.

Working with a mixture of classical Computer Vision and Neural Network approaches is a good way to exploit the good features of both approaches and generate high reliable and fast Computer Vision systems.

Miscellaneous

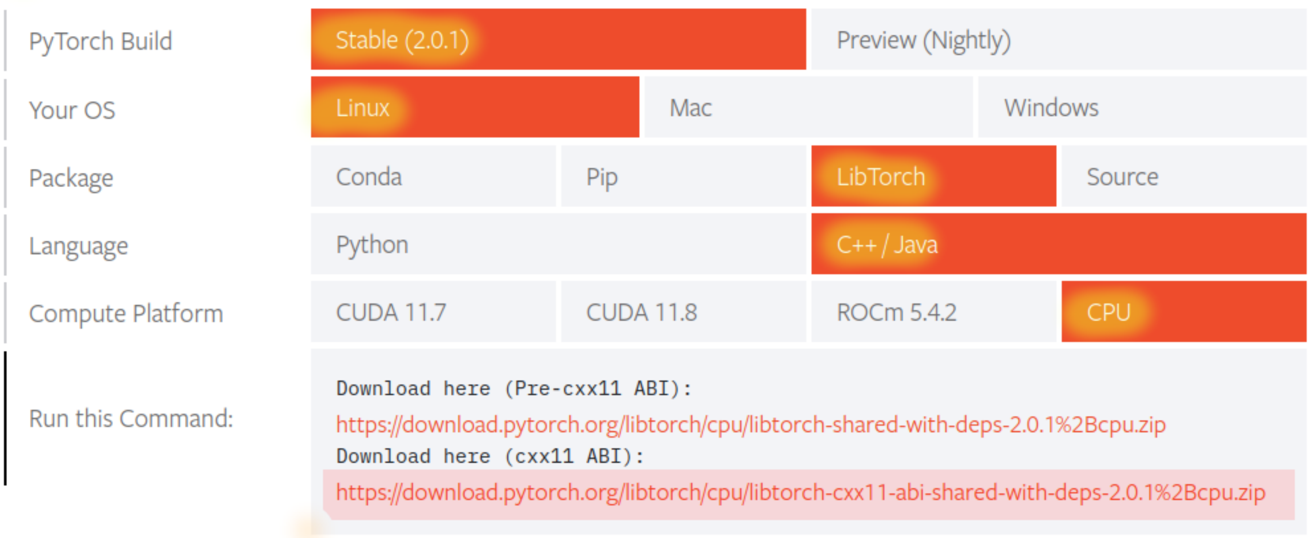
Usage

Building

The program needs essentially two libraries to be installed:

- **OpenCV 4** (or greater)
- **LibTorch** (cxx11 ABI)

Be aware that for compatibility reasons and ease of use the neural network is meant to be used on CPU only. This image should clarify which version of LibTorch should be downloaded:



PyTorch Build	[✓ Stable (2.0.1)]	Preview (Nightly)		
Your OS	[✓ Linux]	Mac	Windows	
Package	Conda	Pip	[✓ LibTorch]	Source
Language	Python	[✓ C++ / Java]		
Compute Platform	CUDA 11.7	CUDA 11.8	ROCm 5.4.2	[✓ CPU]

Run this Command:

```
Download here (Pre-cxx11 ABI):
https://download.pytorch.org/libtorch/cpu/libtorch-shared-with-deps-2.0.1%2Bcpu.zip
Download here (cxx11 ABI):
https://download.pytorch.org/libtorch/cpu/libtorch-cxx11-abi-shared-with-deps-2.0.1%2Bcpu.zip
```

Make sure to download the same version, otherwise the neural network won't load or you could encounter name clashing between the two libraries.

Once downloaded libtorch, unzip the content and you have either two options:

1. Leave libtorch in any folder and specify at build time the path
2. Move the libtorch folder (just unzipped) to the main root of the project folder

For option 1:

```
mkdir build && cd build
cmake -DCMAKE_PREFIX_PATH=<path_to_libtorch> ..
cmake --build .
```

For option 2:

```
mkdir build && cd build
cmake ..
cmake --build .
```

Execution

To execute the program you just have to run the following command in /build:
./src/main ../Food_leftover_dataset/
or in general:
<path-to-executable> <path-to-entire-dataset>

Workload

	Enrico D'Alberton	**Niccolò Tesser**	**Pietro Girotto**
Total hours estimated	70h	70h	100h

Bibliography

Works Cited

He, Kaiming, et al. "Deep Residual Learning for Image Recognition." https://arxiv.org/abs/1512.03385.

The Linux Foundation. "PyTorch C++ API — PyTorch main documentation." *PyTorch*, https://pytorch.org/cppdocs/index.html#torchscript. Accessed 19 July 2023.

Radford, Alec, et al. "Learning Transferable Visual Models From Natural Language Supervision." 2021, https://arxiv.org/abs/2103.00020.